Diseño Curricular de la Carrera de Ingeniería en Sistemas de Información

Plan 2023 - Ordenanza 1877

Contenidos Mínimos s/Ordenanza

Bases de Datos

Sistema de Gestión de Bases de Datos

Arquitectura de los Sistemas de Gestión de Bases de Datos

Almacenamiento y Acceso a Datos

OBJETIVOS s/Ordenanza

Comprender los diversos modelos conceptuales de datos

COMPETENCIAS RELACIONADAS s/Ordenanza

Competencias específicas de la carrera:

CE 1.1. Especificar, proyectar y desarrollar sistemas de información.

CE 1.3. Especificar, proyectar y desarrollar software.

CE 5.1. Dirigir y controlar la implementación, operación y mantenimiento de sistemas de información, software.

Competencias genéricas de la carrera:

CG 1. Identificar, formular y resolver problemas de ingeniería.

CG 2. Concebir, diseñar y desarrollar proyectos de ingeniería.

CG 4. Utilizar de manera efectiva las técnicas y herramientas de aplicación en la ingeniería.

CG 8. Fundamentos para una actuación ética y responsable.

CG9. Fundamentos para evaluar y actuar en relación con el impacto social de su actividad profesional en el contexto global y local.

CG 10. Aprender en forma continua y autónoma.

RESULTADO DE APRENDIZAJE RELACIONADO DEFINIDO:

RA2: Implementar el modelo relacional utilizando un Sistema de Gestión de Bases de Datos Relacionales para aplicar las funciones que hacen a la administración, operación y mantenimiento de una base de datos.

INDICE

INTRODUCCIÓN

A partir de la definición de los SISTEMAS DE GESTION DE BASES DE DATOS RELACIONALES (SGBDR), de los que realizamos una introducción en la Unidad anterior, trajo consigo la necesidad de un lenguaje de base de datos que fuera suficientemente amistoso para el usuario, a la vez que adecuado para el programador y el creador de aplicaciones.

El SQL, **Lenguaje de consulta estructurado** o por sus siglas en inglés **S**tructured **Q**uery **L**anguage, es un **lenguaje declarativo**(1) que da acceso a **bases de datos relacionales** y que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo hacer:

- cambios con el fin de agregar, borrar y modificar información
- consultas con el fin de recuperar información

en una base de datos.

Se puede intercalar en los lenguajes **procedimentale**s(2) como C, Cobol, entre otros, actualmente es utilizado en los **lenguajes orientados a objetos** como Java, Python, entre otros.

(1) **Lenguajes declarativos** solo hay que indicar que se quiere hacer.

(2) **Lenguajes procedimentales**, es necesario especificar cómo hay que hacer cualquier acción sobre la BD.

El SQL es una versión evolucionada del SEQUEL (**S**tructured **E**nglish **QUE**ry Language), creado por Chamberlain y Boyce en 1974. En 1986 **ANSI** da lugar a la primera versión estándar de este lenguaje, el SQL-86 o SQL1. Al año siguiente este estándar es también adoptado por la ISO.

El **ANSI SQL** tuvo varias revisiones y agregados a lo largo del tiempo. El estándar existente actualmente representa tanto un subconjunto de las principales implementaciones comunes como un superconjunto de casi todas las implementaciones. Es decir, el núcleo del estándar consta de características que podemos encontrar virtualmente en cada implementación comercial del lenguaje.

En esta Unidad nos concentraremos en el Lenguaje de Definición de Datos que permite la definición de la Base de Datos en el SGBDR.

LENGUAJE DE DEFINICIÓN DE DATOS (DDL)

El lenguaje de definición de datos (en inglés *Data Definition Language*, o *DDL*), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Esta estructura se almacenará en el diccionario de datos.

DICCIONARIO DE DATOS

El Diccionario de Datos o catálogo, es una base de datos del sistema que contiene información sobre las bases de datos, las tablas, las vistas, los derechos de acceso, usuarios y demás elementos del motor de base de datos.

El ANSI indica que el diccionario de datos debe ser accedido a través de una serie de vistas conocidas como INFORMATION_SCHEMA.

Su finalidad es proporcionar información relativa a la estructura de las bases de datos.

Los cambios en el diccionario de datos se realizan en forma indirecta a través de los comandos DDL, accesibles a los usuarios con derecho a este tipo de comandos: específicamente el DBA (Administrador de Bases de Datos).

Los usuarios o programadores no deben tener acceso a los comandos del DDL, solo podrían tener acceso, con los permisos correspondientes, a la consulta de la estructura que se realiza a partir del comando del DML (Lenguaje de manipulación de datos): SELECT. El resto de los comandos del DML, que permiten realizar cambios, no pueden ser utilizados para alterar la estructura de las bases de datos.

OPERACIONES BÁSICAS DEL DDL

Existen cuatro operaciones básicas:

- **CREATE** ->para crear base de datos, tablas, dominios, aserciones y vistas
- **ALTER** -> para modificar tablas y dominios
- **DROP** -> para borrar base de datos, tablas, dominios aserciones y vistas
- **TRUNCATE**.-> para eliminar datos de una tabla

Los ejemplos que utilizaremos para comprender el manejo de las operaciones del DDL se encuentran en el anexo I de esta Unidad: TSL (Tintorería siempre limpia) utilizada en este apunte y Ferretería (utilizada en las diapositivas de esta unidad).

