Introducción a las bases de datos y SQL

DESARROLLO DE SISTEMAS

Campos Ortega Carlos Alberto
Cuéllar Martínez Hugo Germán

¿Qué es SQL y para qué sirve?

El Lenguaje de Consulta Estructurado o SQL(Structured Query Language) por sus siglas en inglés, es la herramienta que sirve para manipular y emplear la información al interior de las bases de datos, sin esta herramienta los datos no serían aprovechados al máximo, de ahí deriva la importancia de conocer este lenguaje y la relación que guarda con los modelos relacionales de las bases de datos.

¿Qué es Postgresql?

Sistema de gestión de base de datos relacional orientada a objetos de software libre, publicado bajo la licencia BSD. Como muchos otros proyectos open source, el desarrollo de PostgreSQL no es manejado por una sola compañía, sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Características de Postgresql

- Alta concurrencia: mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés).
- Integridad de los datos: claves primarias, llaves foráneas con capacidad de actualizar en cascada o restringir la acción y restricción not null.
- Resistencia a fallas. Escritura adelantada de registros (WAL) para evitar pérdidas de datos en caso de fallos por: Energía, Sistema Operativo, Hardware.

Características de Postgresql

- Multiplataforma. Linux, Unix, BSD's, Mac OS X, Solaris, AIX, Irix, HP-UX, Windows.
- PITR. Puntos de recuperación en el tiempo.
- Tablespaces. (Ubicaciones alternativas para los datos)
- Replicación síncrona y asincrónica.
- Tipos de datos No-SQL

Características de Postgresql

- Definir funciones personalizadas por medio de varios lenguajes:
  - PL/pgSQL, PL/Tcl, PL/Perl, PL/Python, PL/PHP, PL/Ruby, PL/Java

Caracteristicas de Postgresql

- Es más lento en inserciones y actualizaciones, ya que cuenta con cabeceras de intersección.
- Sin experticia, configurar llega a ser un caos.
- InnoDB genera mucho footprint en memoria al indizar.
- El toolset empresarial tiene un costo adicional por suscripción anual.
- Reducida cantidad de tipos de datos.
- No soporta consultas en paralelo hasta la versión 9.6

Caracteristicas de Postgresql

Base de datos	Ilimitado reportada una de 32 tb
Tabla	32 TB
Fila	400 GB
Campo	1 GB
Filas en una tabla	Ilimitado
Columnas en una tabla(dependiendo de los tipos de datos)	250-1600
Índices de una tabla	Ilimitado

Tipos de datos usados por los RDBMS

Sinónimos de tipos de Dato	Tamaño	Descripción
BINARY, VARBINARY, BINARY , VARYING, BIT VARYING	1 byte por carácter	Permite almacenar imágenes, videos, entre otros .
BIT, BOOLEAN, LOGICAL, LOGICAL1, YESNO	1 byte	Valores Sí y No, y campos que contienen solamente uno de dos valores.
BYTE, INTEGER1, TINYNIT	1 byte	Un número entero entre 0 y 255.
COUNTER , AUTOINCREMENT, SEQUENCE		Se utiliza para campos contadores cuyo valor se incrementa automáticamente al crear un nuevo registro.
MONEY , CURRENCY	8 bytes	Un número entero comprendido entre – 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME , DATE , TIME	8 bytes	Una valor de fecha u hora entre los años 100 y 9999
DECIMAL , NUMERIC, DEC	17 bytes	Un tipo de datos numérico exacto con valores comprendidos entre 1028 - 1 y -1028 - 1. Puede definir la precisión (1 - 28) y la escala (0 - precisión definida). La precisión y la escala predeterminadas son 18 y 0, respectivamente.
CHAR, TEXT(n), ALPHANUMERIC, CHARACTER, STRING, VARCHAR, CHARACTER VARYING, NCHAR, NATIONAL CHARACTER, NATIONAL CHAR, NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING	2 bytes por carácter.	Desde cero a 255 caracteres.

Tipos de Datos **PostgreSQL**

Nombre	Tamaño	Rango
smallint	2 bytes	De -32768 a +32767
integer	4 bytes	De -2147483648 a +2147483647
bigint	8 bytes	De -9223372036854775808 a 9223372036854775807
decimal	variable	Sin límite
numeric	variable	Sin límite
real	4 bytes	6 dígitos decimales de precisión
double precision	8 bytes	15 dígitos decimales de precisión
serial	4 bytes	De 1 a 2147483647
bigserial	8 bytes	De 1 a 9223372036854775807

Nombre	Descripción
character varying(n), varchar(n)	De longitud variable, con límite
character(n), char(n)	De longitud fija
text	De longitud variable, ilimitado

Nombre	Tamaño	Descripción
bytea	4 bytes además de la cadena binaria actual	Cadena binaria de longitud variable

Tipos de Datos **PostgreSQL**

Nombre	Tamaño	Descripción	Valor bajo	Valor alto	Resolución
timestamp [(*p*)] [sin zona horaria]	8 bytes	Fecha y hora	4713 BC	5874897 AD	1 microsegundo / 14 dígitos
timestamp [(*p*)] con zona horaria	8 bytes	Fecha y hora con zona horaria	4713 BC	5874897 AD	1 microsegundos / 14 dígitos
interval [(*p*)]	12 bytes	Intervalo de hora	-178000000 años	178000000 años	1 microsegundo
date	4 bytes	Sólo fecha	4713 BC	32767 AD	1 día
time [(*p*)] [sin zona horaria]	8 bytes	Sólo hora del día	00:00:00.00	23:59:59.99	1 microsegundo
time [(*p*)] con zona horaria	12 bytes	Horas del día con zona horaria	00:00:00.00+12	23:59:59.99-12	1 microsegundo

Componentes (DML, DDL, DCL)

DDL Lenguaje de Definición de Datos

DDL o Lenguaje de Definición de Datos: Se utiliza para crear, eliminar o modificar tablas, índices, vistas, triggers, procedimientos; es decir, nos permite definir la estructura de la base de datos mediante comandos como crear (*CREATE*), eliminar (*DROP*), o modificar (*ALTER*).

- **CREATE** Sirve para crear objetos en la base de datos.
- **ALTER** Permite modificar la estructura de un objeto.
- **DROP** Permite eliminar objetos de la base de datos.
- **TRUNCATE** Elimina todos los registros de una tabla.

CREATE

Utilizado para crear nuevas bases de datos, tablas, índices, vistas, defaults, reglas, procedimientos, funciones, triggers.

- CREATE DATABASE
- CREATE DEFAULT
- CREATE FUNCTION
- CREATE PROCEDURE
- CREATE RULE
- CREATE TABLE
- CREATE VIEW

CREATE DATABASE

```
CREATE DATABASE new_db
 WITH OWNER = usuario
 ENCODING = 'UTF8'
 TEMPLATE = postgres;
```

CREATE TABLE

```
CREATE TABLE nombre_tabla (
        atributo tipoDeDato restricciones,
        ...          ...                ... , ...
);

CREATE TABLE  entidad_federativa_copia  (
        entidad_federativa_id int(11) NOT NULL AUTO_INCREMENT,
        entFederativa_clave char(2) NOT NULL,
        entFederativa_nombre varchar(25) NOT NULL,
        PRIMARY KEY (entidad_federativa_id)
)
ENGINE = InnoDB
AUTO_INCREMENT = 33
DEFAULT CHARSET = latin1
COLLATE = latin1_spanish_ci;
```

CREATE

```
CREATE TABLE pais (
        pais_id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
        pais_clave VARCHAR(5) NOT NULL,
        pais_nombre VARCHAR(120) NOT NULL,
        pais_nacionalidad VARCHAR(50),
        PRIMARY KEY (pais_id)
)
ENGINE = InnoDB;

CREATE TABLE institucion
(
        inst_clave integer NOT NULL,
        inst_nombre VARCHAR (40) NOT NULL,
        inst_siglas VARCHAR (5),
        PRIMARY KEY (inst_clave)
) ENGINE = InnoDB;
```

ALTER

Utilizado para modificar la estructura de una tabla para agregar campos o constraints, también se utiliza para modificar algunas características globales de las bases de datos.

- ALTER TABLE
- ALTER DATABASE

ALTER TABLE

	Ejemplo
Modificar una columna	ALTER TABLE t1 **MODIFY** b BIGINT NOT NULL;
Agregar una columna	ALTER TABLE t2 **ADD** d TIMESTAMP;
Agregar una llave primaria	ALTER TABLE t2 **ADD** PRIMARY KEY (c);
Renombrar la tabla	ALTER TABLE t1 **RENAME** t2;
Agregar la restricción de NOT NULL	ALTER TABLE t2 **MODIFY** a TINYINT NOT NULL;
Quitar la restricción de NOT NULL	ALTER TABLE t2 **MODIFY** a TINYINT;
Renombrar un atributo	ALTER TABLE t1 **CHANGE** a b INTEGER;
Agregar una restricción de llave foránea	ALTER TABLE nombreTabla **ADD** [CONSTRAINT *symbol*] **FOREIGN KEY** [*id*] (*index_col_name*, ...) **REFERENCES** *tbl_name* (*index_col_name*, ...) [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}] [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
Quitar una columna	ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
Quitar una restricción de llave foránea	ALTER TABLE *nombreTabla* **DROP FOREIGN KEY** *fk_symbol*;

DROP

Utilizado para eliminar bases de datos, tablas, campos, índices, vistas, defaults, reglas, procedimientos, funciones, triggers.

- DROP DATABASE
- DROP DEFAULT
- DROP FUNCTION
- DROP PROCEDURE
- DROP RULE
- DROP TABLE
- DROP VIEW

DROP

DROP TABLE nombre_tabla;
DROP TABLE IF EXISTS institucion**;**

IF EXISTS No devuelve un error cuando no existe la tabla. En estos casos devuelve una notificación (NOTICE).

name El nombre (opcionalmente con el esquema) de la tabla a eliminar.

DML o Lenguaje de Manipulación de Datos

DML o Lenguaje de Manipulación de Datos: Se utiliza para realizar la consulta y edición de la información contenida en la base de datos, esto implica: **seleccionar, insertar, modificar y borrar**.

Los DML se distinguen por sus sublenguajes de recuperación subyacentes; se pueden distinguir dos tipos de DML, el procedural y el no procedural. La principal diferencia entre ambos, es que los lenguajes procedurales tratan a los registros individualmente, mientras que los no procedurales operan a un conjunto de registros. Las instrucciones relacionadas con este componente son:

- **SELECT** Permite **realizar consultas** a la base de datos.
- **INSERT** Empleado para **agregar registros** a una tabla.
- **UPDATE** Utilizado para **modificar los valores** de los campos de una tabla.
- **DELETE** Utilizado para **eliminar** los registros de una tabla.

DCL Lenguaje de control de datos

DCL o Lenguaje de Control de Datos: Se utiliza para la definición de los privilegios de control de acceso y edición a los elementos que componen la base de datos (seguridad), es decir, permitir o revocar el acceso.

Los permisos a nivel base de datos pueden otorgarse a usuarios para ejecutar ciertos comandos dentro de la base o para que puedan manipular objetos y los datos que puedan contener estos.

Las instrucciones relacionadas con este componente son:

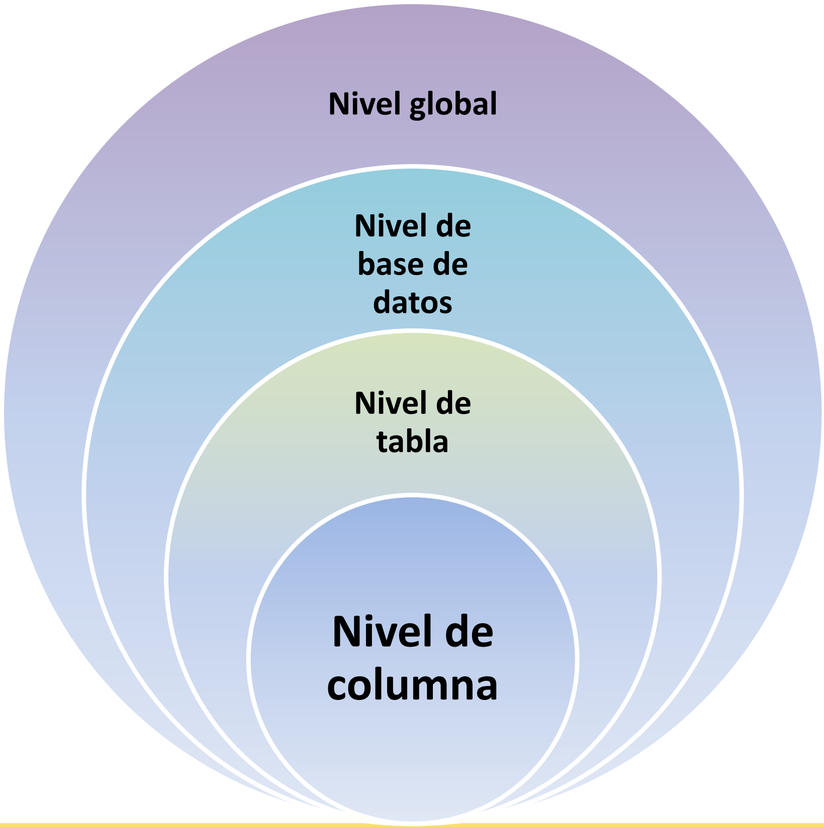
-**grant.** Permite otorgar permisos a los usuarios sobre los objetos definidos en la base de datos, así como las operaciones a utilizar sobre ellos.

