INGENIERÍA | DEPARTAMENTO
UNIVERSIDAD DE LOS ANDES | DE COMPUTACIÓN
MÉRIDA VENEZUELA | ESCUELA DE SISTEMAS

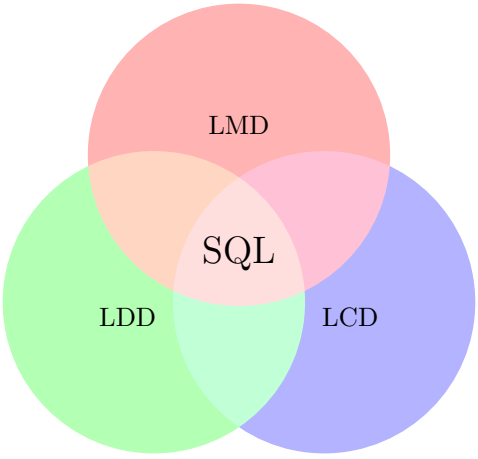
Laboratorio de Base de Datos
Práctica Nro. 7, SQL

Prof. Solazver Solé
Preps. Victor Albornoz, Yenifer Ramirez
Semestre B-2018

1. SQL

SQL por sus siglas en inglés (**Structured Query Languaje**) [1] es un lenguaje de dominio específico, usado en la programación y diseñado para administrar los datos persistentes en los sistemas de gestión de bases de datos relacionales (**SGBDR**).

Originalmente se basó en el **álgebra relacional** y en **cálculo relacional de tuplas**, SQL es tanto un lenguaje de manipulación de datos (**LMD**), como un lenguaje de definición de datos (**LDD**) y un lenguaje de control de datos (**LCD**).



[1] Lenguaje estructurado de consultas

1.1. Lenguaje de definición de datos: LDD

Una vez finalizado el diseño de una base de datos y escogido un **SGBD**[2] para su implementación, el primer paso consiste en especificar el esquema conceptual y los esquemas externo e interno de la base de datos.

El **SGBD** posee un compilador del **LDD** cuya función consiste en procesar las sentencias del lenguaje para identificar las descripciones de los distintos elementos del esquema y almacenar la descripción del esquema en el catálogo o diccionario de datos.

1.1.1. Expresiones Regulares POSIX para la creación de Dominios

Las **expresiones regulares**, son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto.

Ejemplo de una expresión regular:

(@[A-Za-z]{3,5} | #[A-Za-z]{3,5})

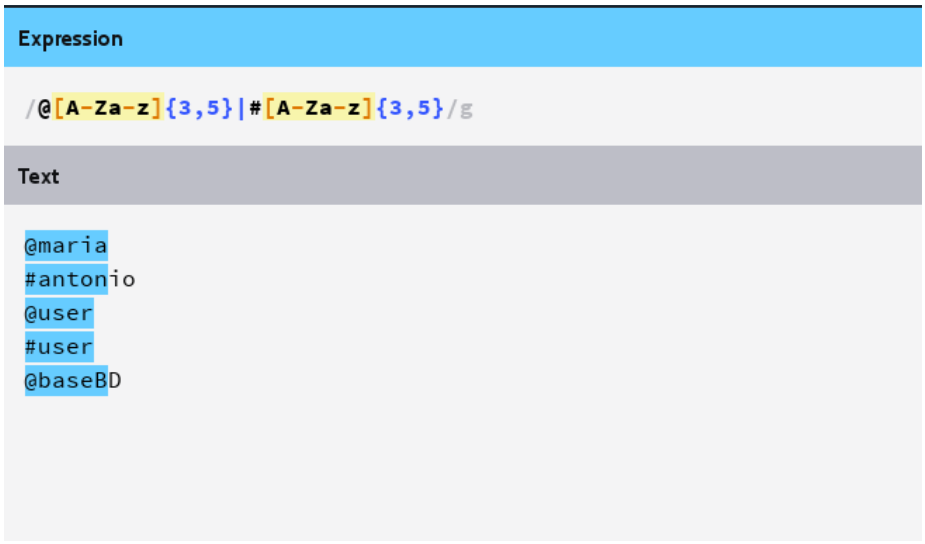


Figura 1: Programa para estudiar expresiones regulares RegExp

Preguntas:

1. ¿Para qué cree usted que seria útil encontrar patrones en las palabras?
2. ¿Cómo puede beneficiar las expresiones regulares el esquema de una base de datos?.