CREACIÓN Y BORRADO DE UNA BASE DE DATOS RELACIONAL

CREACIÓN

Visto que el esquema de una base de datos no es más que un conjunto de tablas el estándar nos ofrece sentencias que se concentran en la creación, la modificación y el borrado de las mismas.

Se dispone de una sentencia de creación de esquemas denominada **CREATE SCHEMA o DATABASE**.

Con la creación de esquemas podemos agrupar un conjunto de elementos de la base de datos que son propiedad de un usuario.

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[create_option] ...

create_option: [DEFAULT] {
CHARACTER SET [=] charset_name
| COLLATE [=] collation_name
| ENCRYPTION [=] {'Y' | 'N'}
}
```

También puede utilizarse la sentencia **CREATE DATABASE**

Aunque todos los parámetros de la sentencia **CREATE SCHEMA** son opcionales, como mínimo se debe dar el nombre del esquema.

Ejemplo TSL:

```
CREATE DATABASE 'ropa_siempre_limpia'
```

En MySQL debe registrarse la Base de Datos para que sea accesible por un usuario

BORRADO

La sentencia para el borrado de una base de datos presenta la siguiente sintaxis:

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

Ejemplo TSL:

```
DROP DATABASE 'ropa_siempre_limpia'
```

CREACIÓN DE TABLAS

La estructura de almacenamiento de los datos del modelo relacional son las tablas

```
CREATE TABLE nombre_tabla
    ( definición_columna
    [, definición_columna...]
    [, restricciones_tabla]);
```

Donde **definición_columna** es:

nombre_columna {tipo_datos|dominio} [def_defecto] [restric_col]

A cada una de las columnas se le asigna un **tipo de datos** predefinido o bien un **dominio** definido por el usuario. También podremos dar definiciones por defecto y restricciones de columna. Una vez definidas las columnas, sólo nos quedará definir las restricciones de tabla.

Ejemplo TSL para la creación de la tabla Clientes

```
CREATE TABLE  `clientes' (
    'tipo_doc' varchar(4) NOT NULL
    `nro_doc' int(11) NOT NULL
    'nom_ape' varchar(50) NOT NULL,
    'tel' VARCHAR(2) default NULL,
    `dir`varchar(50) default NULL
```

TIPOS DE DATOS

Para cada columna tenemos que elegir entre algún dominio definido por el usuario o alguno de los tipos de datos predefinidos que se describen a continuación:

Tipo de datos: Cadena de Caracteres

Tipo	Bytes	Observaciones
CHAR(N)	N	Reserva N caracteres. La cadena podrá contener desde 0 a 255 caracteres.
VARCHAR(N)	N+1	Almacena hasta N caracteres (no reserva N caracteres - solo 1 para cadenas vacías) La cadena podrá contener desde 0 a 255 caracteres.
TYNITEXT	longitud + 1	Longitud máxima de 255 caracteres
TEXT	longitud + 2	un texto con un máximo de 65535 caracteres
MEDIUMTEXT	longitud + 3	un texto con un máximo de 16.777.215 caracteres
LONG TEXT	longitud + 4	un texto con un máximo de 4.294.967.295 caracteres

Valor	CHAR(4)	Bytes	VARCHAR(4)	Bytes
''	''	4	''	1
'ab'	'ab '	4	'ab'	3
'abcd'	'abcd'	4	'abcd'	5
'abcdefgh'	'abcd'	4	'abcd'	5

Tipo de datos: Numéricos Enteros

Tipo	Bytes	Con/sin signo	Valor Min.	Valor Max
TINYINT	1	con signo sin signo	-128 0	127 255
SMALLINT	2	con signo sin signo	-32768 0	32767 65535
MEDIUMINT	3	con signo sin signo	-8388608 0	8388607 16777215
INT / INTEGER	4	con signo sin signo	-2147483648 0	2147483647 4294967295
BIGINT	8	con signo sin signo	-9223372036854770000 0	9223372036854770000 18446744073709500000

se puede especificar la cantidad de dígitos enteros a mostrar. Ejemplo: INT(4). No restringe el tamaño a almacenar ni el tamaño si los datos exceden el valor indicado

se puede especificar ZEROFILL. Por ejemplo para INT(5) ZEROFILL, el valor 23 se mostrará 00023

se puede especificar UNSIGNED. Evita que se almacenen valores negativos. En este caso el rango de valores admitidos es el que se muestra en la tabla para "sin signo

Tipo de datos: Numéricos Decimales Flotantes y Exactos

Tipo	Bytes	CON/SIN SIGNO	Mínimo	Máximo
FLOAT (M,D)	4	con signo sin signo	-3.402823466E+38 0	1.175494351E-38 3.402823466E+38
DOUBLE(M,D) **DOUBLE PRECISION** **REAL**	8	con signo sin signo	-1.7976931348623157E+308 0	-2.2250738585072014E-308 1.7976931348623157E+308
DECIMAL(M, D) **DEC** **NUMERIC**	M+2 si D > 0 M+1 si D = 0	El número se almacena como una cadena de caracteres		

se puede especificar ZEROFILL. A diferencia de los numéricos enteros no varían los mínimos y máximos a almacenar

se puede especificar UNSIGNED. En este caso el rango de valores admitidos es el que se muestra en la tabla para "sin signo"

M indica precisión (cantidad de dígitos totales almacenados)

D indica escala (cantidad de dígitos después de la coma decimal)

Tipo de datos: Fecha

Tipo	Bytes	Observaciones
DATE	3	Almacena año-mes-día (AAAA-MM-DD)
DATETIME	8	Almacena año-mes-día-hora-minuto-segundo (AAAA-MM-DD HH:MM:SS)
TIMESTAMP	4	Indica como valor por defecto CURRENT_TIMESTAMP y almacena automáticamente la fecha y hora del momento del registro
TIME	3	Almacena hora:minuto:segundo (HH:MM:SS)
YEAR	1	almacena año (AAAA)

CREACIÓN, MODIFICACIÓN Y BORRADO DE DOMINIOS

Además de los dominios dados por el tipo de datos predefinidos, el ESTÁNDAR nos ofrece la posibilidad de trabajar con dominios definidos por el usuario.

CREACIÓN

Para crear un dominio es necesario utilizar la sentencia CREATE DOMAIN:

CREATE DOMAIN nombre dominio [**AS**] tipos_datos
[def_defecto] [restricciones_dominio];

Donde **restricciones_dominio** tiene el siguiente formato:
[**CONSTRAINT** nombre_restricción] **CHECK** (condiciones)

Ejemplo TSL:

Si quisiéramos definir un dominio para las ciudades donde se encuentran los clientes para el ejemplo TSL, partiendo que la empresa no tomará prendas a limpiar de clientes que no pertenezcan a las ciudades definidas:

```
CREATE DOMAIN dom_ciudades
AS VARCHAR (20)
CONSTRAINT ciudades_validas
CHECK (VALUE IN ('Rosario', 'Perez', 'Galvez', 'Funes'));
```

De este modo, cuando definimos la columna **ciudades** dentro de la tabla clientes no se tendrá que decir que es de tipo CHAR (50), sino de tipo **dom_ciudades.**

No obstante, la creación de un dominio si bien sirve para validación permite la comparación con otra columna definida del mismo tipo de datos.
Por ejemplo, si tenemos una columna con los nombres de los empleados definida sobre el tipo de datos VARCHAR (50), el SQL nos permite compararla con la columna ciudades, aunque semánticamente no tenga sentido.

BORRADO

Para borrar un dominio definido por el usuario DROP DOMAIN:

DROP DOMAIN nombre_dominio **{RESTRICT|CASCADE};**

Donde:

RESTRICT hace que el dominio sólo se pueda borrar si no se utiliza en ningún sitio.

CASCADE borra el dominio aunque esté referenciado, y pone el tipo de datos del dominio allí donde se utilizaba.

MODIFICACIÓN

Para modificar un dominio semántico es necesario utilizar la sentencia **ALTER DOMAIN.**

ALTER DOMAIN nombre_dominio {acción_modificar_dominio|
acción_modif_restricción_dominio};

donde acción_modif_restricción_dominio puede ser**:**
{ADD restricciones_dominio**|DROP CONSTRAINT** nombre_restricción}

Ejemplo TSL:

Si quisiéramos añadir una nueva ciudad (Villa Constitución) al dominio que hemos creado antes para las ciudades donde se encuentran los clientes:

ALTER DOMAIN dom_ciudades DROP CONSTRAINT ciudades_validas;

Con esto hemos eliminado la restricción de dominio antigua. Y ahora tenemos que introducir la nueva restricción:

ALTER_DOMAIN dom_ciudades ADD CONSTRAINT ciudades_validas
CHECK(VALUE IN ('Rosario', 'Perez', 'Galvez', 'Funes', 'Villa Constitución'));

DEFINICIONES POR DEFECTO

La opción por defecto nos permite especificar qué nomenclatura queremos dar a nuestros valores por omisión.

Hay que tener en cuenta que si elegimos la opción DEFAULT NULL, la columna para la que daremos la definición por defecto de valor nulo debería admitir valores nulos.

La opción **DEFAULT** tiene el siguiente formato:

DEFAULT (literal|función|NULL)

La opción más utilizada y por defecto, es la palabra reservada NULL. También podemos definir nuestro propio literal, o bien recurrir a una de las funciones que aparecen en la tabla siguiente:

Función	Descripción
{USER|CURRENT_USER}	Identificador del usuario actual
SESSION_USER	Identificador del usuario de esta sesión
SYSTEM_USER	Identificador del usuario del sistema
CURRENT_DATE	Fecha actual
CURRENT_TIME	Hora actual
CURRENT_TIMESTAMP	Fecha y hora actuales

RESTRICCIONES DE COLUMNA

Una vez les hemos dado un nombre a las tablas y se ha definido su dominio, podemos imponer ciertas restricciones que siempre se tendrán que cumplir. Las restricciones que se pueden dar son:

Restricciones de columna	
Restricción	Descripción
NOT NULL	La columna no puede tener valores nulos.
UNIQUE	La columna no puede tener valores repetidos. Es una clave
PRIMARY KEY	La columna no puede tener valores repetidos ni nulos. Es la clave primaria.
REFERENCES Tabla [(columna)]	La columna es la clave foránea de la columna de la tabla especificada.
CHECK (condiciones)	La columna debe cumplir las condiciones especificadas.

Ejemplo TSL:

```
CREATE TABLE 'empleados'
'cuil`varchar(20) NOT NULL PRIMARY KEY ,
'nom_ape' varchar(50) NOT NULL,
sexo char(1) CHECK VALUE IN ("F","M")
```

RESTRICCIONES DE TABLA

Una vez que definido una tabla y hemos impuesto ciertas restricciones para cada una de las columnas, podemos aplicar restricciones sobre toda la tabla, que siempre se deberán cumplir. Las restricciones que se pueden dar son las siguientes:

Restricciones de tabla	
Restric	Descripción
UNIQUE (columna [, columna. . .])	El conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa.
PRIMARY KEY (columna [, columna. . .])	El conjunto de las columnas especificadas no puede tener valores nulos ni repetidos. Es una clave primaria.
FOREIGN KEY (columna [, columna. . .]) REFERENCES tabla [(columna2 [, columna2. . .])]	El conjunto de las columnas especificadas es una clave foránea que referencia la clave primaria formada por el conjunto de las columnas2 de la tabla dada. Si las columnas y las columnas2 se denominan exactamente igual, entonces no sería necesario poner columnas2.
CHECK (condiciones)	La tabla debe cumplir las condiciones especificadas.

Ejemplo TSL.

```
CREATE TABLE `procesos_realizados` (
`nro_servicio` int(11) NOT NULL,
`orden` int(11) NOT NULL,
`cod_proceso` int(11) NOT NULL,
`fecha_inicio` date NOT NULL,
`hora_inicio` time NOT NULL,
`cuil_empleado` varchar(20) NOT NULL,
`fecha_fin` date default NULL,
`hora_fin` time default NULL,
 `resultado_proceso` varchar(20) default NULL,
PRIMARY KEY
(`nro_servicio`,`orden`,`fecha_inicio`,`hora_inicio`),
KEY `procesos_realizados_fk1` (`cod_proceso`),
KEY `procesos_realizados_fk2` (`cuil_empleado`),
CONSTRAINT `procesos_realizados_fk` FOREIGN KEY (`nro_servicio`,
`orden`) REFERENCES `tratamiento_limpieza` (`nro_servicio`, `orden`)
ON UPDATE CASCADE,
CONSTRAINT `procesos_realizados_fk1` FOREIGN KEY (`cod_proceso`)
REFERENCES `procesos`   (`cod_proceso`) ON UPDATE CASCADE,
CONSTRAINT `procesos_realizados_fk2` FOREIGN KEY (`cuil_empleado`)
REFERENCES `empleados` (`cuil`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Orden de creación
Antes de crear una tabla con una o más claves foráneas, se deben haber creado las tablas que tienen como clave primaria las referenciadas por las foráneas.

NOTA

Al crear una tabla vemos que muchas restricciones se pueden imponer de dos formas:

- como restricciones de columna
- como restricciones de tabla.

Cuando queremos decir cuál es la clave primaria de una tabla, tenemos las dos posibilidades:

- En el caso de que la restricción haga referencia a un solo atributo, podemos elegir la posibilidad que más nos guste.
- En el caso de que la restricción es compuesta por más de un atributo, tenemos que elegir por fuerza la opción de restricciones de tabla.

Observar en la creación de tablas que hay casos donde tenemos que elegir restricción de tabla porque la clave primaria está compuesta por más de un atributo. En general, lo pondremos todo como restricciones de tabla, excepto NOT NULL y CHECK cuando haga referencia a una sola columna.

MODIFICACIÓN Y BORRADO DE CLAVES PRIMARIAS CON CLAVES FORÁNEAS QUE HACEN REFERENCIA A ÉSTAS

En casos de borrado y modificación de filas que tienen una clave primaria referenciada por claves foráneas hay que tener en cuenta que existen políticas de restricción, actualización en cascada y la anulación.

```
CREATE TABLE nombre_tabla
    ( definición_columna
    [, definición_columna. . .]
    [, restricciones_tabla]);
```

Donde una de las restricciones de tabla es la definición de claves foráneas, que tiene el siguiente formato:

```
FOREIGN KEY clave_secundaria REFERENCES tabla [(clave_primaria)]
ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
```

Donde:

NO ACTION corresponde a la política de restricción;
CASCADE a la actualización en cascada;
SET NULL sería la anulación;
SET DEFAULT se podría considerar una variante de SET NULL, donde en lugar de valores nulos se puede poner el valor especificado por defecto.