-**revoke.** Permite revocar permisos sobre los objetos definidos en la base de datos y las operaciones sobre los mismos.

Instrucciones DCL

- **GRANT** – Permite crear cuentas de usuario y otorgarles privilegios.
- **REVOKE** – Permite quitar privilegios otorgados con el comando GRANT.

Niveles de permisos



Instrucciones DCL

Permiso	Significado
ALL [PRIVILEGES]	Da todos los permisos simples excepto GRANT OPTION
ALTER	Permite el uso de ALTER TABLE
ALTER ROUTINE	Modifica o borra rutinas almacenadas
CREATE	Permite el uso de CREATE TABLE
CREATE ROUTINE	Crea rutinas almacenadas
CREATE TEMPORARY TABLES	Permite el uso de CREATE TEMPORARY TABLE
CREATE USER	Permite el uso de CREATE USER, DROP USER, RENAME USER, y REVOKE ALL PRIVILEGES.
CREATE VIEW	Permite el uso de CREATE VIEW
DELETE	Permite el uso de DELETE
DROP	Permite el uso de DROP TABLE
EXECUTE	Permite al usuario ejecutar rutinas almacenadas
FILE	Permite el uso de SELECT ... INTO OUTFILE y LOAD DATA INFILE
INDEX	Permite el uso de CREATE INDEX y DROP INDEX
INSERT	Permite el uso de INSERT
LOCK TABLES	Permite el uso de LOCK TABLES en tablas para las que tenga el permiso SELECT

Instrucciones DCL

Permiso	Significado
PROCESS	Permite el uso de SHOW FULL PROCESSLIST
REFERENCES	No implementado
RELOAD	Permite el uso de FLUSH
REPLICATION CLIENT	Permite al usuario preguntar dónde están los servidores maestro o esclavo
REPLICATION SLAVE	Necesario para los esclavos de replicación (para leer eventos del log binario desde el maestro)
SELECT	Permite el uso de SELECT
SHOW DATABASES	SHOW DATABASES muestra todas las bases de datos
SHOW VIEW	Permite el uso de SHOW CREATE VIEW
SHUTDOWN	Permite el uso de **mysqladmin shutdown**
SUPER	Permite el uso de comandos CHANGE MASTER, KILL, PURGE MASTER LOGS, and SET GLOBAL , el comando **mysqladmin debug** le permite conectar (una vez) incluso si se llega a max_connections
UPDATE	Permite el uso de UPDATE
USAGE	Sinónimo de “no privileges”
GRANT OPTION	Permite dar permisos

https://www.postgresql.org/docs/9.0/static/sql-grant.html

Instrucciones DCL - **GRANT**

GRANT *priv_type* [(*column_list*)] [, *priv_type* [(*column_list*)]] ... ON [*object_type*] {*tbl_name* | * | *.* | *db_name*.*} TO *user* [IDENTIFIED BY [PASSWORD] '*password*']

GRANT ALL PRIVILEGES ON cursosql.* TO sql12@localhost IDENTIFIED BY '$q1012' WITH GRANT OPTION;

GRANT ALL PRIVILEGES ON cursosql.* TO 'sql12'@'%' IDENTIFIED BY '$q1012' WITH GRANT OPTION;

GRANT ALL PRIVILEGES ON cursosql.* TO sql12@132.248.81.242 IDENTIFIED BY '$q1012' WITH GRANT OPTION;

- CREATE USER *user* [IDENTIFIED BY [PASSWORD] '*password*']

Instrucciones DCL - **GRANT**

REVOKE *priv_type* [(*column_list*)] [, *priv_type* [(*column_list*)]] ... ON
[*object_type*] {*tbl_name* | * | *.* | *db_name*.*} FROM *user* [, *user*] ...

REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';

REVOKE ALL PRIVILEGES, GRANT OPTION FROM *user* [, *user*] ...

Instrucciones DML - **SELECT**

SELECT [DISTINCT | * | expresión | columna [AS output_name] [, ...]
[**FROM** tabla [, ...]]
[**WHERE** condición]

SELECT – Indica que campos o datos va a devolver la consulta.
FROM – Indica a partir de que tablas, vistas o subconsultas se obtienen los datos.
WHERE – Indica las condiciones que deben cumplir los datos.

- Las sentencias se pueden escribir en una o varias líneas.
- Las clausulas no son sensitivas al cambio de mayúsculas o minúsculas.
- Se recomienda que cada clausula se coloque en una línea distinta e indentarlas para mayor legibilidad.

Instrucciones DML - **SELECT**

```
SELECT * FROM tabla;
SELECT campo1, campo2 FROM tabla;
SELECT 1 + 1;

SELECT al_numcta, al_nombre, al_apellidoPat, al_apellidoMat
FROM alumno;

SELECT * FROM entidad_federativa;
SELECT entFederativa_clave AS clave, entFederativa_nombre AS Nombre
FROM entidad_federativa;
```

Instrucciones DML - **SELECT**

Al incluir la cláusula **DISTINCT** en un SELECT, se **eliminan** los registros repetidos del resultado

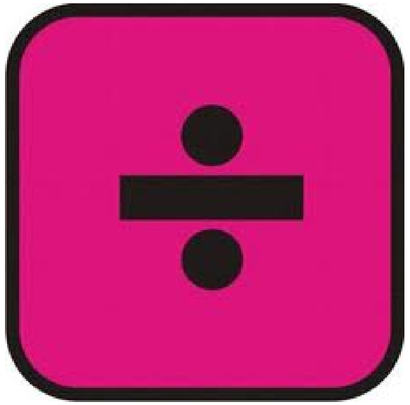
```
SELECT DISTINCT campo1, campo2 …
FROM tabla1
WHERE
```

```
SELECT DISTINCT al_nombre, al_apellidoPat
FROM alumno;
```

```
SELECT DISTINCT al_genero
FROM alumno;
```

Instrucciones DML – SELECT – operadores aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División



Criterios de selección

La clausula WHERE se utiliza para restringir los registros devueltos por la consulta.

```
SELECT *
FROM
WHERE …..;
```

```
SELECT * FROM alumno
WHERE  alumno_id = 10;
```

```
SELECT * FROM alumno
WHERE al_nombre = 'Martha';
```

```
SELECT * FROM alumno
WHERE al_fechaNac = '1998-10-10';
```

El valor **NULO**

Un valor nulo se representa en SQL con la cláusula NULL y representa la ausencia de información. Frecuentemente un valor nulo es confundido con un valor numérico de 0 o una cadena vacía.

Esto es importante recordarlo cuando se deseen realizar ciertas operaciones, por ejemplo un promedio de edades, dado que la función para determinar el promedio no contemplará valores nulos.

Edad 1 25
Edad 2 NULL
Edad 3 30
Edad 4 NULL
Promedio: 27.5

Promedio = (25 + 30) / 2 = 55 / 2 = 27.5

En el ejemplo anterior se puede confirmar que **los valores nulos no son considerados como cero**.

Hasta este momento se ha hablado únicamente de tipos de datos numéricos, aunque los valores nulos también se utilizan en cualquier otro tipo de dato, por ejemplo en texto: **No es lo mismo una cadena vacía que un valor nulo.**

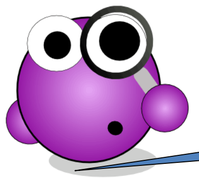
Operadores de comparación

Operador	Descripción
=	Igual a
<	Menor que
<=	Menor que o igual
>	Mayor que
>=	Mayor que o igual
<>	Diferente de
BETWEEN ...AND…	Entre dos valores (incluyente)
IN(set)	Corresponde a algunos de los valores de la lista
LIKE	Corresponde con un patrón de caracteres
IS NULL	Es un valor nulo

Ejemplos con operadores de comparación

```
SELECT * FROM alumno
WHERE alumno_id < 10;
SELECT * FROM alumno
WHERE  al_fechaNac > '1981-01-01';
SELECT * FROM alumno
WHERE  alumno_id between 5 AND 10;
SELECT * FROM alumno
WHERE  al_nombre LIKE '%o%';
SELECT * FROM alumno
WHERE  al_nombre LIKE 'M%';
SELECT * FROM alumno
WHERE  al_nombre LIKE 'M';
```

```
SELECT * FROM alumno
WHERE alumno_id != 10;
SELECT * FROM alumno
WHERE alumno_id <> 10;
SELECT * FROM alumno
WHERE al_fechaNac
between '1981-01-01' AND '1999-12-31';
SELECT * FROM alumno
WHERE  al_apellidoMat IS NULL;
SELECT * FROM alumno
WHERE  alumno_id IN (5, 10, 11);
```



✓ La comparación campo **IS NULL** es correcta.

✗ La comparación campo = NULL es incorrecta.

Operadores lógicos
Predicados y conectores

Operador	Descripción
AND	Devuelve verdadero si ambas condiciones se cumplen.
OR	Devuelve verdadero si cualquiera de las condiciones se cumple.
NOT	Devuelve verdadero si la siguiente condición es falsa.

Ejemplos de operadores lógicos

```
SELECT * FROM alumno
WHERE  al_nombre LIKE '%a%' AND al_apellidoPat LIKE '%e%' ;

SELECT * FROM alumno
WHERE  al_nombre = 'Juan' OR al_nombre = 'Pedro';

SELECT al_nombre, al_genero,  entidad_federativa_id
FROM alumno
WHERE  al_nombre= 'Martha' AND (entidad_federativa_id IS NULL OR entidad_federativa_id = 9) ;

SELECT * FROM alumno
WHERE  alumno_id  NOT IN (5, 10, 11);

SELECT * FROM alumno
WHERE  al_nombre NOT LIKE '%o%';

SELECT * FROM alumno
WHERE  al_apellidoMat IS NOT NULL;
```

Ejemplos de operadores lógicos

Permite ordenar los resultados de una consulta.

ORDER BY expression [ASC | DESC]

```
SELECT * FROM alumno
 ORDER BY al_apellidoPat, al_apellidoMat, al_nombre ;

SELECT * FROM alumno
 ORDER BY al_apellidoPat ASC, al_apellidoMat ASC, al_nombre ASC;

 SELECT * FROM alumno
 ORDER BY al_apellidoPat DESC, al_apellidoMat DESC, al_nombre DESC ;

SELECT al_apellidoPat, al_apellidoMat, al_nombre, al_fechaNac
FROM alumno
ORDER BY al_apellidoPat, al_apellidoMat, al_nombre ;

SELECT al_apellidoPat, al_apellidoMat, al_nombre, al_fechaNac
FROM alumno
ORDER BY al_apellidoPat, 2, al_nombre ;
```

Índices

Los índices permiten encontrar más rápidamente los registros en una base de datos.
Con los índices el manejador puede determinar en menos tiempo la posición a partir de la cual se puede buscar, en lugar de buscar en todos los datos.

Algunos de los **usos** de los índices son:
Para obtener los registros que corresponden a la cláusula **WHERE** rápidamente.
Para disminuir el número de registros a considerar.
Para obtener registros de otras tablas cuando se realiza un JOIN.
Para ordenar o agrupar una tabla cuando alguno de los campos .

Desventajas:
Ocupan espacio

Índices

CREATE [UNIQUE|FULLTEXT|SPATIAL] **INDEX** *index_name* [USING *index_type*] **ON** *tbl_name* (*index_col_name*,...)