[2]SGBD: Sistema de Gestión de Bases de Datos

A continuación se muestran algunos metacaracteres básicos utilizados en **POSIX** [3] los cuales son compatibles con **Postgres**.

Metacaracter	Descripcion
.	Coincide cualquier caracter. Ejemplo '**a.b**' coincide con "abc "
[ ]	Expresión de corchete cerrado. Coincide con cualquier caracter contenido dentro de los corchetes. Ejemplo: '[**abc**]' coincide con "a ", "b "ó "c "
[^]	Coincide con cualquier caracter que no este contenido dentro de los corchetes cerrados. Ejemplo '[^**abc**]' coincide con cualquier palabra que no sea "a ", "b "ó "c "
^	Coincide la posición inicial de una cadena, si este es el primer caracter de la expresion regular. Ejemplo la expresión '^**N**' coincide con todas las palabras que empiecen en N: Naas , Neas, N*, N1, N2...
$	Coincide la posición final de una cadena, si este es el carácter final de una cadena. Ejemplo: la expresión ' **^.N$** ' coincide con las palabras que empiecen en cualquier caracter y terminen en N : aN, eN, *N, 1N, 2N ...
*	Coincide con cualquier caracter cero o mas veces. Ejemplo: la expresión '**N***' coincide con todas las palabras que tienen cero o mas caracteres 'N': aNa, aNNa, bNNNNb, acNNNNNac ...
+	Coincide con cualquier caracter 1 o mas veces. Ejemplo: la expresión ' **Ne+** ' coincide con todas las palabras que contienen una N le sigue una 'e' por lo menos una vez: aNea, aNNeea, bNeNeNeNeb, NeeNeeeNeeNeeeNeeec ...
{ m }	Acepta el elemento anterior exactamente m veces. Ejemplo: La expresión' **a{ 3}** ', solo acepta aaa.
{ m, }	Acepta el elemento anterior como mínimo m veces. Ejemplo: La expresión '**a{ 3,}**', acepta aaa, aaaa, aaaa, aaaaa, aaaaaa...
{ m,n }	Acepta el elemento anterior como mínimo m veces y como máximo n veces . Ejemplo: La expresión '**a{ 3,5}**', solo acepta aaa, aaaa, aaaaa.
( )	Define un sub-expresión, que es tratada como un elemento. Ejemplo: la expresión '**(ab)***' coincide con ' ',ab, ababa, ababab ...
[**0-9**]	Coincide con los dígitos del 0 al 9.
[**A-Za-z**]	Coincide con las letras del abecedario mayúsculas y minúsculas.

[3] POSIX:Portable Operating System Interface; la X viene de UNIX como seña de identidad de la API.

1.1.2. Dominios en Postgresql

Los **dominios** son útiles para abstraer restricciones comunes sobre los datos de las tablas de una base de datos. Por ejemplo, muchas tablas deben contener direcciones de correo, todas requieren la misma restricción de validación para **verificar la sintaxis**, por lo cual se hace útil definir un dominio en lugar de una restricción por cada tabla de manera individual.

```
1
2   CREATE DOMAIN name [ AS ] data_type
3   [ DEFAULT expression ]
4   [ constraint [ ... ] ]
5
6   where constraint is:
7
8   [ CONSTRAINT constraint_name ]
9   { NOT NULL | NULL | CHECK (expression) }
```

Sintaxis para la creación de un dominio en Postgresql

Los tipos de dominios mas comunes, validan cierta combinación de caracteres sobre una cadena. Por ejemplo, el siguiente dominio valida un **RIF**[4]:

```
1
2   --Puede empezar con V o J o G seguido de nueve digitos
3   --exactos y luego termina
4   CREATE DOMAIN RIF VARCHAR(10)
5   CONSTRAINT valid_rif
6   CHECK( VALUE ~'^(V|J|G)[0-9]{9}$');--tested
```

Dominio para validar el RIF.

