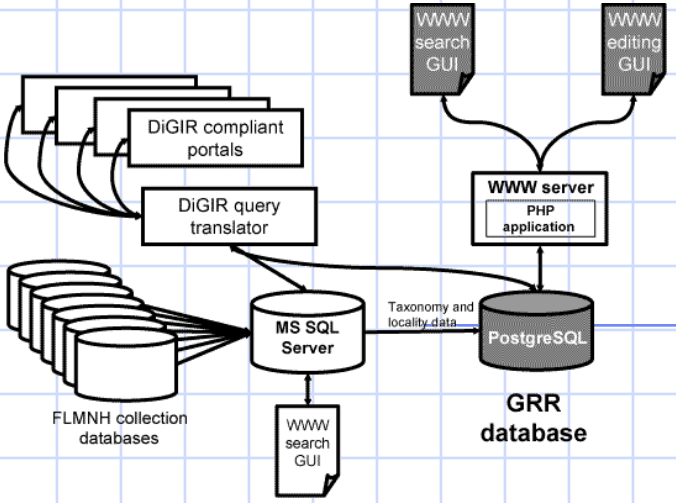
Universidad Tecnológica Nacional
Regional Paraná

Diseño y Adm.Base de Datos

Introducción a DDL
Vistas e Índices



Ing. Fernando Sato
fsatopna@gmail.com

Ultima Actualización: 26/03/2018

Resumen

- Vistas.
  - Concepto de Vistas.
  - DDL de vistas.
  - Vistas Verticlaes y Horizontales, opción WITH CHECK OPTION.
  - Temporales y Materializadas.
  - Vistas Actualizables y NO Actualizables.
  - Vistas del Sistema.
- Indices.
  - Conceptos, Estructura y Caracteristicas
  - DDL de Indices.
  - Indices Implicitos.

Estructura sintáctica de DDL

Para vistas se utiliza la misma estructura que para tablas con el tipo de objeto **view**.

DDL de vistas

- Create
- Drop
- Alter (postgreSQL tiene create or replace)

VISTAS

- Constituyen la segunda estructura fundamental de las BD.
- Implementan las *Relaciones Virtuales de Codd.*
- Son objetos que no contienen “datos” o “información”.
- Su estructura esta formada por una instrucción de selección (Select , Join, Union , Intersect , Excecpt, Subselect , etc).
- La información del metadato de una base, se “guarda “ en tablas especiales denominadas tablas del sistema, a estas se asocian vistas que presentan esta información agregada para mejor visualización.
- Las DMLs (insert / delete o /update) sobre vistas terminan en instrucciones DMLs sobre las tablas respectivas).

Utilización de las Vistas

- Por seguridad.
  - No deseamos dar acceso a toda la información de una tabla.

  Ver caso de estudio 1.
- Para *simplificar* el modelo.
  - Generación de una vista con Join, o con un agrupamiento o con una tabla derivada, por ejemplo para un reporte.

  Ver caso de estudio 2.
- Para *perfomance*.
  - Generación de una vista materializada que evite efectuar relaciones intermedias, ordenamiento y cambie el costo computacional a los insert, delete o update sobre las tablas base. *Ver caso de estudio 3.*

Estructura de las Vistas

- Una Vista esta definida por:
  - Un nombre (único para todo su esquema, incluyendo las tablas).
  - Una sentencia de selección (Select , Join, Union , Intersect , Excecpt, Subselect , etc).
  - Un Conjunto de nombres de columnas (renombrado). *Opcional*

Tabla base

```
create table productos
  ( codigo       integer not null primary key,
    nombre       varchar(50) unique,
    abreviatura  char(6),
    familia      dfamilia;

create domain dfamilia as varchar(12)
        check( value in ('herramienta','pintura','fertilizante'));
```

Vista

```
create view vherramientas
  as select * from productos where famila =
 'herramienta';
```

Concepto de la vista

Vista

```
create view vherramientas
  as select * from productos where famila =
'herramientas';
```

Conceptos

- **Vherramientas**: No tiene filas, las filas son o provienen de productos.
- **Vherramientas**: Se registra en el catalogo o diccionario del sistema.
- **Productos**: Si agrego una fila a productos también aparecerá en la vista, si cumple con la restricción.
- **Vherramientas**: Si agrego una fila en vherramientas la información se grabara en productos.

View CREATE

Sintaxis

CREATE OR REPLACE [TEMPORARY] VIEW *nombre_view*
[(view_col [, view_col])]
AS *<query>*

TEMPORARY (opcional), crea una vista temporal, idem concepto a tabla;

Nombre de las vistas:

- Hasta 64 caracteres (letras Nros y “_”)
- Deben comenzar con una letra
- No pueden ser palabras reservadas.

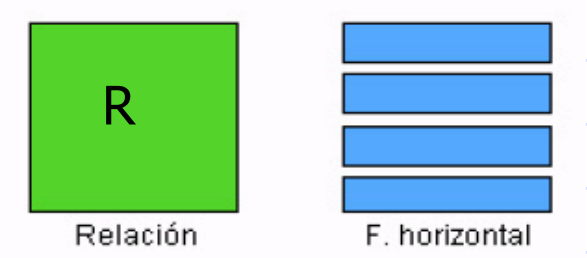
Vistas Verticales y Horizontales

Vistas Verticales: Involucran un subconjunto de columnas, es decir tienen la misma cardinalidad que la tabla asociada y grado < a la de la tabla. Sea F una vista sobre R.