- **CREATE UNIQUE INDEX** al_idx **ON** alumno (al_numcta);
- **DROP INDEX** al_idx **ON** alumno;

Vistas

Es el resultado de ejecutar una consulta en una o varias tablas.

- De las vistas **solamente se almacena la definición**, no los datos.
- La sentencia SELECT no puede contener una subconsulta en su cláusula FROM.
- Cualquier **tabla o vista** referenciada por la definición **debe existir**.
- **No se puede asociar un trigger** con una vista.
- **Algunas vistas permites actualizar datos en la tabla subyacente**, siempre y cuando se trate de:
  - una sola tabla
  - exista una relación de 1 a 1 con los campos de la tabla
  - además de cumplir con otras restricciones como no tener agrupados, DISTINCT entre otros.

Vistas

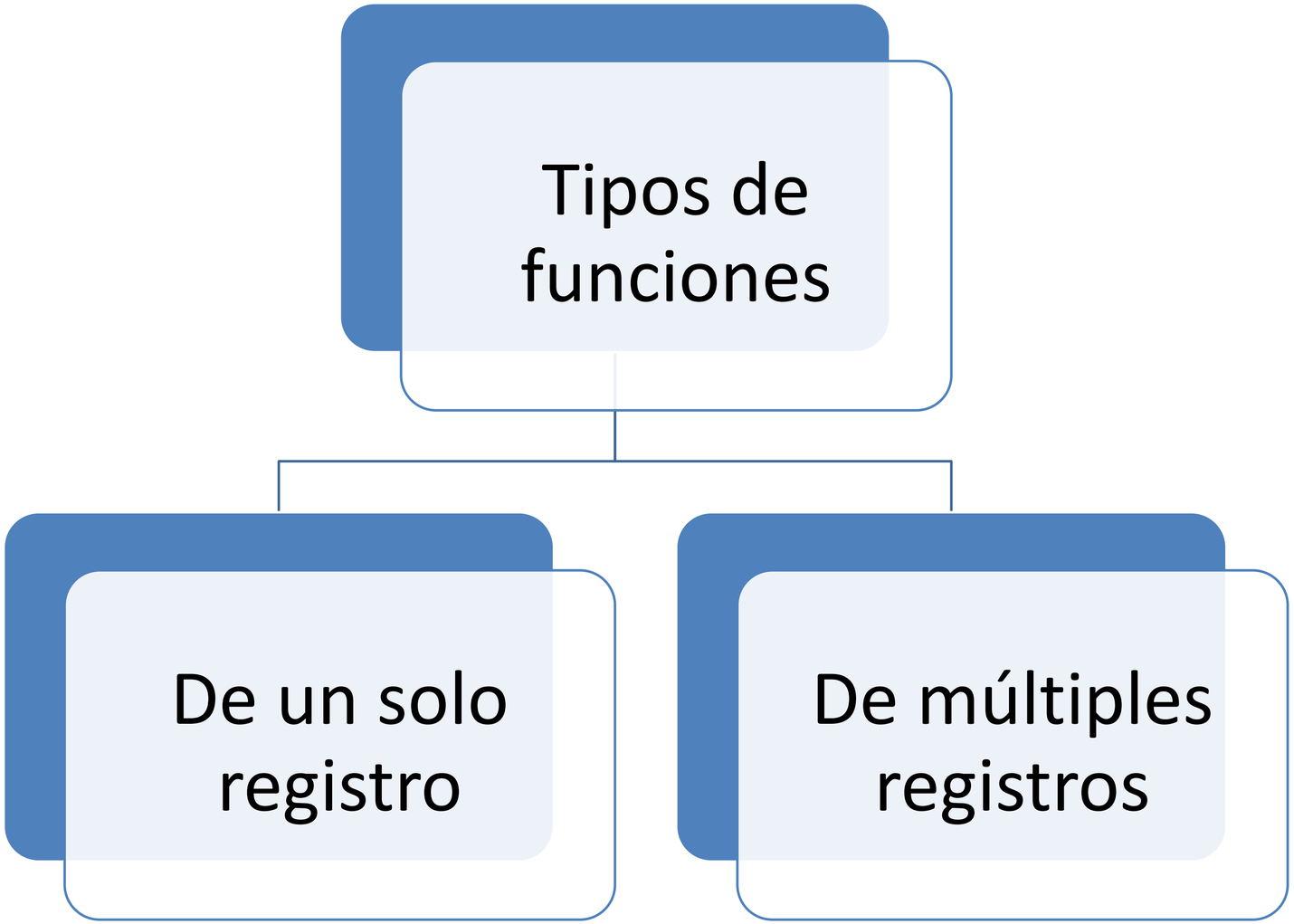
- **CREATE** [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] **VIEW** *nombre_vista* [(*columnas*)] **AS** *sentencia_select* [WITH [CASCADED | LOCAL] CHECK OPTION]

- **CREATE VIEW** alumno_v **AS** SELECT * FROM alumno;

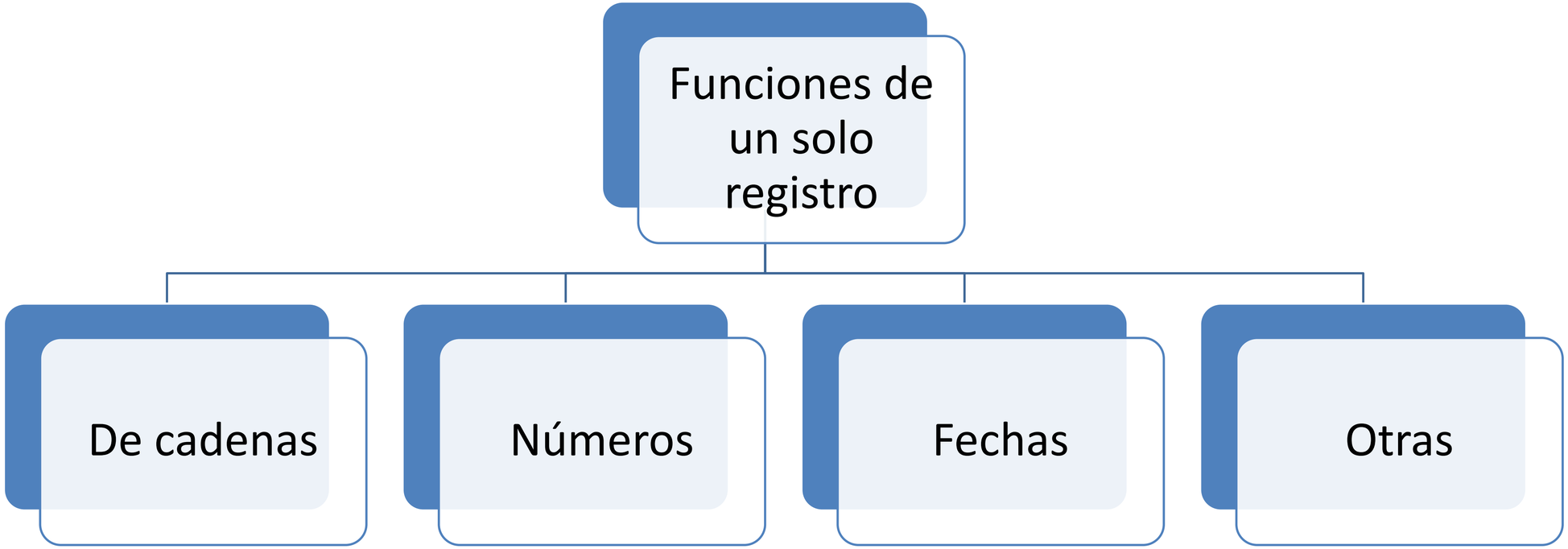
- **CREATE OR REPLACE VIEW** alumno_v **AS** SELECT al_numcta, al_nombre, al_apellidoPat, al_apellidoMat FROM alumno;

- **DROP VIEW** alumno_v;

Tipos de funciones



Funciones de un solo registro



Funciones de cadenas

Función	Descripción	Ejemplo
ASCII(*str*)	Devuelve el valor numérico del carácter más a la izquierda de la cadena de caracteres *str*. Devuelve 0 si *str* es la cadena vacía. Devuelve NULL si *str* es NULL. ASCII() funciona para caracteres con valores numéricos de 0 a 255.	SELECT ASCII('2'); -> 50 SELECT ASCII('@'); ->64 SELECT ASCII(null); -> null
CONCAT(*str1*,*str2*,...)	Devuelve la cadena resultado de concatenar los argumentos. Devuelve NULL si algún argumento es NULL. Puede tener uno o más argumentos.	SELECT CONCAT('Ro', 'ber', 'to'); ->'Roberto'
INSTR(*str*,*substr*)	Devuelve la posición de la primera ocurrencia de la subcadena *substr* en la cadena *str*	SELECT INSTR('foobarbar', 'bar'); ->4
LENGTH(*str*)	Devuelve la longitud de la cadena *str*, medida en bytes.	SELECT LENGTH('hola'); ->4
LOWER(*str*)	Devuelve la cadena en minúsculas.	SELECT LOWER('HOLA'); ->'hola'
LPAD(*str*,*len*,*padstr*)	Devuelve la cadena original agregándole a la izquierda la cadena padstr hasta la longitud indicada.	SELECT LPAD('hola',11,'ab'); ->'abababahola'
LTRIM(*str*)	Devuelve la cadena *str* con los caracteres en blanco iniciales eliminados.	SELECT LTRIM(' holahola'); ->'holahola'

Funciones de cadenas

Función	Descripción	Ejemplo
REPLACE(*str*,*from_str*,*to_str*)	Devuelve la cadena *str* con todas las ocurrencias de la cadena *from_str* reemplazadas con la cadena *to_str*.	SELECT REPLACE('hola mundo', 'hola', 'adios'); ->'adios mundo'
RPAD(*str*,*len*,*padstr*)	Devuelve la cadena *str*, alineada a la derecha con la cadena *padstr* con una longitud de *len* caracteres. Si *str* es mayor que *len*, el valor de retorno se corta a *len* caracteres.	SELECT RPAD('hola',10,'bb'); ->'holabbbbbb'
RTRIM(*str*)	Devuelve la cadena *str* con los espacios a la derecha eliminados.	SELECT LTRIM(' holahola '); ->'holahola'
SUBSTRING(*str*,*pos*,*len*)	Devuelven una subcadena de la cadena *str* comenzando en la posición *pos, el valor len es opcional y se utiliza para indicar cuantos caracteres debe regresar.*	SELECT SUBSTRING('hola',2); ->'ola' SELECT SUBSTRING('hola',2,2); ->'ol'
TRIM(*str*)	Elimina los espacios en blanco a la izquierda y derecha de la cadena.	SELECT TRIM(' holahola '); ->'holahola'
UPPER(*str*)	Devuelve la cadena en mayúsculas	SELECT UPPER('hola'); ->'HOLA'

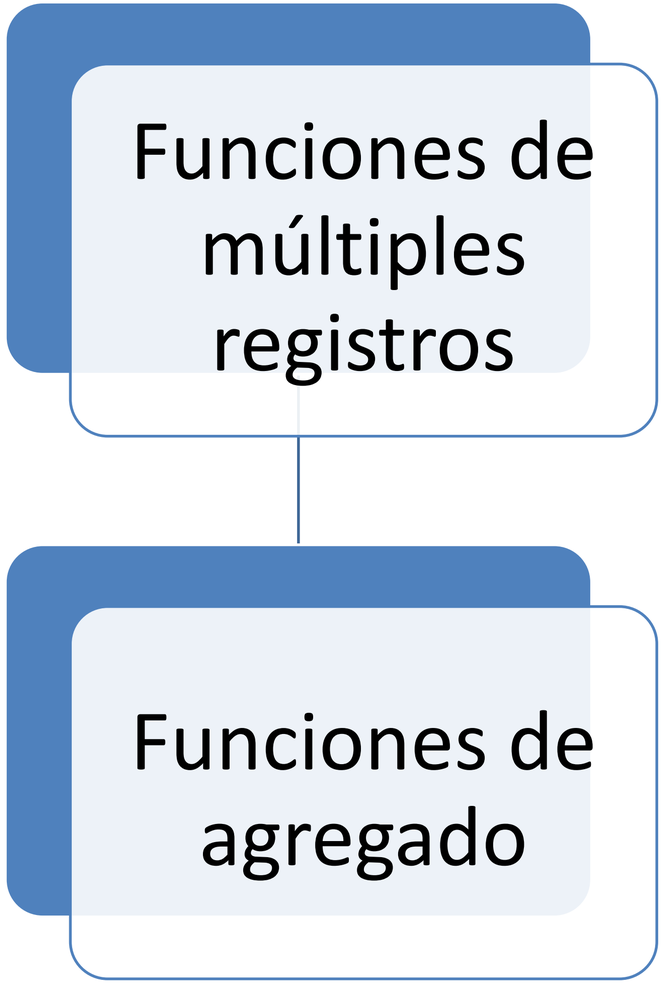
Funciones de números

Función	Descripción	Ejemplo
CEILING(*X*), CEIL(*X*)	Devuelve el valor del siguiente entero mayor que X.	SELECT CEILING(5.28); ->6 SELECT CEIL(-5.28); -> -5
FLOOR(*X*)	Devuelve el valor del siguiente entero menor que X.	SELECT FLOOR(5.28); ->5 SELECT FLOOR(-5.28); -> -6
MOD(*N*,*M*)	Operación de módulo. Retorna el resto de *N* dividido por *M*.	SELECT MOD(234, 10); -> 4
ROUND(*X*)	Retorna el argumento *X*, redondeado al entero más cercano.	SELECT ROUND(5.28); ->5 SELECT ROUND(5.58); -> 6