Otros ejemplos:

```
1
2   --Empieza con @ minimo 3 letras guion o underscore seguido
        de 0 hasta 10 numeros
3   CREATE DOMAIN ID VARCHAR(21)
4   CONSTRAINT valid_user_id
5   CHECK (VALUE ~'^@[A-Za-z]{3,10}(-|_)?[0-9]{0,10}$');--
        tested
6
7   --Empieza con V o E seguida de - y luego de 1 a 8 digitos
8   CREATE DOMAIN CI VARCHAR(10)
9   CONSTRAINT valid_identifier
10  CHECK (VALUE ~'^(V|E)-[0-9]{1,8}$');--tested
```

Ejemplos de dominios

[4]Régimen de Incorporación Fiscal (RIF)

Muchas veces es necesario validar que un campo pertenezca a un conjunto previamente definido. Por ejemplo una clasificación o un color.

```
-- Verifica que la palabra sea M o F
CREATE DOMAIN GENERO VARCHAR(1)
CONSTRAINT set_of_genre
CHECK (VALUE IN ('M','F'));
```

Dominio para validar género.

Otros ejemplos:

```

--Valida la categoria de un restaurant
CREATE DOMAIN CATEGORIA_RESTAURANT VARCHAR(8)
CONSTRAINT set_of_categories
CHECK (VALUE IN ('LUJO','PRIMERA CLASE','SEGUNDA CLASE','
    TERCERA CLASE','CUARTA CLASE'));
```

Dominio para validar categoría de un restaurante.

1.2. Definición de tablas Postgres

Ejemplo de creación de tablas:

```

CREATE TABLE NOMBRETABLA(
  CAMPO1       TIPO_CAMPO[NOT NULL|DEFAULT|UNIQUE],
  CAMPO2       TIPO_CAMPO[NOT NULL|DEFAULT|UNIQUE],
  CAMPO3       TIPO_CAMPO[NOT NULL|DEFAULT|UNIQUE],
  CAMPO4       TIPO_CAMPO[NOT NULL|DEFAULT|UNIQUE],
  CAMPO5       TIPO_CAMPO[NOT NULL|DEFAULT|UNIQUE],
  CAMPO6       TIPO_CAMPO[NOT NULL|DEFAULT|UNIQUE],

  CONSTRAINT PK_NOMBRE_RESTRICCION PRIMARY KEY (CI),

  CHECK(CAMPO1 >  CAMPO2),

  CONSTRAINT FK_NOMBRE_RESTRICCION  FOREIGN KEY
  (MI_CAMPO) REFERENCES OTRA_TABLA (CAMPO_OTRA_TABLA)
   on delete [cascade|no action|set null|set default]
   on update [cascade|no action|set null|set default]
);

ALTER TABLE NOMBRE_TABLA
ADD CONSTRAINT FK_NOMBRE_RESTRICCION FOREIGN KEY
(CAMPO_TABLA) REFERENCES OTRA_TABLA (CAMPO_OTRA_TABLA)
on delete [cascade|no action|set null|set default]
on update [cascade|no action|set null|set default];
```