ASERCIONES

Una aserción es una restricción general que hace referencia a una o más columnas de más de una tabla. Para definir una aserción se utiliza la sentencia CREATE ASSERTION, y tiene el siguiente formato:

CREATE ASSERTION nombre_aserción **CHECK (condiciones);**

Ejemplo TSL:

Creamos una aserción sobre la base de datos que nos asegure que no hay ningún empleado con un sueldo menor a 2.000 y que haya realizado el proceso 11 (Limplieza Térmica):

```
CREATE ASSERTION restriccion1 CHECK
(NOT EXISTS (SELECT * FROM empleados e, procesos_realizados p
WHERE e.cod_proceso =  e.cod_proceso and e.sueldo < 2000
And e.cod_proceso = 11 );
```

Para borrar una aserción es necesario utilizar la sentencia **DROP ASSERTION**:

DROP ASSERTION nombre_aserción;

Ejemplo TSL:

Para borrar la aserción restriccion1, utilizaríamos la sentencia:

```
DROP ASSERTION restriccion1;
```

Nota: MySQL 5.1 no lo incluye

MODIFICACIÓN Y BORRADO DE TABLAS

Para modificar una tabla es preciso utilizar la sentencia **ALTER TABLE**:

ALTER TABLE nombre_tabla {acción_modificar_columna|
acción_modif_restricción_tabla};

Donde:

acción_modificar_columna:

{ADD [COLUMN] columna def_columna **|**
ALTER [COLUMN] columna **{SET def_defecto|DROP DEFAULT}|**
DROP [COLUMN] columna **{RESTRICT|CASCADE}}**

acción_modif_restricción_tabla:

{ADD restricción**|**
DROP CONSTRAINT restricción
RESTRICT|CASCADE}}

Modificar una tabla puede implicar:

- Añadir una columna (ADD columna).
- Modificar las definiciones por defecto de la columna (ALTER columna).
- Borrar la columna (DROP columna).
- Añadir alguna nueva restricción de tabla (ADDCONSTRAINT restricción).
- Borrar alguna restricción de tabla (DROPCONSTRAINT restricción).

Para borrar una tabla es preciso utilizar la sentencia DROP TABLE:

DROP TABLE nombre_tabla **{RESTRICT|CASCADE};**

Donde:

RESTRICT, la tabla no se borrará si está referenciada, por ejemplo, por alguna vista.

CASCADE, todo lo que referencie a la tabla se borrará con ésta.

Ejemplo TSL:

Agregamos a la tabla EMPLEADOS los atributos ciudad, fecha_ingreso y salario

```
ALTER TABLE empleados
ADD COLUMN   ciudad varchar(30) default NULL,
ADD COLUMN   fecha_ingreso date default NULL,
ADD COLUMN   salario decimal (10,3) default NULL
```

Agregamos a la tabla CLIENTES el atributo sexo con la restricción CHECK

```
ALTER TABLE clients
ADD COLUMN sexo char(1) CHECK VALUE IN ("F","M")

ALTER TABLE valores_tratamientos
ADD CONSTRAINT `valores_tratamientos_fk2` FOREIGN KEY
(`cod_tratamiento`)
REFERENCES `tratamientos` (`cod_tratamiento`) ON UPDATE
CASCADE;

ALTER TABLE valores_tratamientos
DROP FOREIGN KEY valores_tratamientos_fk2;
```

CREACIÓN Y BORRADO DE VISTAS

Además de las Tablas, de existencia física, podemos crear vistas. Estas son una tabla virtual derivada de una consulta a tablas de existencia física en la Base de Datos.
Para crear una vista es necesario utilizar la sentencia **CREATE VIEW.**

```
CREATE VIEW nombre_vista [(lista_columnas)] AS (consulta)
[WITH CHECK OPTION];
```

Para crear una vista se define qué nombre le queremos poner (nombre_vista). Si queremos cambiar el nombre de las columnas, o bien poner nombre a alguna que en principio no tenía, lo podemos hacer en lista_columnas. Luego quedará por definir la consulta que formará nuestra vista.

Las vistas no existen realmente como un conjunto de valores almacenados en la base de datos, sino que son tablas ficticias, denominadas derivadas (no materializadas). Se construyen a partir de tablas reales (materializadas) almacenadas en la base de datos, y conocidas con el nombre de tablas básicas (o tablas de base). La no existencia real de las vistas hace que puedan ser actualizables o no. Una de las ventajas que podemos mencionar al usar vistas es que simplifican las consultas generando independencia de los datos.

Para crear una vista, se "embebe" una consulta dentro de la sentencia CREATE VIEW, esta consulta que se realiza con el comando SELECT puede ser tan compleja como necesitemos. Veremos en la próxima unidad consultas de este tipo. Por ahora realizaremos ejemplos simples para su comprensión.

