

Práctica de laboratorio 2

Parte 2. Carga de datos

1. Crear la base de datos
 - Recordad lo visto en la sesión 1 de laboratorio:
 - `psql -U <usuario>` para conectarnos al SGBD.
 - `CREATE DATABASE <nombre>` para crear la base de datos que utilizaremos.
 - `\c <nombre DB>` para conectarse a la base de datos creada.
2. Crearemos el **esquema temporal** usando transacciones (ver plantilla_ejemplo.sql):
 - `BEGIN;` para comenzar el script
 - Crear el esquema temporal: `CREATE SCHEMA IF NOT EXISTS <nombre>`
 - Crear tablas según ficheros CSV proporcionados (sin integridad referencial)
 - Poblarlas de datos a partir de los ficheros CSV proporcionados: `COPY`.
3. Crearemos el **esquema final**:
 - Crear tablas según modelo referencial parte 1 (con integridad referencial)
 - Poblarlas haciendo consultas a las tablas del esquema temporal:
 - `INSERT INTO [esquema final] + SELECT [esquema temporal]`

Pasos a seguir.

1. Crear tablas temporales o intermedias(se puede hacer en cualquier esquema)

- Una tabla por fichero csv con los mismos atributos que los ficheros del dataset y sin restricciones activas (constrains). Por ejemplo:

```
CREATE TABLE
    esquema.[nombre_tabla] (
        atributo1 tipo,
        atributo2 tipo,
        .....
    );
```

2. Cargar los datos en las tablas temporales

- Copiar cada fichero csv en su tabla temporal sabiendo que: tiene formato csv, una cabecera, como delimitador usa ;, el carácter 'NULL' se toma como valor nulo y la codificación es UTF8.

```
\copy PL2temp.canciones_temp
from './datos/discos.csv'
WITH (FORMAT csv, HEADER, DELIMITER ';', NULL 'NULL', ENCODING 'UTF-8');
```

Pasos a seguir.

3. Crear tablas definitivas con las restricciones activas (constrains).

- Crear un esquema nuevo, por ejemplo, pl2final:

```
CREATE SCHEMA IF NOT EXISTS pl2final;
```

- Crear las tablas con las restricciones en dicho esquema:

```
\echo 'creando la tabla discos'
```

```
CREATE TABLE pl2final.discos (  
    a_pub integer NOT NULL,  
    t_disco text NOT NULL,  
    ...  
    CONSTRAINT discos_pk PRIMARY KEY (a_pub,t_disco)  
);
```

Pasos a seguir.

4. Cargar los datos en las tablas definitivas desde las tablas temporales

—Por ejemplo:

```
\echo 'cargando la tabla discos'  
INSERT INTO pl2final.discos(t_disco,a_pub,...)  
  SELECT DISTINCT ON(t_disco,a_pub) t_disco,a_pub::int,...  
FROM      PL2temp.discos_temp;
```

Creación de tablas

```
CREATE TABLE [IF NO EXISTS] esquema.nombre_tabla(  
    columna1 tipo_de_dato restricciones_de_columna,  
    columna2 tipo_de_dato restricciones_de_columna,  
    .....  
    columnan tipo_de_dato restricciones_de_columna  
    restricciones_de_tabla  
);
```

- A una tabla se le pueden aplicar restricciones a una columna determinada o a toda la tabla

Restricciones

○ Tipos de restricciones para forzar la integridad referencial:

- **NOT NULL**: La columna a la que se aplica la restricción no puede ser nula
- **UNIQUE**: El valor de la columna no se puede repetir, debe de ser único
- **PRIMARY KEY**: Establece un atributo como clave primaria
- **FOREIGN KEY**: Establece un atributo como clave foránea

○ Ejemplo:

```
CREATE TABLE IF NO EXISTS Persona(  
    DNI      CHAR[9] NOT NULL,  
    Nombre  TEXT UNIQUE,  
    Edad    INT  
);
```