Funciones de fechas

Función	Descripción	Ejemplo
CURRENT_DATE()	Regresan los valores de la zona horaria de la conexión.	SELECT CURRENT_DATE(); ->'2012-06-14'
ADDDATE(*date*, INTERVAL *expr type*)	Regresa la fecha que da como resultado agregar el periodo de tiempo indicado.	SELECT ADDDATE ('2012-01-02', INTERVAL 31 DAY); ->'2012-02-02‘ SELECT ADDDATE('2012-01-02', INTERVAL 31 MONTH); SELECT ADDDATE('2012-01-02', 31);
CURTIME()	Retorna la hora actual como valor en formato 'HH:MM:SS'	SELECT CURTIME(); ->11:19:16
DATEDIFF(*expr*,*expr2*)	retorna el número de días entre la fecha inicial *expr* y la fecha final *expr2*.	SELECT DATEDIFF('2012-06-08', '2012-06-06'); ->2 SELECT DATEDIFF('2012-06-06', '2012-06-08'); ->-2
DATE_FORMAT(*date*,*format*)	Da formato a la fecha.	SELECT DATE_FORMAT('2012-10-04 22:23:00', '%W %M %Y'); ->'Thursday October 2012' SELECT DATE_FORMAT('2012-10-04 22:23:00', '%d/%m/%Y');

Funciones de múltiples registros



Funciones de agregación

Función	Tipo de dato que recibe	Tipo de dato que devuelve	Descripción
avg(expression)	smallint, int, bigint, real, double precision, numeric, or interval	numeric para integer, double precision para floating-point, otro el mismo tipo de dato que el argumento	El promedio de todos los valores de entrada
count(*)		bigint	Retorna un contador del número de registros, contengan o no valores NULL
count(expression)	any	bigint	Retorna el contador del número de valores no NULL
max(expression)	any array, numeric, string, or date/time type	El mismo tipo de dato que el argumento	Regresa el valor máximo entre todos los valores de entrada
min(expression)	any array, numeric, string, or date/time type	El mismo tipo de dato que el argumento	Regresa el valor mínimo entre todos los valores de entrada
stddev(expression)	smallint, int, bigint, real, double precision, or numeric	double precision for floating-point arguments, otherwise numeric	Devuelve la desviación estándar de los valores de entrada
sum(*expr*)	smallint, int, bigint, real, double precision, numeric, or interval	bigint for smallint or int arguments, numeric for bigint arguments, double precision for floating-point arguments, otherwise the same as the argument data type	Retorna la suma de *expr*

Funciones de agregación

- SELECT avg(pro_salario) FROM profesor;
- SELECT count(*) FROM alumno;
- SELECT count(entidad_federativa_id) FROM alumno;
- SELECT count(DISTINCT entidad_federativa_id) FROM alumno;
- SELECT max(alumno_id) FROM alumno;
- SELECT min(alumno_id) FROM alumno;
- SELECT stddev(pro_salario) FROM profesor;
- SELECT sum(pro_salario) FROM profesor;

Group by

Condensa en un solo registro, todos los registros seleccionados que comparten los mismos valores de la expresión de agrupado.

GROUP BY expresión [, ...]

```
SELECT min(pro_salario) as salario_minimo, p.pro_genero
FROM profesor p
GROUP BY p.pro_genero;

SELECT count(*), a.al_genero as genero, a.entidad_federativa_id
FROM alumno a
GROUP BY a.al_genero, a.entidad_federativa_id ;

SELECT count(*), a.al_genero, a.entidad_federativa_id
FROM alumno a
GROUP BY a.al_genero, 3;
```

Having

Elimina grupos de registros que no satisfacen una condición

HAVING condition

```
SELECT COUNT(*), entidad_federativa_id FROM alumno
GROUP BY  entidad_federativa_id
HAVING COUNT(*) > 2;
```

```
SELECT p.pro_genero , p.pro_salario , COUNT(*)
FROM profesor p
GROUP BY  1, p.pro_salario
HAVING avg(p.pro_salario) > 2000;
```

Having Y Where

```
SELECT p.pro_genero , p.pro_salario,
  COUNT(*)
FROM (SELECT * FROM PROFESOR WHERE
  profesor_id > 5) p
WHERE p.pro_genero ='H'
GROUP BY p.pro_genero, p.pro_salario
HAVING avg(p.pro_salario) > 2000 AND
  MIN(p.pro_salario)> 1000;
```

JOIN

Construye una relación formada por todas las eneadas que aparecen en cualquiera de las dos relaciones especificadas en que se cumple alguna condición en dominios comunes. Se obtiene concatenando una eneada de r con otra de q, de forma que cumpla con una condición en los dominios comunes. Si no hay dominios comunes, esta operación es un producto cartesiano.

Sean q y r dos conjuntos como siguen:

q

A	B	C
a1	b1	c1
a2	b2	c1
a2	b1	c2

r

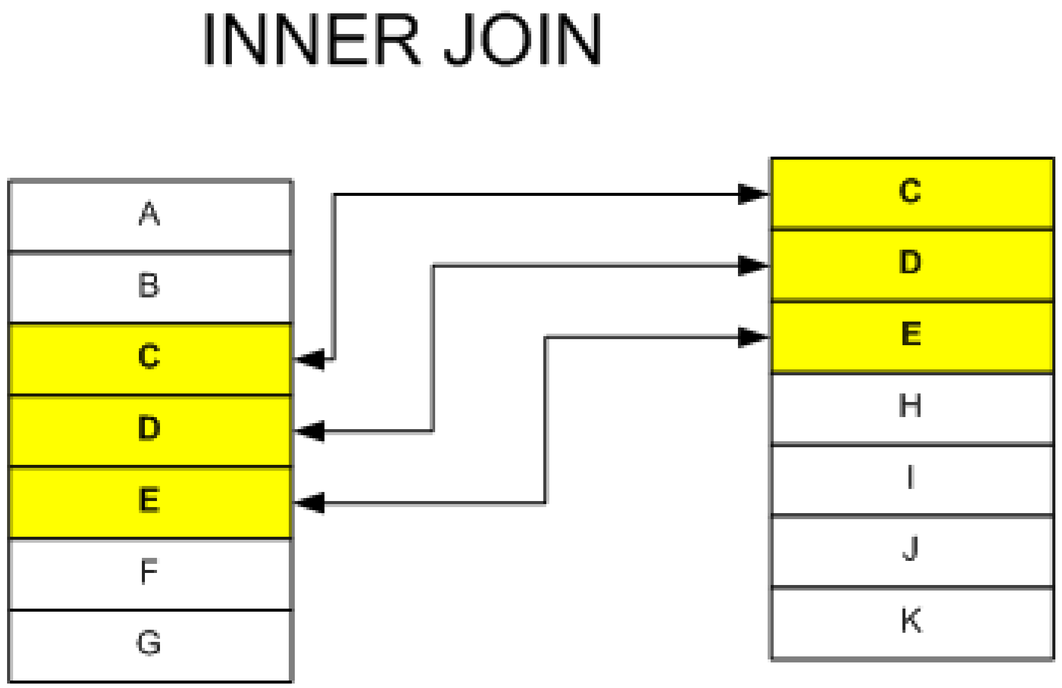
C	D	E
c1	d1	e1
c1	d2	e2
c3	d2	e2

Obtendríamos:

A	B	C	C	D	E
a1	b1	c1	c1	d1	e1
a2	b2	c1	c1	d2	e2

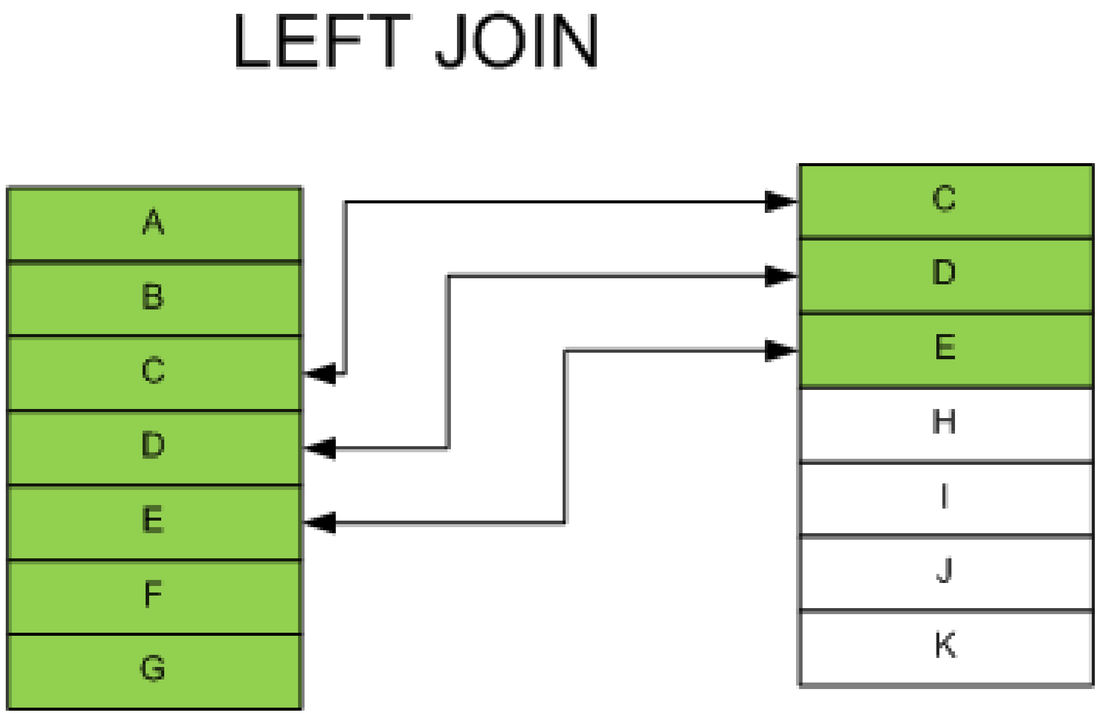
INNER JOIN

[INNER] **JOIN** Todos los valores de las filas del resultado son valores que están en las tablas que se combinan.