Las vistas se pueden clasificar en:

- Simples: Poseen una sola tabla, no contiene funciones, no contiene grupos. Por estas características se pueden hacer operaciones de DML a través de la vista
- Compuestas: Poseen una o más tablas, contienen funciones y/o funciones de grupo. Por estas características NO se pueden hacer operaciones de DML a través de la vista

Ejemplo TSL:

Creamos una vista sobre la base de datos TSL que contenga para cada empleado y por cada proceso, la cantidad de veces que se ha realizado el proceso hasta el momento.

```
CREATE VIEW procesos_empleados (nombre_emp, desc_proceso,
cant_procesos) AS
(SELECT nom_ape, desc_proceso
FROM empleados e
INNER JOIN procesos_realizados pr
ON e.cuil = pr.cuil_empleado
INNER JOIN procesos p
ON pr.cod_proceso = p.cod_proceso);
```

Para borrar una vista es preciso utilizar la sentencia **DROP VIEW**:

```
DROP VIEW nombre_vista (RESTRICT|CASCADE);
```

Donde:

RESTRICT, la vista no se borrará si está referenciada,por ejemplo, por otra vista.

CASCADE, todo lo que referencie a la vista se borrará con ésta.

Ejemplo TSL:

Para borrar la vista procesos_empleados:

```
DROP VIEW procesos_empleados RESTRICT;
```

BORRADO DEL CONTENIDO DE UNA TABLA

El comando TRUNCATE borra todo el contenido de una tabla.
Si bien, en un principio, esta sentencia parecería ser DML (Lenguaje de Manipulación de Datos), es en realidad una DDL, ya que internamente, el comando TRUNCATE borra la tabla y la vuelve a crear y no ejecuta ninguna transacción.

El comando similar del DML es el comando DELETE que borra registros, pero, en casos que se quiera borrar todo el contenido de la tabla el comando TRUNCATE es más eficiente por su rapidez, especialmente si la tabla es muy grande y contiene muchos índices.

Ejemplo TSL:

```
TRUNCATE valores_tratamientos
```

ANEXO I: ENUNCIADO Y MODELO CONCEPTUAL: TINTORERÍA SIEMPRE LIMPIA

La empresa **Tintorería ropa siempre limpia** nos ha convocado para desarrollar un sistema para la atención y seguimiento de los **servicios de limpieza** que brindan a sus clientes.

La operatoria de la empresa es la siguiente:

Cuando un cliente (1) se presenta y trae alguna prenda , se le entrega un comprobante por el Servicio de Limpieza a realizar numerado de forma única y secuencial (cada prenda entregada se considera como un Servicio de Limpieza por separado) y se registra, fecha de entrega, prenda entregada, estado de la prenda (una descripción de la prenda y de su estado inicial para constatar en caso de reclamos), observaciones del cliente (sin almidonar, lugar donde se encuentran las manchas, etc.), observaciones internas hechas por el encargado para referencia de los empleados y la fecha estimada de entrega de la prenda.

En función de la solicitud del cliente, el encargado del mostrador define los tratamientos (2) que deben realizarse para las prendas entregadas. Para ello indica además en qué orden se realizan y alguna observación para el/los empleados que realicen el tratamiento. Para un mismo Servicio de Limpieza se pude realizar un mismo tratamiento varias veces.

Los tratamientos se componen de procesos (3).

Cuando un empleado (4) va a realizar algún proceso sobre la prenda, debe indicarlo en el sistema para que quede registrado el Servicio de Limpieza, el tratamiento y proceso a realizar, además el sistema registra la fecha y hora de inicio y al empleado como el responsable.

Una vez realizado el proceso, el empleado registra la finalización del mismo en el sistema. Luego el encargado revisa las prendas y determina el resultado de los procesos para el servicio (Satisfactorio o Insatisfactorio). Si el proceso resulta insatisfactorio deberá realizarse nuevamente, puede ser por el mismo empleado que antes o no.

---

(1) De los clientes se conoce tipo y número de documento, nombre y apellido, teléfono, dirección y ciudad.

(2) La empresa tiene ciertos tratamientos predefinidos como: tintura a color más oscuro, tintura a color más claro, lavado en seco, lavado común, limpieza de ropa delicada, etc. De cada uno de estos tratamientos se conoce su código y descripción y tienen un precio que varía en el tiempo. Hay dos clases de Tratamientos: sólo manuales y en máquina (5). Para aquellos que se realizan en máquina debe registrarse en que máquina se realiza. (Por simplificación se considera que un proceso sólo puede realizarse en una máquina).