- Se crea una tabla con los atributos DNI, Nombre y Edad:
 - DNI es un string de 9 caracteres de longitud y su valor no puede ser nulo
 - Nombre es de tipo Text y debe ser único, no se puede repetir el nombre de una persona. Su valor si puede ser nulo
 - Edad es un entero sin restricciones

Restricciones de clave primaria

- Define la columna o columnas que forman la clave primaria. Sólo puede existir una por tabla
- Su sintaxis es: `CONSTRAIN Nombre PRIMARY KEY(columna1, columna2,etc..)`

○ Ejemplo:

```
CREATE TABLE IF NO EXISTS Persona(  
  DNI      CHAR(9) NOT NULL,  
  Nombre  TEXT UNIQUE,  
  Edad    INT  
  CONSTRAINT Persona_pk PRIMARY KEY (DNI)  
);
```

- Se establece el DNI de una persona como la Primary Key de esa tabla

Restricciones de clave foránea

- Define la columna o columnas que forman una clave foránea o ajena (FK).
- Asegura que un valor de una columna o grupo de columnas existe en otra columna o grupos de columnas de otra tabla.
- Puede haber más de una clave foránea o FK.
- No se requiere igualdad de nombres, pero sí compatibilidad de dominios entre la clave ajena y la PK de la columna referenciada.

Restricciones de Clave foránea

Su sintaxis es:

CONSTRAIN nombre **FOREIGN KEY**(col1, col2, etc) –Establece qué columnas de la tabla forman la FK
REFERENCES tabla_referenciada (col1_referenciada, etc.) –Se indica la tabla a la que referencia la FK y las columnas referenciadas
MATCH FULL ON DELETE acción **ON UPDATE** acción

MATCH FULL: Cuando la clave foránea se define sobre varios atributos y pueden contener nulos

- Garantiza que no es posible una mezcla de valores nulos y no nulos:
 - Para cada tupla de la tabla referenciante, o todas las columnas de la FK son NULL, o ninguna lo es.
- Declarar las columnas referenciadas como NOT NULL para evitar que la restricción devuelva false.

Restricciones de Clave foránea

ON DELETE acción / **ON UPDATE** acción.

- Indican qué acción realizar cuando se borra/modifica una columna o columnas referenciadas por una FK.

- La acción puede ser:

- **RESTRICT**. No se permite el borrado. Se produce un error indicando que el borrado o modificación viola la restricción de la FK.
- **CASCADE**. Elimina/actualiza las filas que hagan referencia a la tupla eliminada/actualizada.
- **SET NULL**. La eliminación/actualización en la tabla referenciada implica poner en NULL la FK de todas las tuplas de la tabla referenciante cuya FK coincida con la PK de la tupla a borrar. Solo se puede especificar un subconjunto de columnas para las acciones ON DELETE.
- **SET DEFAULT**. El borrado en la tabla referenciada actualiza con el valor por defecto todas las columnas de referencia. Solo se puede especificar un subconjunto de columnas para las acciones ON DELETE.

Ejemplos

personas		
persona_id	persona_nom	ciudad_id
1	Pedro	1
2	Santiago	2
3	Juan	3
4	Andrés	1

RESTRICT

* Si intentamos borrar una fila en **ciudades** que tenga valores en **personas** tendremos el error **foreign key constraint fails**

* Si no hay registro relacionado se borrará

ciudades	
ciudad_id	ciudad_nom
1	Galilea
2	Betsaida
3	Patmos
4	Jerusalén

personas		
persona_id	persona_nom	ciudad_id
1	Pedro	1
2	Santiago	2
3	Juan	3
4	Andrés	1

CASCADE

ciudades	
ciudad_id	ciudad_nom
1	Galilea
2	Betsaida
3	Patmos
4	Jerusalén

* Si intentamos borrar una fila en **ciudades** que tenga valores en **personas** todos los registros relacionados en **personas** se borrarán

personas		
persona_id	persona_nom	ciudad_id
1	Pedro	NULL
2	Santiago	2
3	Juan	3
4	Andrés	NULL

SET NULL

ciudades	
ciudad_id	ciudad_nom
1	Galilea
2	Betsaida
3	Patmos
4	Jerusalén

* Si intentamos borrar una fila en **ciudades** que tenga valores en **personas** todos los registros relacionados en **personas** tendrán el valor **NULL** en la columna que es llave foránea