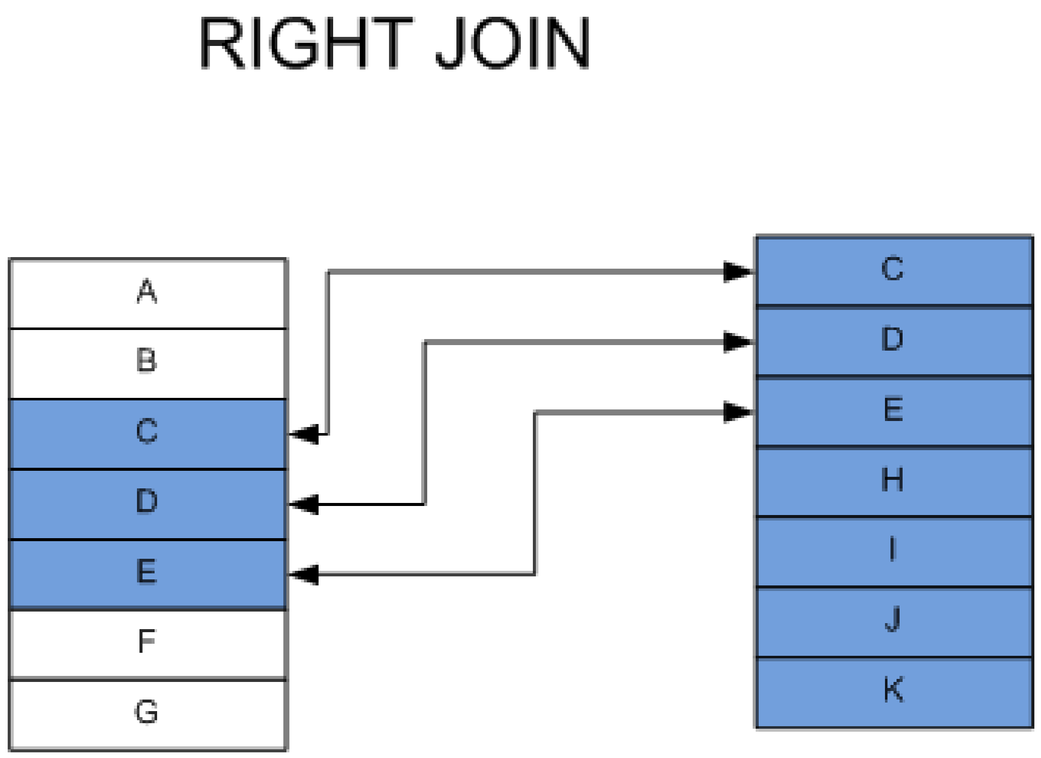
LEFT JOIN

LEFT [OUTER] **JOIN**- Devuelve todos los registro de la tabla a la izquierda, **aunque no tengan una fila coincidente** en la otra tabla.



RIGHT JOIN

RIGHT [OUTER] **JOIN** - Devuelve todos los registro de la tabla a la derecha, **aunque no tengan una fila coincidente** en la otra tabla.



JOIN

- [INNER] JOIN

```
SELECT a.*, ef.* from alumno a JOIN entidad_federativa ef ON (a.entidad_federativa_id =
  ef.entidad_federativa_id);
```

- LEFT [OUTER] JOIN

```
SELECT a.*, ef.* from alumno a LEFT JOIN entidad_federativa ef ON
  (a.entidad_federativa_id = ef.entidad_federativa_id);
```

- RIGHT [OUTER] JOIN

```
SELECT a.alumno_id, a.al_numcta, a.al_nombre, a.al_apellidoPat, a.al_apellidoMat,
  a.entidad_federativa_id, ef.entidad_federativa_id, ef.entFederativa_nombre from alumno a
  RIGHT JOIN entidad_federativa ef ON (a.entidad_federativa_id =
  ef.entidad_federativa_id);
```

Ejemplo JOINs

```
SELECT employee_id, city, department_name
FROM employees e
LEFT JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

Instrucciones DML - **INSERT**

- INSERT INTO table [(column [, ...])] { DEFAULT VALUES | VALUES ({ expression | DEFAULT } [, ...]) | query }
- Cláusula INTO
  - Permite indicar en que tabla se van a agregar los datos
  - El nombre de los campos es opcional. Si se omite se deben indicar los valores de todos los campos en el orden en que están en la tabla.

- Cláusula VALUES
  - Para expresar un valor de tipo alfanumérico o fecha, es necesario escribirlo entre comillas simples.
  - Los valores numéricos se escriben sin comillas.
  - Los datos alfanuméricos que no se incluyan entre comillas simples, el manejados de bases de datos intentará interpretarlos como funciones o objetos de la base de datos, en caso de que no correspondan a un objeto válido devolverá una excepción.

Instrucciones DML - **INSERT**

```
INSERT INTO alumno (al_numcta, al_nombre, al_apellidoPat,
  al_apellidoMat, al_genero, al_fechaNac, entidad_federativa_id)
VALUES ('307492333', 'Patricia', 'Castillo', 'Morett', 'M', '1983-05-01', 1);

INSERT INTO alumno (al_numcta, al_nombre, al_apellidoPat,
  al_apellidoMat, al_genero, al_fechaNac, entidad_federativa_id)
VALUES ('377792334', 'Mariano', 'Castillo', 'Mora', 'H', current_date(), 2);

ALTER TABLE NombreTabla AUTO_INCREMENT = 26000;
```

Ejercicios

Inserte los siguientes datos en la tabla **curso**:

curso_id	sede_id	cur_nombre
1	1	MATEMATICAS
2	1	GEOGRAFIA
3	1	FISICA
1	2	QUIMICA
2	2	BIOLOGIA
3	2	TEATRO

Inserte los siguientes datos en la tabla **programa**:

programa_id	curso_id	sede_id	profesor_id	prog_anio
1	1	1	1	2012
2	2	1	2	2012
3	3	1	3	2012
4	1	2	4	2012
5	2	2	5	2012
6	3	2	6	2012

Inserte los siguientes datos en la tabla **historial**:

historial_id	programa_id	alumno_id	his_calificacion
1	1	2	10
2	2	2	9
3	3	2	8
4	4	4	9
5	5	4	8
6	6	4	7
7	1	3	10
8	2	3	10
9	3	3	10

Instrucciones DML - **DELETE**

DELETE FROM table
[USING usinglist]
[WHERE condition] ;

DELETE FROM alumno
WHERE al_numcta='307492334';

DELETE FROM alumno
USING alumno, entidad_federativa ef
WHERE ef.entidad_federativa_id = alumno.entidad_federativa_id AND
entFederativa_nombre ='AGUASCALIENTES';

Instrucciones DML - **UPDATE**

```
UPDATE table SET column = { expression | DEFAULT } [, ...]
  [ FROM fromlist ]
  [ WHERE condition ]
```

```
UPDATE alumno SET al_numcta = '098987679', al_nombre ='Horacio',
  al_apellidoPat= concat('Del Bosque y ', al_apellidoPat)
WHERE alumno_id = 3;
```

```
UPDATE programa SET profesor_id=NULL
FROM profesor p
WHERE p.profesor_id =programa.profesor_id AND p.profesor_id = 1;
```

Subconsultas

Una subconsulta es un comando SELECT dentro de otro comando.

Subconsultas

- SELECT *

  FROM alumno

  WHERE alumno_id IN (SELECT alumno_id FROM alumno WHERE entidad_federativa_id=2);

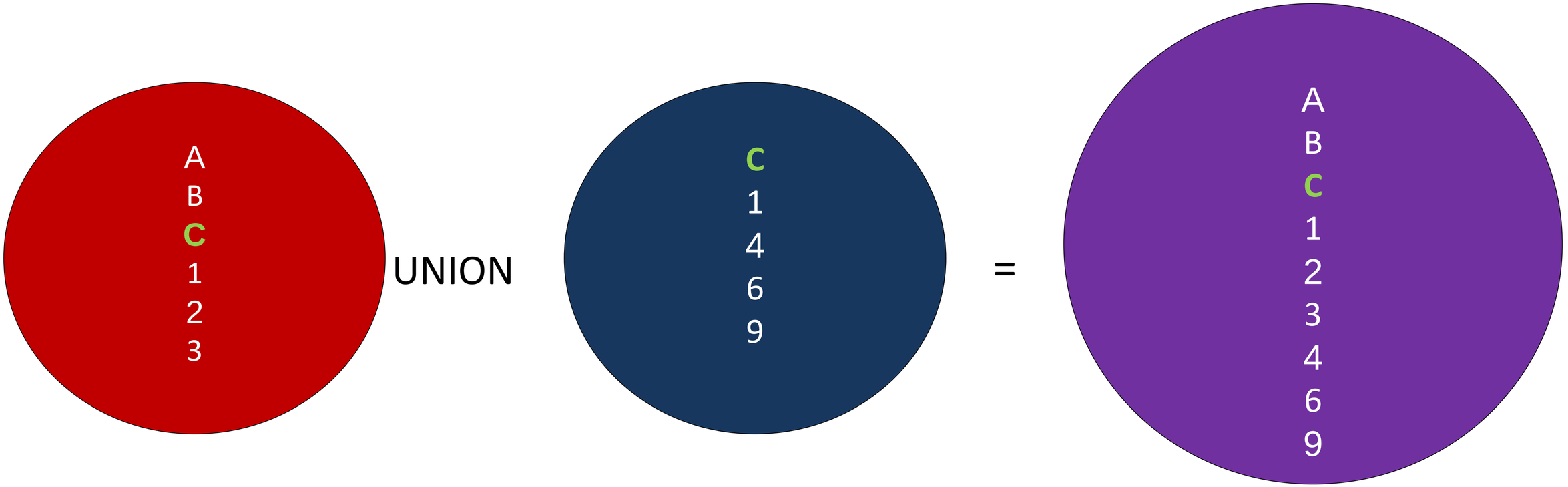
- SELECT *

  FROM alumno WHERE al_fechaNac = (SELECT min(al_fechaNac) FROM alumno);

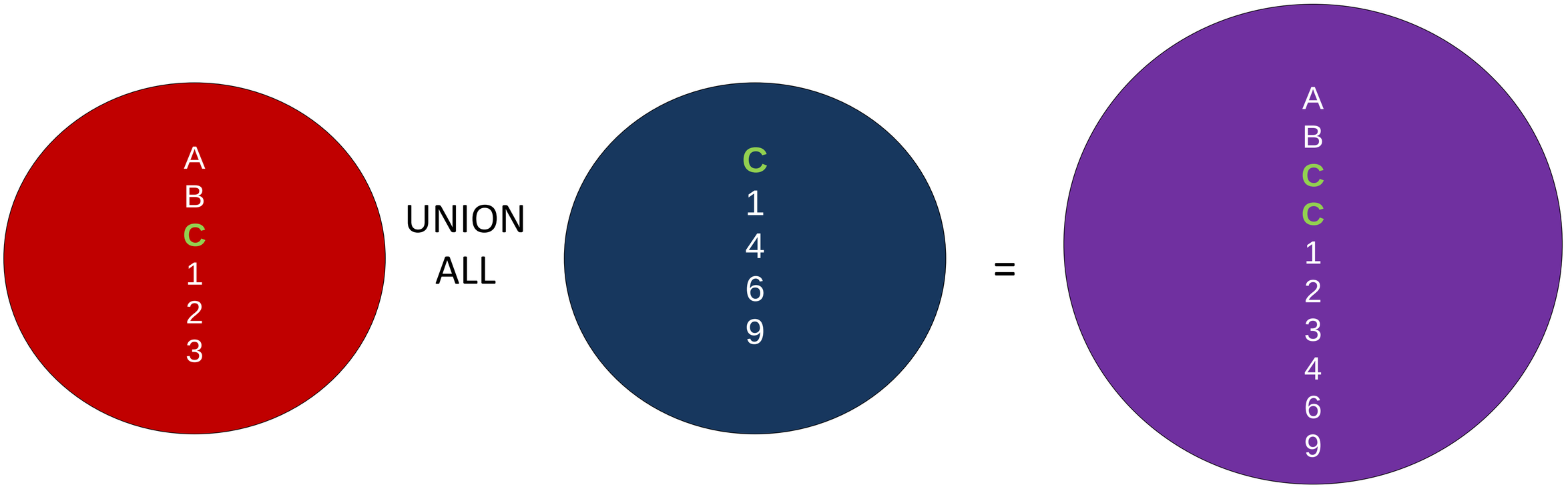
- SELECT *

  FROM (SELECT al_nombre, al_apellidoPat, al_apellidoMat FROM alumno WHERE al_genero= 'M') as mujer;

UNION



UNION



UNION

SELECT ... UNION [ALL]
SELECT ... [UNION [ALL] SELECT ...]

Respaldo y Restauración

Cuando se trabaja con base de datos, uno de las cosas más importantes de hacer es respaldar los datos en caso de que ocurra cualquier situación que dañe nuestro servidor de base de datos. De nada nos serviría hacer esos respaldos si no los podemos volver a cargar (restaurar) cuando lo necesitemos.

Respaldo

pg_dump [opciones] [nombre_base]

- **-a o –data-only:** Hace un volcado solo de los datos y no del esquema.
- **-c o –clean:** Crea instrucciones para eliminar los elementos antes de crearlos. Es útil para evitar los errores del tipo ‘la relacion nombre_relación ya existe’ a la hora de restaurar el respaldo.
- **-C o –create:** Escribe las instrucciones para la creación de la base de datos dentro del script del respaldo.
- **-f <archivo> o –file=<archivo>:** Escriba la salida (el volcado) en el archivo especificado. En caso de que no se utilice esta opción, el volcado se hace a la salida estándar.
- **-F <formato_de_archivo> o –format=<formato_de_archivo>:** Permite especificar el formato de la salida del dump. El formato de salida puede ser:
  - **p o plain: Texto plano.**
  - **c o custom:** Formato de salida customizable. Este tipo de salida siempre se realiza compreso por defecto.
  - **d o directory:** Formato de directorio con un archivo por cada tabla en formato de pg_restore.
  - **t o tar:** Crea la salida en formato tar.
- **-n <nombre_esquema> o –schema=<nombre_esquema>**: Realiza el dumpúnicamente del esquema (y todos los objetos que contengan) que concuerde con el<nombre_esquema>. Si no se especifica, se hará un dump de todos los esquemas que no sean del sistema y que estén en la base destino. Si se quiere incluir en el dump más de un esquema se pueden poner multiples -n <nombre_esquema> como sean necesarios.
- **-N <nombre_esquema> o –exclude-schema=<nombre_esquema>:** Omite los esquemas que concuerden con <nombre_esquema> del dump a realizarse. Se pueden incluir tantos -N como sean necesarios.

Respaldo

pg_dump [opciones] [nombre_base]

- **-s o –schema-only:** Hace un volcado únicamente del esquema, no de los datos.
- **-t <nombre_tabla> o –table=<nombre_tabla>:** Hace un volcado solo de las tablas que se especifiquen. Se pueden utilizar -t <nombre_tabla> tantas veces como se necesite. Se debe tener en cuenta que pg_dump no hace un seguimiento de las tablas de las que pueda depender la tabla que se desee volcar con el dump, así que hay que tener cuidado de incluirlas todas las que sean necesarias (que tengan relación con llaves primarias o foráneas) para garantizar que se puede hacer la restauración de los datos exitosamente.
- **-T <nombre_tabla> o — exclude-table: <nombre_tabla>:** Excluye del dumplas tablas listadas. Esta opción puede ser utilizada más de una vez.
- **--column-inserts o --attribute-inserts:** Utiliza inserts en lugar de copy en las instrucciones de SQL.
- **-p <port> --port = <puerto>:** Especifica el puerto TCP o local en el cual el servidor está escuchando.
- **-U <nombre_de_usuario> o --username=username:** Especifica el nombre de usuario con el que se hará la conexión a la base de datos que se desea respaldar.
- **-W o --password:** Forza a pg_dump a pedir el password de la cuenta.

https://www.postgresql.org/docs/current/static/app-pgdump.html

Respaldo