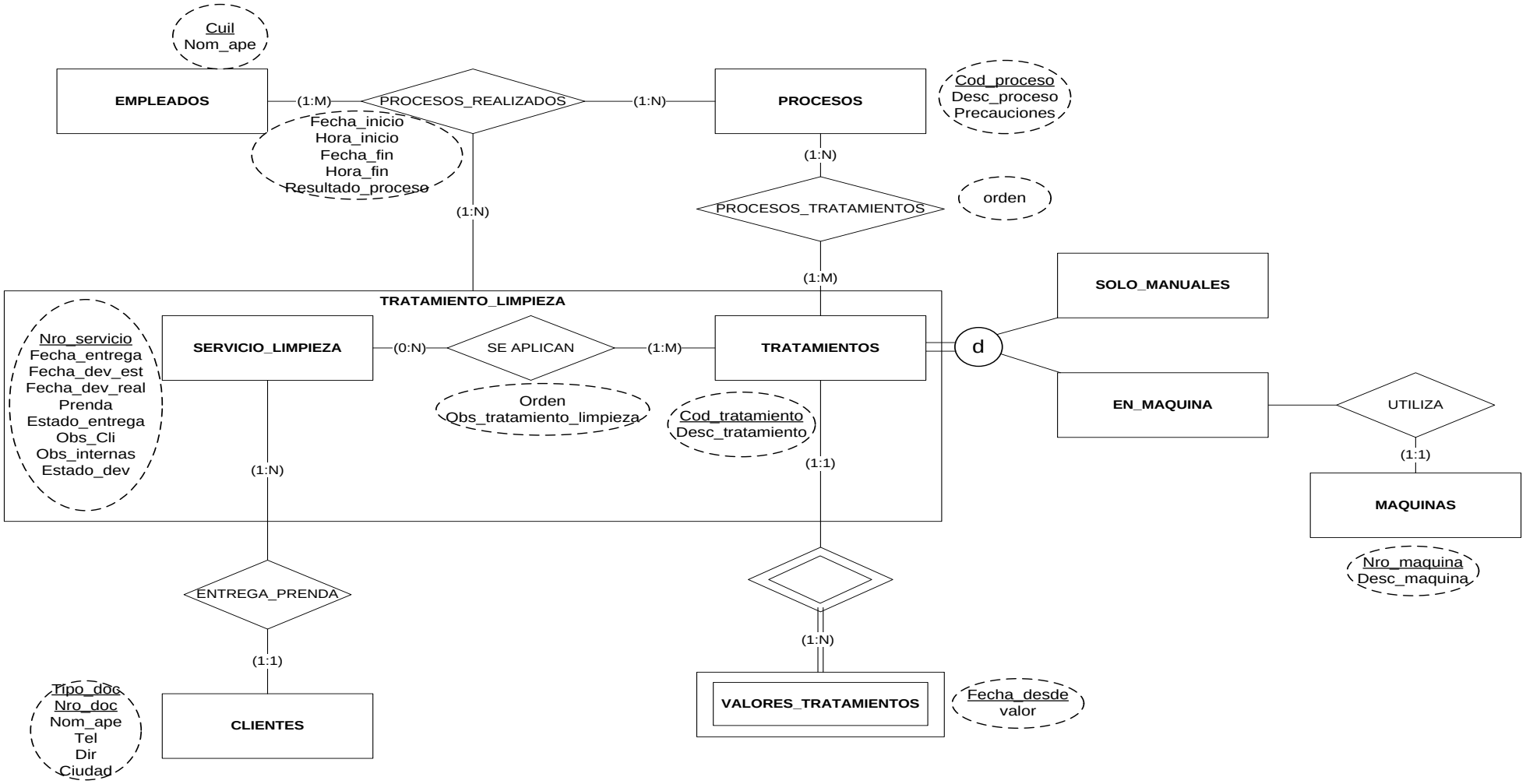
(3) De los procesos se registra su código, descripción y precauciones (un texto indicando las precauciones que el empleado que lo realice debe tener en cuenta para su seguridad). Los procesos pueden ser parte de distintos tratamientos y además debe registrase el orden en que suceden para cada tratamiento.

Ejemplo de Tratamientos y Procesos predefinidos:

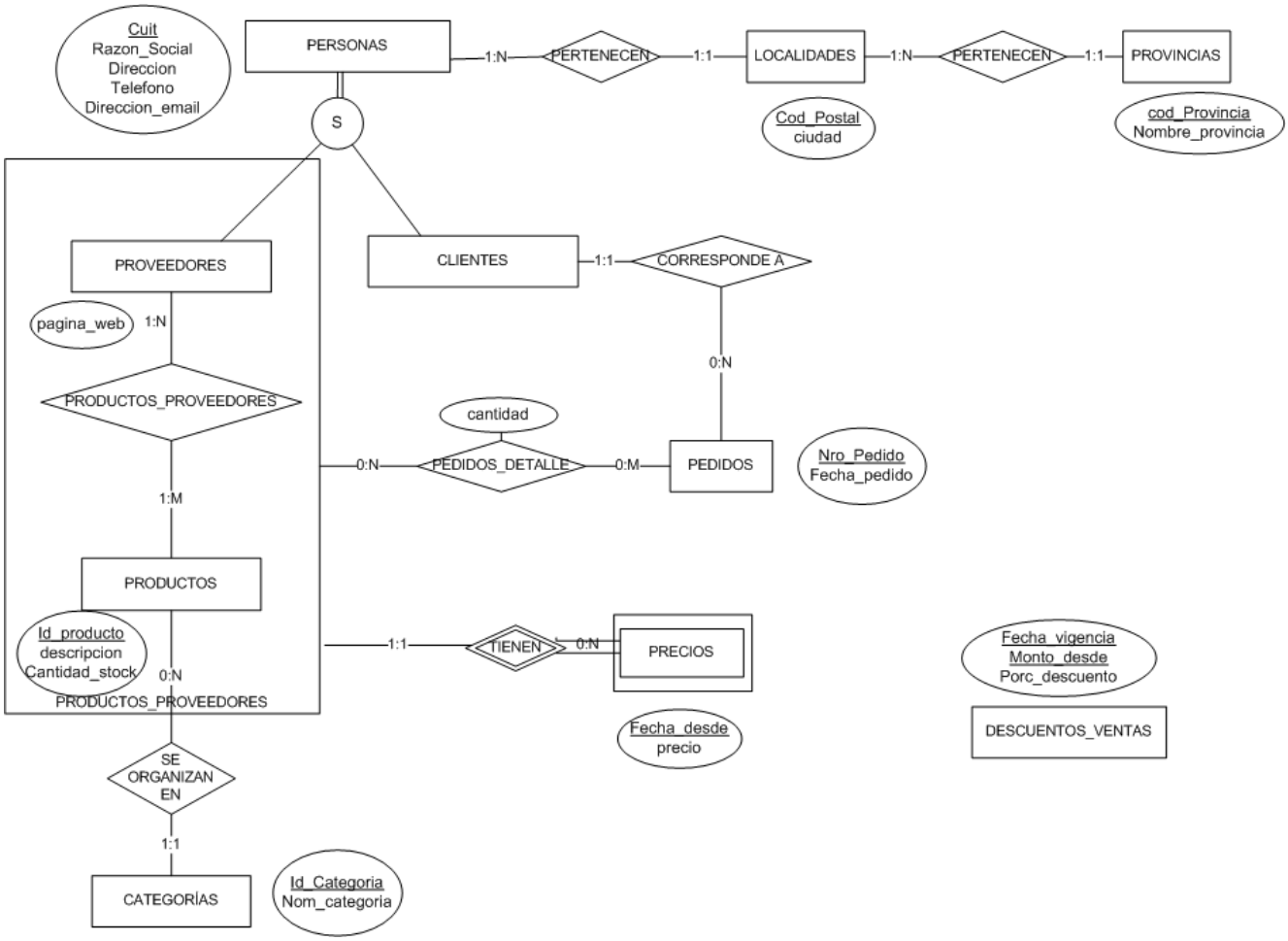
Tratamiento	Proceso	Orden
Tintura a color más claro	Lavado	1
	Decoloración	2
	Teñido	3
	Limpieza	4
Tintura a color más oscuro	Lavado	1
	Teñido	2
	Limpieza	3

(4) De los empleados se conoce cuil, nombre, apellido, ciudad, fecha de ingreso y salario.

(5) De las máquinas se conoce su número y su descripción.


Cuil
Nom_ape
EMPLEADOS
(1:M)
PROCESOS_REALIZADOS
(1:N)
PROCESOS
Cod_proceso
Desc_proceso
Precauciones
Fecha_inicio
Hora_inicio
Fecha_fin
Hora_fin
Resultado_proceso
(1:N)
(1:N)
PROCESOS_TRATAMIENTOS
orden
(1:M)
TRATAMIENTO_LIMPIEZA
SOLO_MANUALES
Nro_servicio
Fecha_entrega
Fecha_dev_est
Fecha_dev_real
Prenda
Estado_entrega
Obs_Cli
Obs_internas
Estado_dev
SERVICIO_LIMPIEZA
(0:N)
SE APLICAN
(1:M)
TRATAMIENTOS
d
EN_MAQUINA
UTILIZA
(1:1)
MAQUINAS
Nro_maquina
Desc_maquina
Orden
Qbs_tratamiento_limpieza
Cod_tratamiento
Desc_tratamiento
(1:N)
(1:1)
ENTREGA_PRENDA
(1:1)
(1:N)
Tipo_doc
Nro_doc
Nom_ape
Tel
Dir
Ciudad
CLIENTES
VALORES_TRATAMIENTOS
Fecha_desde
valor

ANEXO II: MODELOS CONCEPTUAL Y RELACIONAL: FERRETERÍA



MODELO RELACIONAL

PROVINCIAS (**cod_provincia**, nombre_provincia)
CP

CF(PROVINCIAS) NN
LOCALIDADES (**cod_postal**, ciudad, **cod_provincia**)
CP

PT de PERSONAS a SUPERCLASE

CF(LOCALIDADES) NN
PERSONAS (**Cuit**, razon_social, direccion, **cod_postal**, telefono, direccion_email, pagina_web)
CP

CATEGORIAS (**Id_categoria**, nom_categoria)
CP

CF(CATEGORIAS) NN
PRODUCTOS (**Id_producto**, descripcion, cantidad_stock, **Id_categoria**)
CP

CF(PRODUCTOS) CF(PERSONAS)
PRODUCTOS_PROVEEDORES (Id_producto , Cuit)
CP

CF(PRODUCTOS_PROVEEDORES)
PRECIOS (Id_producto , Cuit, fecha_desde, precio)
CP

CF(PERSONAS) NN
PEDIDOS (**Nro_pedido,** fecha_pedido, **Cuit**)
CP

CF(PEDIDOS) CF(PRODUCTOS_PROVEEDORES)
PEDIDOS_DETALLE(Nro_pedido, Id_producto , Cuit, cantidad)
CP

DESCUENTOS_VENTAS (**fecha_vigencia, monto_desde**, porc_descuento)
CP