Sintaxis para la creación de tablas usando SQL en Postgresql

1.2.1. Descripción de los elementos involucrados en la creación de esquemas

- La palabra reservada **CREATE TABLE** crea una nueva tabla, inicialmente vacía en la base de datos actual.
- La palabra reservada **ALTER TABLE** modifica una tabla existente, alterando su estructura. Generalmente es usado para agregar restricciones a las tablas.
- La restricción **NOT NULL**, se usa para indicar que un campo específico de la tabla no puede ser nulo. Es decir al momento de insertar una nueva fila, este campo debe estar definido.
- La restricción **DEFAULT**, se usa para indicar que un campo específico de la tabla tendrá un valor por omisión. Es decir, si al momento de insertar una nueva fila este campo no se define, el SGBD le coloca el valor por omisión.
- La restricción **UNIQUE**, se usa para indicar que todos los elementos de la columna correspondiente a un campo especifico de la tabla tendrán valores únicos. Es decir al momento de insertar una nueva fila, este campo no debe repetirse.
- La restricción **PRIMARY KEY**, se usa para indicar la clave primaria de una tabla, es decir, aquel campo que no puede ser nulo y ademas tiene un valor único. Así al momento de insertar una nueva fila, el campo con clave primaria determinará de manera única la fila en la cual se encuentra.
- La restricción **FOREIGN KEY**, se usa para indicar que un campo específico de la tabla hace referencia a un campo de otra tabla o un campo de si misma. Es decir al momento de insertar una nueva fila si queremos colocar un valor en este campo que es clave foránea el SGBD chequea que este valor exista en el campo de la tabla a la que hace referencia. Por lo general se hace referencia a a la clave primaria de otra tabla, y a la tabla a la cual hace referencia, se le conoce como tabla padre (**parent table**).
- La restricción **CHECK**, se usa para indicar que un campo específico de la tabla debe satisfacer una condición que se expresa como una expresión booleana.

2. Diseño Físico

El diseño físico es el proceso de producir la descripción de la implementación de la base de datos en memoria secundaria, a partir del esquema lógico. Describe las relaciones base, organización de archivos, **índices** para lograr un eficiente acceso a los datos, **restricciones de integridad** asociadas y **medidas de seguridad**.

2.1. Pasos de el diseño físico de la base de datos.

1. Traducir el modelo lógico para el SGBD.
   - Diseñar las relaciones base.
   - Diseñar representación de los tipos de datos derivados.
   - Diseñar restricciones generales.
2. **Diseño de organización de archivos e índices**
   - Analizar transacciones
   - Elegir la organización de archivos.
   - Elegir índices.
   - Estimar espacio de disco.
3. **Diseño de vistas.**
4. **Diseño de mecanismos de seguridad**
5. Mecanismos de control de concurrencia.
6. Monitoreo y ajuste del sistema.

3. Indices

Los índices son estructuras adicionales que se utilizan para **acelerar el acceso a las tablas** en respuesta a ciertas condiciones de búsqueda.

- Hay que tener en cuenta que los índices conllevan un **coste de mantenimiento y espacio** que es necesario evaluar frente a la ganancia en prestaciones.
- Cada SGBD proporcionará uno o varios tipos de índices entre los que esco- ger. Los más habituales son los índices basados en **árboles B+ (o árboles B*)** y los basados en la **dispersión (hash)**.

3.1. Pautas para la creación de índices

- Crear un índice sobre las claves ajenas que se utilicen con frecuencia en operaciones de JOIN.
- Crear un índice sobre los atributos que se utilizan con frecuencia para hacer restricciones WHERE (son condiciones de búsqueda).
- Crear un índice único sobre las claves alternativas que se utilizan para hacer búsquedas.
- Evitar los índices sobre atributos que se modifican a menudo.
- Evitar los índices sobre atributos poco selectivos: aquellos en los que la consulta selecciona una porción significativa de la tabla (más del 15 % de las filas).
- Evitar los índices sobre atributos formados por cadenas de caracteres largas.
- Evitar los índices sobre tablas que se actualizan mucho y que se consultan muy esporádicamente (tablas de auditoría o diarios).
- Revisar si hay índices redundantes o que se solapan y eliminar los que no sean necesarios.

3.2. Índices en Postgres.