Ejemplo funcionamiento restri

Curso

<u>IdCarrera</u>	<u>NroCurso</u>	NbreCurso
APU	302	BDatos

Libro

<u>Titulo</u>	<u>ISBN</u>	<u>IdCarrera</u>	<u>NroCurso</u>
FDBS	1111	APU	302

```
CREATE TABLE Curso (  
    IdCarrera DominioCarrera NOT NULL,  
    NroCurso INT CHECK(NroCurso BETWEEN 100 AND 999)  
        DEFAULT 100,  
    NbreCurso CHAR(25),  
    PRIMARY KEY (IdCarrera, NroCurso)  
);  
CREATE TABLE Libro (  
    Titulo CHAR(30) NOT NULL,  
    ISBN INT PRIMARY KEY,  
    Id_Carrera DominioCarrera,  
    NroCurso INT CHECK(NroCurso > 99 AND NroCurso <  
        1000),  
    FOREIGN KEY (IdCarrera, NroCurso) REFERENCES Curso  
        MATCH <condicion> ON DELETE <accion referencial>
```

Supongamos:

Acción referencial: **SET NULL**
MATCH FULL

Veamos el comportamiento de algunas actualizaciones/borrados:

INSERT INTO Libro VALUES (Bases, 2222, APU, null);

Se rechaza por MATCH FULL

INSERT INTO Libro VALUES (Bases, 2222, null, null);

Se ejecuta satisfactoriamente por ~~MATCH FULL~~

DELETE FROM Curso WHERE IdCarrera = APU **AND** NroCurso = 302;

Se ejecuta satisfactoriamente por SET NULL (la tupla de Curso será borrada y la FK de la tupla de Libro se actualizará con NULL)

Ejemplo funcionamiento restri

Curso

<u>IdCarrera</u>	<u>NroCurso</u>	NbreCurso
APU	302	BDatos

Libro

<u>Titulo</u>	<u>ISBN</u>	<u>IdCarrera</u>	<u>NroCurso</u>
FDBS	1111	APU	302

```
CREATE TABLE Curso (  
  IdCarrera DominioCarrera NOT NULL,  
  NroCurso INT CHECK(NroCurso BETWEEN 100 AND 999)  
  DEFAULT 100,  
  NbreCurso CHAR(25),  
  PRIMARY KEY (IdCarrera, NroCurso)  
);  
CREATE TABLE Libro (  
  Titulo CHAR(30) NOT NULL,  
  ISBN INT PRIMARY KEY,  
  Id_Carrera DominioCarrera,  
  NroCurso INT CHECK(NroCurso > 99 AND NroCurso <  
  1000),  
  FOREIGN KEY (IdCarrera, NroCurso) REFERENCES Curso  
  MATCH <condicion> ON DELETE <accion referencial>
```

Supongamos:

Acción referencial: **CASCADE**
MATCH FULL

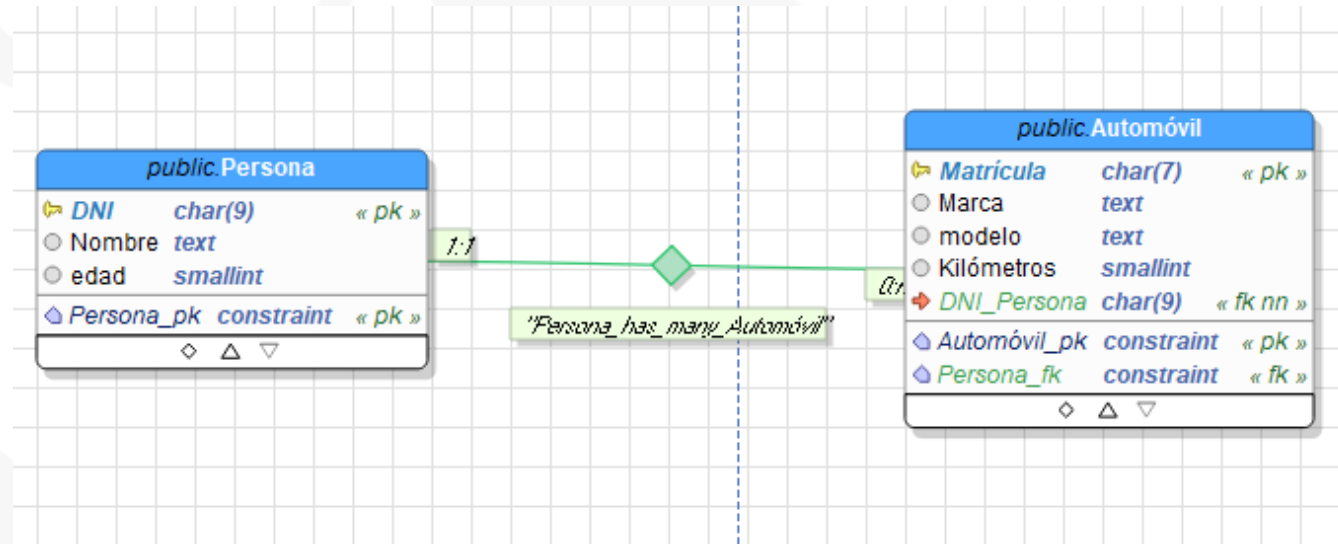
Veamos el comportamiento de algunas actualizaciones/borrados:

DELETE FROM Curso WHERE IdCarrera = APU AND NroCurso = 302;

Se ejecuta satisfactoriamente por CASCADE → se borrarán todas las tuplas (ambas relaciones)

Ejemplos

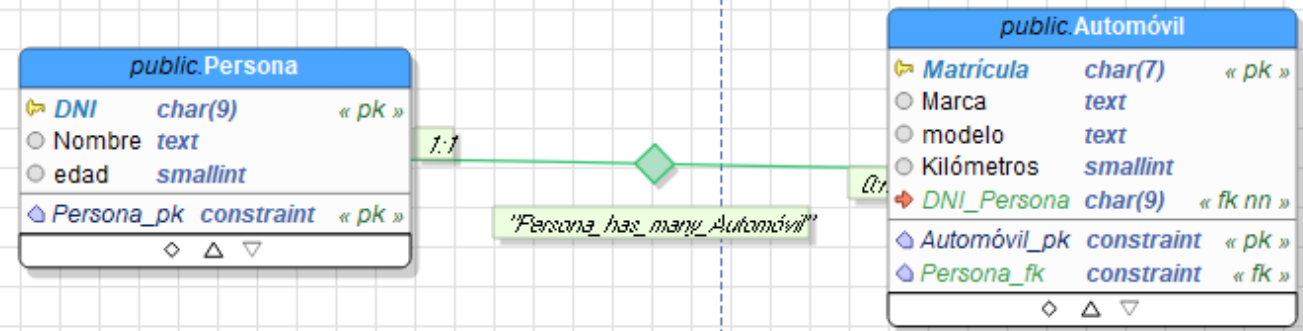
- Se quiere crear una tabla con los datos de los automóviles que pertenecen a las personas registradas en la tabla PERSONAS.
- Hay que tener en cuenta que una persona puede tener varios automóviles, pero un automóvil solo puede pertenecer a una sola persona
- El modelo relacional es el siguiente