```
$ pg_dump mydb > db.sql

$ psql -d newdb -f db.sql

$ pg_dump -Fc mydb > db.dump

$ pg_dump -Fd mydb -f dumpdir

$ pg_dump -T 'ts_*' mydb > db.sql

$ pg_dump –U user -O -c -x mydb > db.sql
```

Restauración

pg_restore [opciones] [archivo_a_restaurar]

- **-a o –data-only:** Restaura solo los datos, no el esquema.
- **-c o –clean:** Elimina los objetos antes de volverlos a crear.
- **-C o –create:** Crea la base de datos especificada con la opción -d, sin embargo, los datos son restaurados a la base de datos que aparece en el script.
- **-j <numero> o –jobs=<numero>:** Realiza la restauración de los datos utilizando<numero> hilos o procesos (dependiendo del sistema operativo). Cada *jobs* es un proceso o hilo que utiliza una conexión separada. La utilización de esta opción, permite poder restaurar un *dump* en una forma más rápida, sin embargo, utiliza más recursos en el servidor. La velocidad de la restauración, por tanto, depende en gran medida de la configuración del hardware del servidor.
- **-n <nombre_esquema> o –schema=<nombre_esquema>:** Realiza la restauración únicamente del esquema llamado <nombre_esquema>.
- **-s o –schema-only:** Restaura solo el esquema, no los datos.
- **-t <nombre_tabla> o –table=<nombre_tabla>:** Restaura únicamente la tabla con el nombre <nombre_tabla>. <nombre_tabla> puede ser una expresión regular.
- **-U <nombre_usuario> o –username=<nombre_usuario>:** Nombre de usuario con el que se desea hacer la conexión.
- **-W o --password:** Forza a pg_restore a pedir el password de la cuenta.

https://www.postgresql.org/docs/current/static/app-pgrestore.html

Restauración

```
$ dropdb mydb
$ pg_restore -C -d postgres db.dump

$ createdb -T template0 newdb
$ pg_restore -d newdb db.dump

$ pg_restore -l db.dump > db.list

$psql –U user –W mydb < db.sql
```

Transacciones

Hay operaciones en los sistemas de bases de datos (DBMS) que no pueden expresarse como una única operación SQL sino como el resultado de un conjunto de dos o más operaciones SQL, cuyo éxito depende de que cada una de esas operaciones se ejecute correctamente ya que si una de ellas falla se considera que toda la operación fallo.

El control de transacciones es una característica fundamental de cualquier DBMS (como PostgreSQL,MS SQL Server ú Oracle) esto permite agrupar un conjunto de operaciones o enunciados SQL en una misma unidad de trabajo discreta , cuyo resultado no puede ser divisible ya que solo se considera el total de operaciones completadas, si hay una ejecución parcial el DBMS se encarga de revertir esos cambios para dejar la información consistente.

Transacciones

- **Atomicity(Atomicidad):** Una transacción es una unidad atómica o se ejecutan las operaciones múltiples por completo o no se ejecuta absolutamente nada, cualquier cambio parcial es revertido para asegurar la consistencia en la base de datos.

- **Consistency (Consistencia):** Cuando finaliza una transacción debe dejar todos los datos sin ningún tipo de inconsistencia, por lo que todas las reglas de integridad deben ser aplicadas a todos los cambios realizados por la transacción, o sea todas las estructuras de datos internas deben de estar en un estado consistente.

- **Isolation (Aislamiento o independencia):** Esto significa que los cambios de cada transacción son independientes de los cambios de otras transacciones que se ejecuten en ese instante, o sea que los datos afectados de una transacción no están disponibles para otras transacciones sino hasta que la transacción que los ocupa finalice por completo.

- **Durability (Permanencia):** Después de que las transacciones hayan terminado, todos los cambios realizados son permanentes en la base de datos incluso si después hay una caída del DBMS.

Transacciones

- **BEGIN:** Empieza la transacción
- **SAVEPOINT [name]:** Le dice al DBMS la localización de un punto de retorno en la transacción si una parte de la transacción es cancelada. El DBMS guarda el estado de la transacción hasta este punto.
- **COMMIT:** Todos los cambios realizados por las transacciones deben ser permanentes y accesibles a las demás operaciones del DBMS.
- **ROLLBACK [savepoint]:** Aborta la actual transacción todos los cambios realizados deben ser revertidos.

Transacciones

```
BEGIN;
        UPDATE table set column = 1;
        DELETE from table;
ROLLBACK;

BEGIN;
        UPDATE table set column = column + 1;
COMMIT;

BEGIN;
        INSERT INTO table1 VALUES (1);
        SAVEPOINT my_savepoint;
        INSERT INTO table1 VALUES (2);
        ROLLBACK TO SAVEPOINT my_savepoint;
        INSERT INTO table1 VALUES (3);
COMMIT;
```

Funciones

- También se conocen como procedimientos almacenados.
- Permiten llevar a cabo muchas operaciones que normalmente tomarían varias consultas y operaciones.
- Permiten la reutilización y pueden funcionar como capa intermedia entre la aplicación y la base de datos.
- Las funciones se pueden crear con el lenguaje de tu elección como son SQL, PL/SQL, C, Python, Java etc.

Sintaxis

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
[ WITH ( attribute [, ...] ) ]
```

Parámetros

- **name:** Nombre de la función.
- **argmode:** El modo de un argumento: IN, OUT, INOUT o variadic. El valor predeterminado es IN. Los argumentos OUT y INOUT no se pueden utilizar junto con la notación RETURNS TABLE.
- **argname:** El nombre de un argumento. Algunos idiomas (incluyendo PL / pgSQL, no SQL) le permiten usar el nombre en el cuerpo de la función. En cualquier caso, el nombre de la columna en el tipo de fila resultado. (Si se omite el nombre para un argumento de salida, el sistema escogerá un nombre de columna por defecto.)
- **argtype:** El tipo de dato (s) de argumentos de la función (opcionalmente calificado por el esquema), si los hubiere. Los tipos de argumentos pueden ser tipos de base, compuesto, o de dominio, o pueden hacer referencia al tipo de una columna de tabla.
- **default_expr:** Una expresión para ser utilizada como valor por defecto si no se especifica el parámetro. La expresión tiene que ser coercible para el tipo de argumento del parámetro. Sólo entrada (incluyendo INOUT) parámetros puede tener un valor por defecto. Todos los siguientes parámetros de entrada deben tener valores predeterminados también.
- **rettype:** El tipo de dato de retorno. El tipo de retorno puede ser una base, compuesto, o el tipo de dominio, o puede hacer referencia al tipo de una columna de tabla. Si no se supone que la función devuelva un valor, especifique vacío como el tipo de retorno. Cuando existen parámetros OUT o INOUT, la cláusula RETURNS se puede omitir. Si está presente, debe estar de acuerdo con el tipo de resultado que implican los parámetros de salida: RECORD si hay varios parámetros de salida, o del mismo tipo que el parámetro de salida única.
- **column_name**: El nombre de una columna de salida en la sintaxis RETURNS TABLE. Se trata efectivamente de otra forma de declarar un parámetro denominado OUT.

Parámetros

- **column_type:** El tipo de datos de una columna de salida en la sintaxis RETURNS TABLE.
- **lang_name:** Lenguaje que la función ocupa. Puede ser SQL, C, Interno. El nombre puede ser encerrado en comillas simples.
- **WINDOW:** Indica que la función es una función de ventana en lugar de una función normal. Esta es actualmente sólo es útil para funciones escritas en C.
- **IMMUTABLE, STABLE, VOLATILE:** Estos atributos informan al optimizador de consultas sobre el comportamiento de la función. Si esta es omitida, volátil es la opción por defecto.
    - **INMUTABLE**: indica que la función no puede modificar la base de datos y siempre devuelve el mismo resultado cuando se les da los mismos valores de los argumentos
    - **STABLE:** indica que la función no puede modificar la base de datos, y que dentro de una única tabla de exploración de manera consistente devolverá el mismo resultado para los mismos valores de los argumentos, sino que su resultado podría cambiar a través de sentencias SQL. Esta es la selección apropiada de funciones cuyos resultados dependerá de las búsquedas de bases de datos, variables de parámetros (como la zona horaria actual, etc. No es apropiado para triggers AFTER)
    - **VOLATILE:** indica que el valor de la función puede cambiar, incluso dentro de un mismo recorrido de tabla, por lo que no puede ser optimizada. Relativamente pocas funciones de base de datos son volátiles en este sentido; algunos ejemplos son random (), currval (), timeofday (). Pro hay que tener en cuenta que cualquier función que tiene efectos secundarios debe clasificarse volátil, incluso si su resultado es bastante predecible; un ejemplo es setval ().

Parámetros

- **CALLED ON NULL INPUT, RETURNS NULL ON NULL INPUT, STRICT**
  - **CALLED ON NULL INPUT: (valor determinado)** Indica que la función se llama normalmente cuando algunos de sus argumentos son nulos. Es responsabilidad del autor comprobar si hay valores nulos y en caso necesario responder de manera apropiada.
  - **RETURNS NULL ON NULL INPUT o STRICT:** La función siempre devuelve un valor nulo cuando alguno de sus argumentos sean nulos. Si se especifica este parámetro, la función no se ejecuta y regresa nulo automáticamente.
- **[EXTERNAL] SECURITY INVOKER, [EXTERNAL] SECURITY DEFINER:**
  - **SECURITY INVOKER:** Especifica que la función va a ser ejecutado con los privilegios del usuario que lo esta llamando.
  - **SECURITY DEFINER** Especifica que la función va a ser ejecutado con los privilegios del usuario que lo creó.
- **execution_cost:** Un número positivo que da el costo estimado de ejecución para la función, en unidades de **cpu_operator_cost.** Si no se especifica el coste este será 1 para el lenguaje C y 100 unidades para los demás idiomas.
- **result_rows:** el número estimado de filas que el planificador debe esperar que la función retorne. El valor por default son 1000 filas.
- **configuration_parameter value:** La clausula SET establece el parámetro especificado. SET FROM CURRENT guarda el valor actual de la sesión.