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ name ] ON table
( { column | ( expression ) }  [ ASC | DESC ] [ NULLS {
    FIRST | LAST } ] [, ...] )
[ WHERE predicate ]
```

3.3. Definición de índices sobre la base de datos chinook

```
--Indice sobre la clave foranea de la tabla Album "ArstistI"

CREATE INDEX IFK_AlbumArtistId ON
Album (ArtistId);
```

```
-- Indice sobre la clave foranea de la tabla Customer "
    SupportRepId"

CREATE INDEX IFK_CustomerSupportRepId ON
Customer(SupportRepId);
```

```sql
-- Indice sobre la clave foranea de la tabla Employee "ReportsTo"

CREATE INDEX IFK_EmployeeReportsTo ON
Employee (ReportsTo);
```

```sql
  -- Indice sobre la clave foranea de la tabla Employee "ReportsTo"

  CREATE INDEX IFK_EmployeeReportsTo ON
  Employee (ReportsTo);
```

```sql
-- Indice sobre la clave foranea de la tabla Invoice "CustomerId"

CREATE INDEX IFK_InvoiceCustomerId ON
Invoice (CustomerId);
```

```sql
-- Indice sobre la clave foranea de la tabla InvoiceLine "InvoiceId"

CREATE INDEX IFK_InvoiceLineInvoiceId ON
InvoiceLine (InvoiceId);
```

```sql
-- Indice sobre la clave foranea de la tabla PlaylistTrack "trackId"

CREATE INDEX IFK_PlaylistTrackTrackId ON
PlaylistTrack (TrackId);
```

```sql
-- Indice sobre la clave foranea de la tabla Track "MediaTypeId"

CREATE INDEX IFK_TrackMediaTypeId ON Track (MediaTypeId);
```

```sql
--Indice sobre la clave foranea de la tabla Track "AlbumId"

  CREATE INDEX IFK_TrackAlbumId ON  Track  (AlbumId);
```

```sql
--Indice sobre la clave foranea de la tabla Track "GenreId"