Ejemplos

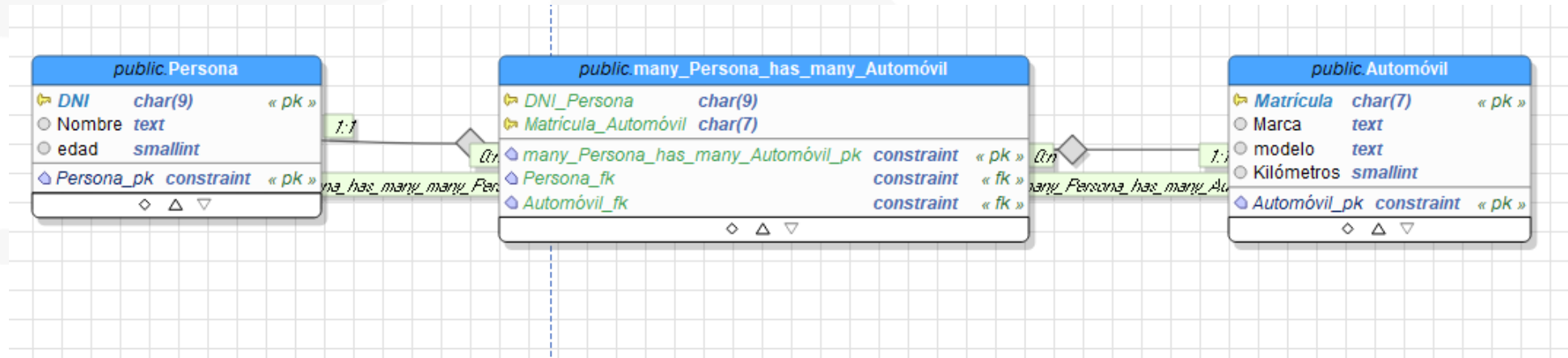
- El Código SQL del modelo relacional anterior es el siguiente:

```
CREATE TABLE IF NO EXISTS Persona(  
  DNI      CHAR(9) NOT NULL,  
  Nombre  TEXT UNIQUE,  
  Edad    INT  
  CONSTRAINT Persona_pk PRIMARY KEY (DNI)  
);  
  
CREATE TABLE IF NO EXISTS Automóvil (  
  Matrícula CHAR(7) NOT NULL,  
  Marca TEXT,  
  modelo TEXT,  
  Kilómetros INT,  
  DNI_Persona char(9) NOT NULL,  
  CONSTRAINT Automóvil_pk PRIMARY KEY (Matrícula),  
  CONSTRAINT Persona_fk FOREIGN KEY (DNI_Persona) REFERENCES Persona (DNI) MATCH FULL  
  ON DELETE RESTRICT ON UPDATE CASCADE  
);
```

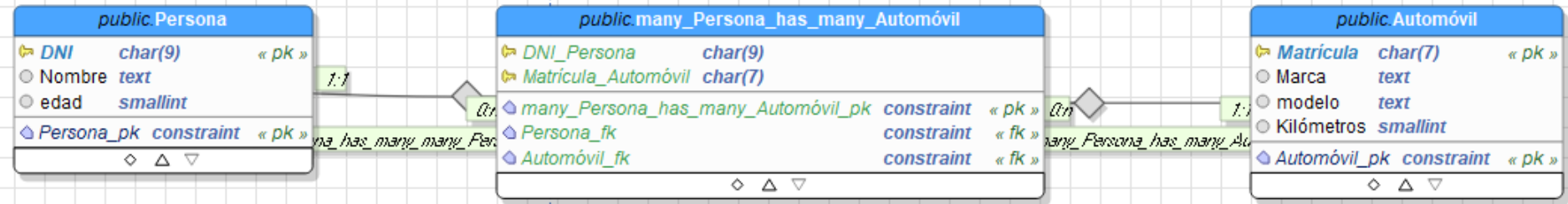


Ejemplos

- Ahora nos interesa registrar las personas que conducen los automóviles.
 - En este caso un automóvil puede ser conducido por más de una persona y una persona puede conducir varios automóviles.
- El modelo relacional sería el siguiente:



Ejemplos



oEn este caso
tenemos que crear
tres tablas con sus
respectivas
restricciones:

```
CREATE TABLE IF NO EXISTS Persona(  
    DNI CHAR(9) NOT NULL,  
    Nombre TEXT UNIQUE,  
    Edad INT  
    CONSTRAINT Persona_pk PRIMARY KEY (DNI)  
);  
  
CREATE TABLE IF NO EXISTS Automóvil (  
    Matrícula CHAR(7) NOT NULL,  
    Marca TEXT,  
    modelo TEXT,  
    Kilómetros INT,  
    CONSTRAINT Automóvil_pk PRIMARY KEY (Matrícula),  
);  
  
CREATE TABLE Conduce (  
    DNI_Persona char(9) NOT NULL,  
    Matrícula_Automóvil char(7) NOT NULL,  
    CONSTRAINT Conduce_pk PRIMARY KEY (DNI_Persona,Matrícula_Automóvil)  
    ADD CONSTRAINT Persona_fk FOREIGN KEY (DNI_Persona) REFERENCES Persona (DNI) MATCH FULL  
    ON DELETE CASCADE ON UPDATE CASCADE,  
    CONSTRAINT Automóvil_fk FOREIGN KEY (Matrícula_Automóvil) REFERENCES Automóvil (Matrícula) MATCH FULL  
    ON DELETE CASCADE ON UPDATE CASCADE;  
);
```

Tipos de datos en Postgres

Algunos de los tipos de datos más utilizados en Postgres son:

- Tipos de datos numéricos
- Tipos de datos carácter
- Tipos de datos lógicos
- Tipos de datos de fecha
- Tipos enumerados
- Etc.

Información detallada de todos los tipos de datos:

[PostgreSQL: Documentation: 16: Chapter 8. Data Types](#)

Tipos de datos en Postgres

Tipos de datos numéricos:

Name	Storage Size	Description	Range
smallint	2 bytes	small-range integer	-32768 to +32767
integer	4 bytes	typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807
decimal	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	4 bytes	variable-precision, inexact	6 decimal digits precision
double precision	8 bytes	variable-precision, inexact	15 decimal digits precision
smallserial	2 bytes	small autoincrementing integer	1 to 32767
serial	4 bytes	autoincrementing integer	1 to 2147483647
bigserial	8 bytes	large autoincrementing integer	1 to 9223372036854775807

Tipos de datos en Postgres

Tipos de datos carácter:

Name	Description
<code>character varying(<i>n</i>)</code> , <code>varchar(<i>n</i>)</code>	variable-length with limit
<code>character(<i>n</i>)</code> , <code>char(<i>n</i>)</code> , <code>bpchar(<i>n</i>)</code>	fixed-length, blank padded
<code>text</code>	variable unlimited length

Tipos de datos fecha y hora:

Name	Storage Size	Description	Low Value	High Value	Resolution
<code>timestamp [(<i>p</i>)] [without time zone]</code>	8 bytes	both date and time (no time zone)	4713 BC	294276 AD	1 microsecond
<code>timestamp [(<i>p</i>)] with time zone</code>	8 bytes	both date and time, with time zone	4713 BC	294276 AD	1 microsecond
<code>date</code>	4 bytes	date (no time of day)	4713 BC	5874897 AD	1 day
<code>time [(<i>p</i>)] [without time zone]</code>	8 bytes	time of day (no date)	00:00:00	24:00:00	1 microsecond
<code>time [(<i>p</i>)] with time zone</code>	12 bytes	time of day (no date), with time zone	00:00:00+1559	24:00:00-1559	1 microsecond
<code>interval [<i>fields</i>] [(<i>p</i>)]</code>	16 bytes	time interval	-178000000 years	178000000 years	1 microsecond

Tipos de datos en Postgres

Tipos de datos enumerados

- Se define el tipo enumerado con los posibles valores

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
```

- Se declara la variable enumerada en la creación de la tabla:

```
CREATE TABLE person (  
    name text,  
    current_mood mood
```

Conversión de tipos

Dentro de la instrucción SELECT de SQL se pueden realizar operaciones con los datos. Entre ellas está la conversión de un tipo a otro

```
SELECT
    año,título,puntuación, puntuación::numeric
FROM
    ddbb.películas;
```

	año text	título text	puntuación text	puntuación numeric
1	2022	Z-O-M-B-I-E-S 3	5.3	5.3
2	2022	X	6.6	6.6
3	2022	WOLFE	5.0	5.0

Estos cambios solo se aplican a la salida de la consulta, no al tipo de datos de la tabla.

Atributos multivaluados

Un atributo multievaluado se puede separar en diferentes tuplas con la instrucción:
(<https://www.postgresql.org/docs/16/functions-string.html>)

```
regexp_split_to_table(string text, pattern text [, flags text]) → setof text
```

Splits *string* using a POSIX regular expression as the delimiter, producing a set of results; see **Section 9.7.3**.