Sobrecarga

- PostgreSQL permite la sobrecarga de funciones; es decir, el mismo nombre se puede utilizar para varias funciones diferentes, con la diferencia que tengan tipos de argumentos de entrada distintos.
- Sin embargo, los nombres de todas las funciones de C deben ser diferentes.

```
CREATE FUNCTION foo(int) ...
CREATE FUNCTION foo(int, out text) ...
```

```
CREATE FUNCTION foo(int) ...
CREATE FUNCTION foo(int, int default 42) ...
```

Funciones

```
CREATE FUNCTION clean_emp() RETURNS void AS ‘
        DELETE FROM emp WHERE salary < 0;
' LANGUAGE SQL;

CREATE FUNCTION one() RETURNS integer AS $$
    SELECT 1 AS result;
$$ LANGUAGE SQL;
```

Funciones

```
CREATE FUNCTION add_em(integer, integer) RETURNS integer AS $$
    SELECT $1 + $2;
$$ LANGUAGE SQL;

CREATE FUNCTION add_em(x integer, y integer) RETURNS integer AS $$
    SELECT x + y;
$$ LANGUAGE SQL;
```

Funciones PL/PgSQL

Es un lenguaje procedural para PostgreSQL

- Se puede utilizar para crear funciones y procedimientos de activación.
- Agrega estructuras de control al lenguaje SQL.
- Puede realizar cálculos complejos.
- Hereda todos los tipos definidos por el usuario, funciones y operadores.
- Se puede definir para ser de confianza para el servidor.
- Es fácil de usar.

Con PL / pgSQL puede una serie de consultas dentro de la base de datos, teniendo así la potencia de un lenguaje procesal y la facilidad de uso de SQL, pero con un considerable ahorro de comunicación entre el cliente / servidor.

- idas y vueltas adicionales entre el cliente y el servidor se eliminan
- El cliente no necesita ser transferido entre el servidor y el cliente.
- Varias rondas de análisis de la consulta se pueden evitar

Esto puede resultar en un aumento de rendimiento considerable en comparación con una aplicación que no utiliza las funciones almacenadas.

Funciones PL/PgSQL

```
CREATE FUNCTION somefunc() RETURNS integer AS $$
        DECLARE
            quantity integer := 30;
        BEGIN
            RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 30
            quantity := 50;
            RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 50
            RETURN quantity;
        END;
$$ LANGUAGE plpgsql;
```

https://www.postgresql.org/docs/9.5/static/plpgsql.html

Funciones PL/PgSQL

```
CREATE FUNCTION somefunc() RETURNS integer AS $$
        << padre >>
        DECLARE
            quantity integer := 30;
        BEGIN
            RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 30
            quantity := 50;
        -- Create a subblock
        DECLARE
                quantity integer := 80;
            BEGIN
                RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 80
                RAISE NOTICE 'Outer quantity here is %', padre.quantity;  -- Prints 50
            END;

            RAISE NOTICE 'Quantity here is %', quantity;  -- Prints 50

            RETURN quantity;
        END;
$$ LANGUAGE plpgsql;
```

Funciones PL/PgSQL

```
CREATE FUNCTION sales_tax(subtotal real) RETURNS real AS $$
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION sales_tax(real) RETURNS real AS $$
DECLARE
    subtotal ALIAS FOR $1;
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

Funciones PL/PgSQL

```
CREATE FUNCTION sales_tax(real) RETURNS real AS $$
DECLARE
    hijo record;
BEGIN
        FOR hijo IN SELECT * FROM tabla LOOP
        END LOOP;
END;
$$
LANGUAGE 'plpgsql'
VOLATILE
RETURNS NULL ON NULL INPUT
SECURITY INVOKER
COST 100;
```

Triggers

Acción definida en una tabla de nuestra base de datos y ejecutada automáticamente por una función programada por nosotros. Esta acción se activará, según la definamos, cuando realicemos un INSERT, un UPDATE ó un DELETE.
Un disparador se puede definir de las siguientes maneras:

- Para que ocurra ANTES de cualquier INSERT,UPDATE ó DELETE
- Para que ocurra DESPUES de cualquier INSERT,UPDATE ó DELETE
- Para que se ejecute una sola vez por comando SQL (statement-level trigger)
- Para que se ejecute por cada línea afectada por un comando SQL (row-level trigger)

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tables and foreign tables	Tables, views, and foreign tables
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/DELETE	Tables and foreign tables	Tables, views, and foreign tables
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	—
	TRUNCATE	—	—

Triggers

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [
OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY
DEFERRED ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE PROCEDURE function_name ( arguments )

where event can be one of:

    INSERT
    UPDATE [ OF column_name [, ... ] ]
    DELETE
    TRUNCATE
```

Parámetros

- **NAME:** El nombre se le asigna el nuevo gatillo. Esto debe ser distinto del nombre de cualquier otro desencadenante de la misma mesa. El nombre no puede ser calificado por el esquema - el gatillo hereda el esquema de su tabla. Para un disparador restricción, este es también el nombre que se utilizará cuando se modifica el comportamiento del disparador en el SET CONSTRAINTS
- **BEFORE, AFTER ó INSTEAD OF:** Determina si la función se llama antes, después o en lugar del evento.
  - Un CONSTRAINT trigger sólo se puede especificar como DESPUÉS.
- **EVENT:** Evento que dispara el trigger puede ser INSERT, UPDATE, DELETE o TRUNCATE.
  - Para eventos múltiples se puede especificar mediante OR.
  - Solo para el evento UPDATE se puede especificar el nombre de los campos UPDATE OF column_name1 [, column_name2 ...]
- **TABLE_NAME:** nombre de la tabla, vista o tabla externa.
- **REFERENCED_TABLE_NAME:** el nombre de la otra tabla referenciada por el constraint. Esta opción es usada por las llaves foráneas y no es recomendable para uso general. Esta opción solo puede ser especificada para constraint triggers.
- **DEFERRABLE, NOT DEFERRABLE, INITIALLY IMMEDIATE ó INITIALLY DEFERRED:** El tiempo que se ejecutara el disparador. Esta opcion solo puede ser especificada para constraint triggers.
- **FOR EACH ROW ó FOR EACH STATEMENT:** Especifica si el procedimiento de activación debe ser encendido una vez para cada fila afectada o sólo una vez por cada instrucción SQL. Si no se especifica ninguno ó FOR EACH STATEMENT es el valor predeterminado. La opción de FOR EACH ROW solo puede ser especificada para constraint triggers.

Parámetros

- **CONDITION:** Una expresión booleana que determina si la función de disparo en realidad se ejecutará. En FOR EACH ROW la condición puede referirse a valores nuevos o viejos (OLD.column_name o NEW.column_name). En INSERT triggers no se puede referir a OLD y en DELETE triggers no se puede referir a NEW.
- **FUNCTION_NAME:** Nombre de la función a ejecutarse declarada por el usuario sin argumentos y el tipo de dato de regreso es trigger.
- **ARGUMENTOS:** lista separada por comas de los argumentos . Los argumentos deben ser constantes de cadena o numéricas , nombre simples

Variables Especiales en PL/pgSQL

- **NEW:** Tipo de dato RECORD; Variable que contiene la nueva fila de la tabla para las operaciones INSERT/UPDATE en disparadores del tipo row-level. Esta variable es NULL en disparadores del tipo statement-level.
- **OLD:** Tipo de dato RECORD; Variable que contiene la antigua fila de la tabla para las operaciones UPDATE/DELETE en disparadores del tipo row-level. Esta variable es NULL en disparadores del tipo statement-level.
- **TG_NAME:** Tipo de dato name; variable que contiene el nombre del disparador que está usando la función actualmente.
- **TG_WHEN:** Tipo de dato text; una cadena de texto con el valor BEFORE o AFTER dependiendo de como el disparador que está usando la función actualmente ha sido definido
- **TG_LEVEL:** Tipo de dato text; una cadena de texto con el valor ROW o STATEMENT dependiendo de como el disparador que está usando la función actualmente ha sido definido
- **TG_OP:** Tipo de dato text; una cadena de texto con el valor INSERT, UPDATE o DELETE dependiendo de la operación que ha activado el disparador que está usando la función actualmente.
- **TG_RELID:** Tipo de dato oid; el identificador de objeto de la tabla que ha activado el disparador que está usando la función actualmente.

Variables Especiales en PL/pgSQL

- **TG_RELNAME:** Tipo de dato name; el nombre de la tabla que ha activado el disparador que está usando la función actualmente. Esta variable es obsoleta y puede desaparecer en el futuro. Usar TG_TABLE_NAME.
- **TG_TABLE_NAME:** Tipo de dato name; el nombre de la tabla que ha activado el disparador que está usando la función actualmente.
- **TG_TABLE_SCHEMA:** Tipo de dato name; el nombre de la schema de la tabla que ha activado el disparador que está usando la función actualmente.
- **TG_NARGS:** Tipo de dato integer; el número de argumentos dados al procedimiento en la sentencia CREATE TRIGGER.
- **TG_ARGV[]:** Tipo de dato text array; los argumentos de la sentencia CREATE TRIGGER. El índice empieza a contar desde 0. Indices inválidos (menores que 0 ó mayores/iguales que tg_nargs) resultan en valores nulos.

Triggers

```
CREATE TABLE numeros(
  numero bigint NOT NULL,
  cuadrado bigint,
  cubo bigint,
  raiz2 real,
  raiz3 real,
  PRIMARY KEY (numero)
);
```

```
CREATE OR REPLACE FUNCTION proteger_datos() RETURNS
TRIGGER AS $proteger_datos$
  DECLARE
  BEGIN
   RETURN NULL;
  END;
$proteger_datos$
LANGUAGE plpgsql;

CREATE TRIGGER proteger_datos BEFORE DELETE
    ON numeros FOR EACH ROW
    EXECUTE PROCEDURE proteger_datos();
```

Triggers

```
CREATE OR REPLACE FUNCTION rellenar_datos() RETURNS TRIGGER AS $rellenar_datos$
  DECLARE
  BEGIN
        NEW.cuadrado := power(NEW.numero,2);
        NEW.cubo := power(NEW.numero,3);
        NEW.raiz2 := sqrt(NEW.numero);
        NEW.raiz3 := cbrt(NEW.numero);
        RETURN NEW;
  END;
$rellenar_datos$ LANGUAGE plpgsql;

CREATE TRIGGER rellenar_datos BEFORE INSERT OR UPDATE
    ON numeros FOR EACH ROW
    EXECUTE PROCEDURE rellenar_datos();

-- Borrar el trigger
DROP TRIGGER proteger_datos ON numeros;
```

Triggers

```
CREATE OR REPLACE FUNCTION proteger_y_rellenar_datos() RETURNS TRIGGER AS $proteger_y_rellenar_datos$
  DECLARE
  BEGIN
    IF (TG_OP = 'INSERT' OR TG_OP = 'UPDATE' ) THEN
       NEW.cuadrado := power(NEW.numero,2);
       NEW.cubo := power(NEW.numero,3);
       NEW.raiz2 := sqrt(NEW.numero);
       NEW.raiz3 := cbrt(NEW.numero);
       RETURN NEW;
    ELSEIF (TG_OP = 'DELETE') THEN
       RETURN NULL;
    END IF;
  END;
$proteger_y_rellenar_datos$ LANGUAGE plpgsql;
```

Full Text Search

- La búsqueda de texto completo (o simplemente Búsqueda de texto) proporciona la capacidad de identificar el lenguaje natural que contenga una consulta y opcionalmente ordenarlos por relevancia.
- Pre procesa los índices de los documentos y este se guarda para una mayor rapidez, esto incluye:
  - Convertir los documentos en tokens: identificar los diferentes tipos de tokens como son números, palabras, palabras complejas, direcciones de correo electrónico.
  - Convertir los tokens en lexemas. Los lexemas son cadenas parecidas a los tokens pero estas son normalizadas por ejemplo quitar las mayúsculas, quitar los sufijos, eliminar las stop words. para esto postgreSQL usa diccionarios.
  - Almacenar los documentos pre procesados para optimizar la consulta. Cada documento es representado por una matriz ordenada de lexemas y posiciones.