  CREATE INDEX IFK_TrackGenreId ON  Track (GenreId);
```

4. Vistas

Hay tres características importantes inherentes a los sistemas de bases de datos: la separación entre los **programas de aplicación y los datos**, el manejo de **múltiples vistas** por parte de los usuarios (esquemas externos) y el uso de un c**atálogo o diccionario** para almacenar el esquema de la base de datos.

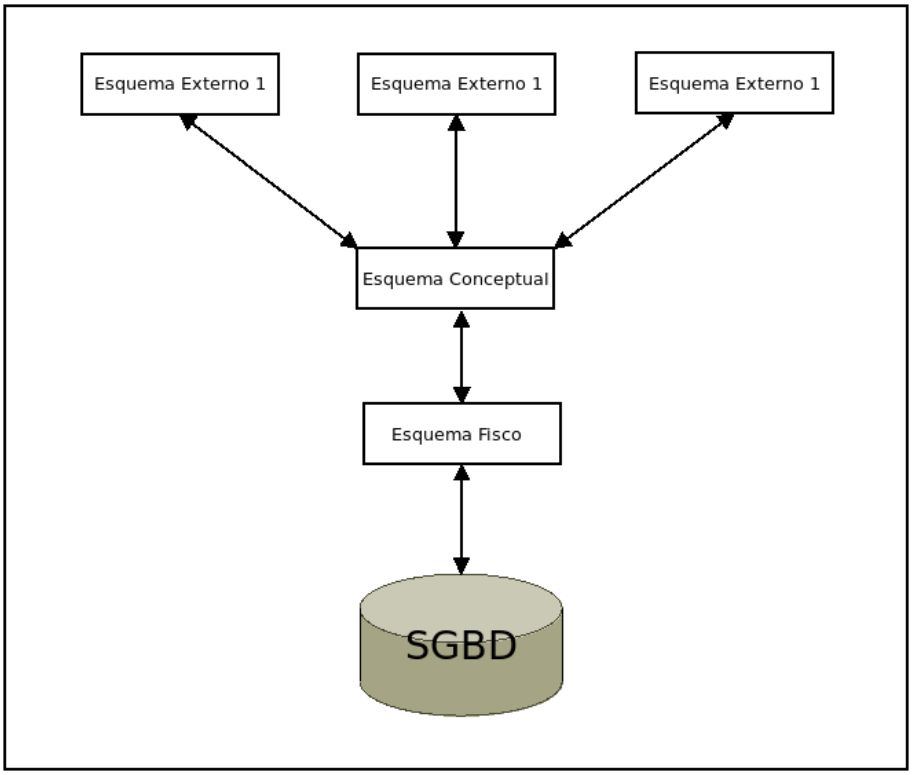


Figura 2: Arquitectura ANSI-SPARC para los SGBD

- Una vista se puede definir como una tabla virtual, una tabla que en realidad no existe como tal.
- Una vista es el resultado dinámico de una o varias operaciones relacionales realizadas sobre las tablas.
- Al usuario le parece que la vista es una tabla que existe y la puede manipular como si se tratara de una tabla, pero la vista no está almacenada físicamente.
- El contenido de una vista está definido como una consulta sobre una o varias tablas.

4.1. Utilidad y restricciones

Las vistas son útiles por varias razones.

- Proporcionan un poderoso mecanismo de seguridad, ocultando partes de la base de datos a ciertos usuarios.
- Permiten que los usuarios accedan a los datos en el formato que ellos desean o necesitan, de modo que los mismos datos pueden ser vistos con formatos distintos por distintos usuarios.
- Se pueden simplificar operaciones sobre las tablas que son complejas.
- El usuario puede hacer restricciones y proyecciones sobre la vista, que el SGBD traducirá en las operaciones equivalentes sobre el JOIN.
- Las vistas proporcionan independencia de datos a nivel lógico, que también se da cuando se reorganiza el nivel conceptual. Si se añade un atributo a una tabla, los usuarios no se percatan de su existencia si sus vistas no lo incluyen. Si una tabla existente se reorganiza o se divide en varias tablas, se pueden crear vistas para que los usuarios la sigan viendo como al principio.
- Cuando se actualiza una tabla, el cambio se refleja automáticamente en todas las vistas que la referencian. Del mismo modo, si se actualiza una vista, las tablas de las que se deriva deberían reflejar el cambio. Sin embargo, hay algu- nas restricciones respecto a los tipos de modificaciones que se pueden realizar sobre las vistas.
- Las vistas simples son automáticamente actualizables: el sistema permitira sentencias INSERT, UPDATE y DELETE para ser usadas en las vistas de la misma forma que una tabla. Una vista es actualizable bajo las siguientes condiciones:
    - La vista debe contener exactamente 1 entrada dentro de su lista FROM, la cual puede ser una tabla u otra vista.
    - La definición de la vista no debe contener clausulas: WITH, DISTINCT, GROUP BY, HAVING, LIMIT, o OFFSET.
    - La definición de la vista no debe contenedor operaciones sobre conjuntos (UNION, INTERSECT o EXCEPT).
    - Todas las columnas en la lista de SELECT deben ser simples referencias a columnas en las tablas. No pueden ser expresiones, funciones, literales.
    - Las columnas de las tablas no puede aparecer mas de una vez en la lista SELECT de la vista.
    - La vista no puede contener la propiedad security_barrier.

5. Vistas en Postgres

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name
[ ( column_name [, ...] ) ] AS query
```

6. Ejemplos de vistas base de datos Chinook

- Vista que contiene todos los empleados que hacen soporte a un cliente:

```
CREATE VIEW trabajadores_clientes AS
SELECT  employee.employeeid,
        employee.firstname as nombre_empleado,
        employee.lastname as apellido_empleado,
        employee.title,
        customer.customerid,
        customer.firstname as nombre_cliente,
        customer.lastname as apellido_cliente,
        customer.email
FROM    employee JOIN customer
ON      employee.employeeid = customer.supportrepid;
```

- Vista de la tabla TRACK con los valores en otro formato:

```
CREATE VIEW TRACK_FORMATTED AS
SELECT  track.name,
        track.composer,
        (track.milliseconds/1000)/60 as minutos,
        track.bytes/(1024*1024) as megabytes,
        track.unitprice * 36000 as Bolivares
FROM
        track
```