```
regexp_split_to_table('hello world', '\s+') →
```

```
hello
```

```
world
```


Atributos multivaluados

SELECT

```
regexp_split_to_table(géneros, '\s+'),  
año::int,  
título
```

FROM

```
dadb.películas
```

	regexp_split_to_table text	año integer	título text
1	Family	2022	Z-O-M-B-I-E-S 3
2	Musical	2022	Z-O-M-B-I-E-S 3
3	Romance	2022	Z-O-M-B-I-E-S 3
4	Action	2022	X
5	Horror	2022	X
6	Mystery	2022	X
7	Thriller	2022	X

Funciones de strings

Postgres posee varias funciones para aplicar a los tipos de datos string:

- Concatenación: `text1 || text2`
- Substring: Función `substring()`
- Recortar caracteres: Funciones `trim()`, `ltrim()`, `rtrim()`, `btrim()`
- Etc.






Para más información:

[PostgreSQL: Documentation: 16: 9.4. String Functions and Operators](#)

Funciones de strings

Ejemplo:



```
SELECT
    sector || planta || pasillo || puerta as número_de_despacho,
    sector, planta, pasillo, puerta
FROM
    ejemplo.despachos;
```

	número_de_despacho 	sector 	planta 	pasillo 	puerta 
	text	character varying	smallint	smallint	smallint
1	Norte345	Norte	3	4	5
2	Norte222	Norte	2	2	2
3	Norte318	Norte	3	1	8
4	Norte311	Norte	3	1	1
5	Norte224	Norte	2	2	4
6	Norte228	Norte	2	2	8

Funciones de strings

Ejemplo. Recortar los ceros a la izquierda de string que representa un código y convertirlo a entero:

```
SELECT
    codpon, ltrim(codpon, '0')::int
FROM
    ponencia;
```

	codpon [PK] character 	ltrim integer 
1	00010	10
2	00020	20
3	00030	30
4	00040	40
5	00050	50
6	00060	60

Funciones de strings

Ejemplo. Recortar un conjunto de caracteres de la parte final de un string:

```
select
    duración, trim(duración, 'mins')::int as duración_integer
from
    ddbb.películas
```

	duración text 	duración_integer integer 
1	88 mins	88
2	105 mins	105
3	150 mins	150
4	86 mins	86
5	92 mins	92
6	105 mins	105

Funciones de strings

Ejemplo. Extraer los substrings que contengan la palabra 'Drama':

```
select
    géneros, substring(géneros, 'Drama')
from
    ddbb.películas;
```

	géneros text		substring text 
1	Family Musical Romance		[null]
2	Action Horror Mystery Thriller		[null]
3	Action Drama		Drama
4	Action Documentary		[null]
5	Action Crime Drama Thriller		Drama

Funciones de strings

Ejemplo. Extraer los substrings que contengan dígitos y convertir el resultado a un entero:

```
select
    duración, substring(duración, '\d+')::int
from
    ddbb.películas
```

	duración text	substring integer
1	88 mins	88
2	105 mins	105
3	150 mins	150
4	86 mins	86
5	92 mins	92

Consideraciones a tener en cuenta para cargar esquema final

- Hay que tener en cuenta, a la hora de realizar las consultas, que podría haber valores duplicados que pueden dar error de duplicidad de la PK.
- Para cargar los géneros hay que usar `regexp_split_to_table`, pero antes hay que quitar los corchetes y las comillas (ej.: `replace` o `regexp_replace`).
- <https://www.postgresql.org/docs/16/functions-string.html>
- En los CVS se incluyen los id de discos y de grupos que pueden utilizarse para hacer las consultas en la inserción de datos en el esquema final (pero no incluirlos en las tablas finales).
- La duración de la canción está en formato Min:Sec y la necesitamos en Horas:Min:Sec.
- <https://www.postgresql.org/docs/16/functions-datetime.html>