Documento

- Es la unidad mínima de búsqueda.
- En Postgres, un documento es un campo en una fila de una tabla, o quizá una concatenación de varios campos, de una misma tabla o de más de una con un join.
- Al combinar campos suele ser conveniente usar la función coalesce para convertir los valores NULL en ”, de otro modo si un campo es nulo, el documento entero será nulo, por la propiedad de que tiene el valor NULL de que genera NULL al participar en cualquier expresión.

```
SELECT campo_1 || ' ' || campo_2 || ' '
|| campo_n as documento
FROM <<tabla>>
WHERE <<condiciones>>;
```

```
SELECT coalesce(campo_1, ' ') ||
coalesce(campo_2, ' ')  ||
coalesce(campo_n, ' ')  as documento
FROM <<tabla>>
WHERE <<condiciones>>;
```

tsvector

- Es un tipo de datos de Postgres, que consiste en una lista de palabras extraídas de un documento. Las palabras están normalizadas, es decir, se eliminan las palabras “stopwords” (artículos, conjunciones, etc.) y los signos de puntuación, y el resto de palabras se reducen a su lexema básico. Finalmente se añade a cada palabra en qué posición o posiciones del documento aparece.

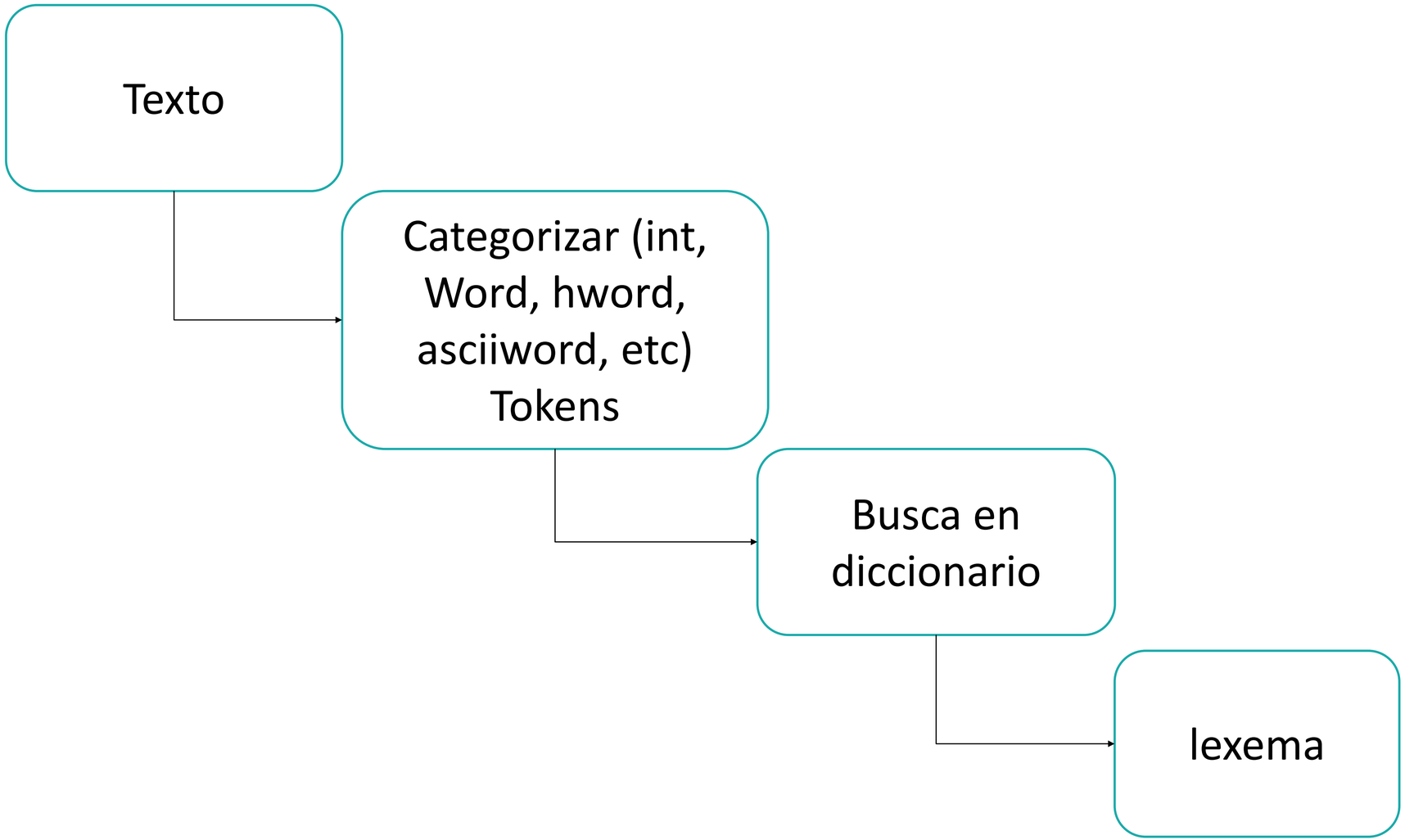
```
SELECT ‘A fat cat - sat on a mat and ate a fat rat'::tsvector;

tsvector
 ----------------------------------------------------
'-' 'A' 'a' 'and' 'ate' 'cat' 'fat' 'mat' 'on' 'rat' 'sat'
```

```
SELECT to_tsvector('english', ‘A fat cat - sat on a mat and ate a fat rat');

to_tsvector;
 ----------------------------------------------------
'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4
```

tsvector


Texto
Categorizar (int, Word, hword, asciiword, etc) Tokens
Busca en diccionario
lexema

tsquery

- Es un tipo de datos que nos brinda herramientas adicionales para consultar el vector de búsqueda de texto completo.
- Operadores & (and), | (or) y ! (not), y paréntesis ().
- También existe una función to_tsquery para convertir un string en un tsquery

```
SELECT ‘rat & cat'::tsquery;

tsquery
-------------------------
'rat' & 'cat'
```

```
SELECT to_tsquery('english', 'Fat & (Rat | Cat)');

to_tsquery
-------------------------
'fat' & ( 'rat' | 'cat' )
```

```
SELECT plainto_tsquery('english', 'The Fat Rats');

plainto_tsquery
----------------
'fat' & 'rat’
```

Operadores

- El FTS en PostgreSQL está basada en el operador @@, el cual retorna true si un tsvector (documento) coincide con una tsquery (consulta). No importa el orden en el que se pongan:

```
SELECT *
FROM <<tabla>>
WHERE tsvector @@
to_tsquery('keyword');
```

```
SELECT *
FROM <<tabla>>
WHERE tsvector @@
plainto_tsquery(‘cadena de texto');
```

Resaltar Resultados (Highlight)

- La siguiente cosa interesante que podemos hacer con los resultados de nuestro texto completo es poner de relieve las palabras relevantes.
- PostgreSQL nos ofrece la función: ts_headline().

```
SELECT title, ts_headline(‘english’, description,
query) AS description
FROM film, to_tsquery(‘english’,'Panorama‘)
query
WHERE fulltext @@ query
ORDER BY title ASC;
```

```
SELECT title, ts_headline(‘english’, description,
query, 'StartSel = <<<, StopSel = >>>') AS
description
FROM film, to_tsquery(‘english’,'Panorama‘)
query
WHERE fulltext @@ query
ORDER BY title ASC;
```

Peso a columnas

- La función setweight se puede utilizar para etiquetar las entradas de un tsvector con un peso dado, donde un peso es una de las letras A, B, C, o D.
- Los pesos por default son de la D a la A : {0.1, 0.2, 0.4, 1.0}

```
UPDATE <<tabla>> SET <<campo>> =
    setweight(to_tsvector(coalesce(<<campo_1>>,'')), 'A')    ||
    setweight(to_tsvector(coalesce(<<campo_2>>,'')), 'B');
```

Ranking de Resultados (Ranking)

- Hay dos funciones predefinidas (ts_rank y ts_rank_cd) que calculan la relevancia de un documento respecto de un tsquery, en función del número de veces que se encuentra cada término de búsqueda, la posición dentro del documento, etc.

```
SELECT <<campo>>,
ts_rank_cd(<<campo_2>>, query) AS rank
FROM <<tabla>>,
to_tsquery('english','<<palabra>>') query
WHERE <<campo_2>> @@ query;
```

Esta función calcula el ranking cover density, como se describe en el artículo "Relevance Ranking for One to Three Term Queries" en la revista "Information Processing and Management", 1999.

```
SELECT <<campo>>,
ts_rank((<<campo_2>>, query) AS rank
FROM <<table>>, to_tsquery('english',
'<<palabra>>') query
WHERE <<campo_2>> @@ query;
```

Basado en la frecuencia de aparicion de sus lexemas.

Índices

- GIST (Generalized Search Tree)
  - El GiST sí no es algo que es único a PostgreSQL, es un proyecto en sí mismo y su concepto se presenta en una biblioteca de C llamada libGist.
  - Siempre devolverá un no si no hay coincidencia o un tal vez si hay. Debido a este comportamiento PostgreSQL tiene que ir y comprobar manualmente todos los tal vez 's y verificar que si coincidan.
  - Las grandes ventajas son el hecho de que el índice se crea más rápido y la actualización de un índice es menos costosa.
- GIN (Generalized Inverted Index)
  - Es un índice determinista, devolverá verdadero si hay coincidencias.
  - No almacena pesos en los lexemas.
  - Entre mas grande se vuelva el índice mas lento será su actualización.

GIST o GIN?

- Los índices GIST usan un hash de longitud fija, que es bastante eficiente en espacio. Pero puede ocurrir que varios documentos generen el mismo hash, por lo que en una búsqueda aparecerán ambos cuando quizá sólo se esté buscando uno de ellos. Estos índices son buenos cuando los documentos no tienen muchas palabras (por debajo de 10.000). Es útil además definir una buena configuración que elimine todas las palabras posibles y normalice mucho.
- Los GIN en cambio, no tienen estas limitaciones, pero ocupan bastante más espacio, y son más lentos de actualizar, aunque son más rápidos de leer. La regla general suele ser usar GIN si los datos cambian poco o si hay muchas palabras distintas, y GIST para datos muy dinámicos pero sin demasiadas palabras, o si el espacio es muy importante.

```
CREATE INDEX index_name ON
<<tabla>> USING gin(<<campo>>);
```

```
CREATE INDEX index_name ON
<<tabla>> USING gist(<<campo>>);
```

Preparando el buscador

- Revisar si hay alguna configuración, diccionario que nos ayude:
  - \dF listado de configuraciones de búsqueda de texto.
  - \dFd diccionarios
  - \dfp analizadores
  - \dFt plantillas

- El diccionario Español que tiene por default PostgreSQL tiene problemas con los acentos en palabras agudas.

```
SELECT to_tsquery('spanish', 'habitacion');

SELECT to_tsquery('spanish', 'habitación');
```

Preparando el buscador

- Crear un lenguaje.
- Ocupar la extensión unaccent.
  - Es un diccionario de búsqueda de texto que elimina acentos (signos diacríticos) de los lexemas. Es un diccionario de filtrado, lo que significa su salida siempre pasa al siguiente diccionario (si existe), a diferencia del comportamiento normal de los diccionarios.

```
select * from pg_available_extensions; (extensiones disponibles)

select * from pg_extension; (extensiones habilitadas)
```

  - Creamos la extensión para usarla en nuestra base de datos

```
CREATE EXTENSION unaccent;
```

Preparando el buscador

- Creamos y configuramos la configuración de búsqueda

```
CREATE TEXT SEARCH CONFIGURATION <<nombre>>
( COPY = spanish );

ALTER TEXT SEARCH CONFIGURATION <<nombre>>
ALTER MAPPING FOR hword, hword_part, word
WITH unaccent, spanish_stem;
```

Ligas de interés

- https://www.postgresql.org/docs/9.6/static/index.html (General)
- https://www.postgresql.org/docs/9.6/static/datatype.html (Tipos de datos)
- https://www.postgresql.org/docs/9.6/static/plpgsql.html (PLSQL)
- https://www.postgresql.org/docs/9.6/static/triggers.html (Triggers)
- https://www.postgresql.org/docs/9.6/static/errcodes-appendix.html (Excepciones)
- https://www.postgresql.org/docs/9.6/static/textsearch.html (Full Text Search)


Gracias
DGTIC