Proyección $\Pi_{R\,i}\,(\,R\,)$

Vistas Horizontales: Involucran un subconjunto de filas, es decir tienen el mismo grado que las tabla asociada y canrdinalidad < a la de la tabla. Sea F una vista sobre R.

Restricción $\sigma_{Ci=''xxx''}\,(\,R\,)$



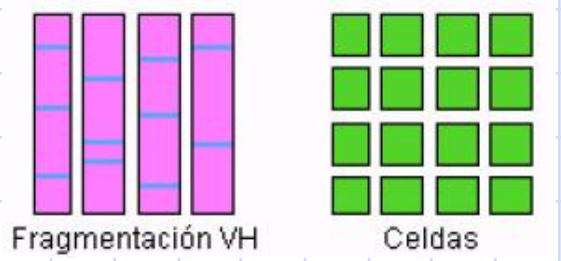
Where $ci = \text{“xxx”}$

Vistas Verticales y Horizontales

Vistas Mixta o Híbrida: Involucran un subconjunto de columnas y filas, es decir tienen < cardinalidad que la tabla asociada y grado < a la de la tabla. Sea F una vista sobre R.

Proyección y Restricción $\sigma_{Ci="xxx"} (\pi_{R\,i} (R))$

Where $ci = "xxx"$



Modificador With Check Option

Especificado por el standard e implementado en postgreSQL a partir de la versión 9.

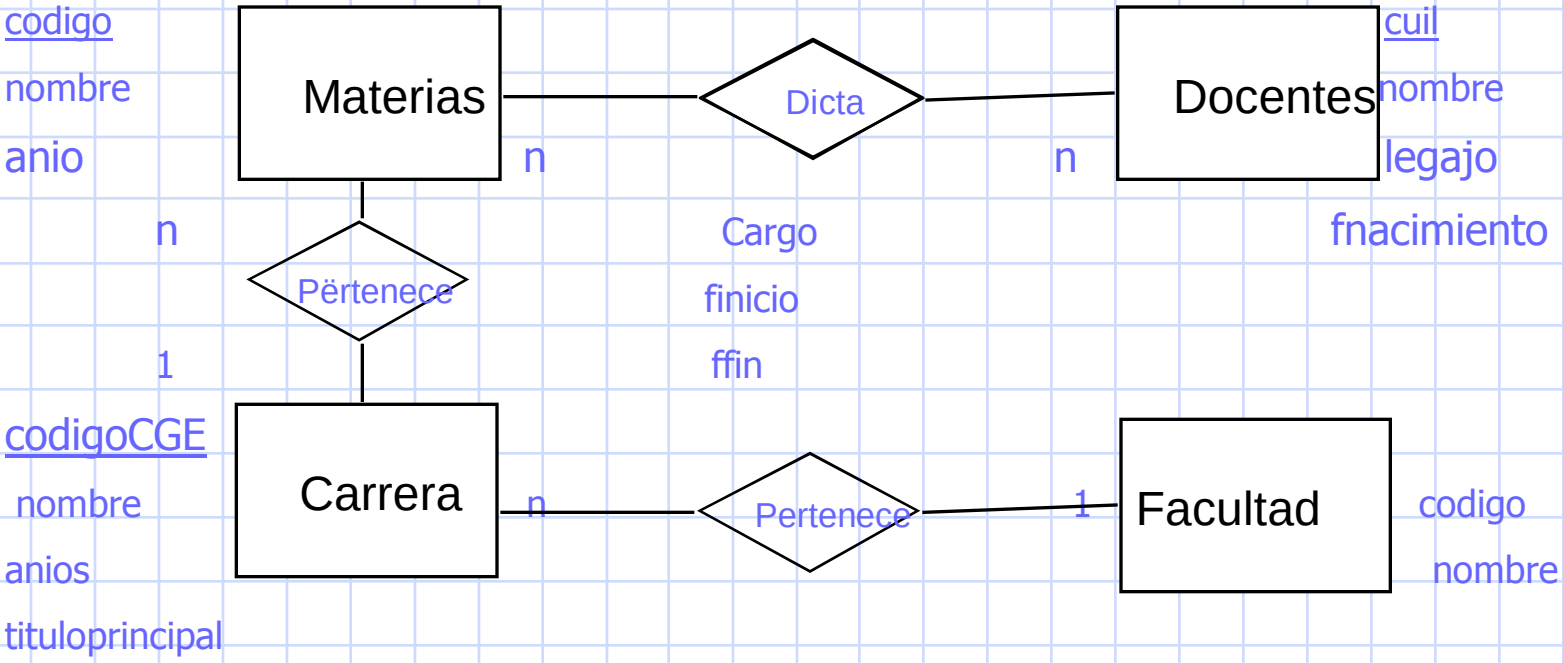
Cláusula asociada a create view para vistas horizontales (con where) de manera que no permita a usuarios autorizados a usar a trangredir su consistencia.

*Cuando esta presente "**with check option**"*

*No permite insertar o actualizar filas que no cumplan con la restricción (**Where**).*

Caso de Prueba

Unidades Académicas



1. Genere la tabla materias e inserte { (11,'B datos',2),(3,'Matematicas',1),(8,'M.Sist',2)}
2. Genere la vista VMATERIASNOMBRE solo con codigo y nombre como cod y nom.
3. Genere una vista temporal VMATERIAS2DOANIO del mismo grado que materias pero son con los elementos de año 2 con WITH CHECK OPTION.

View DROP

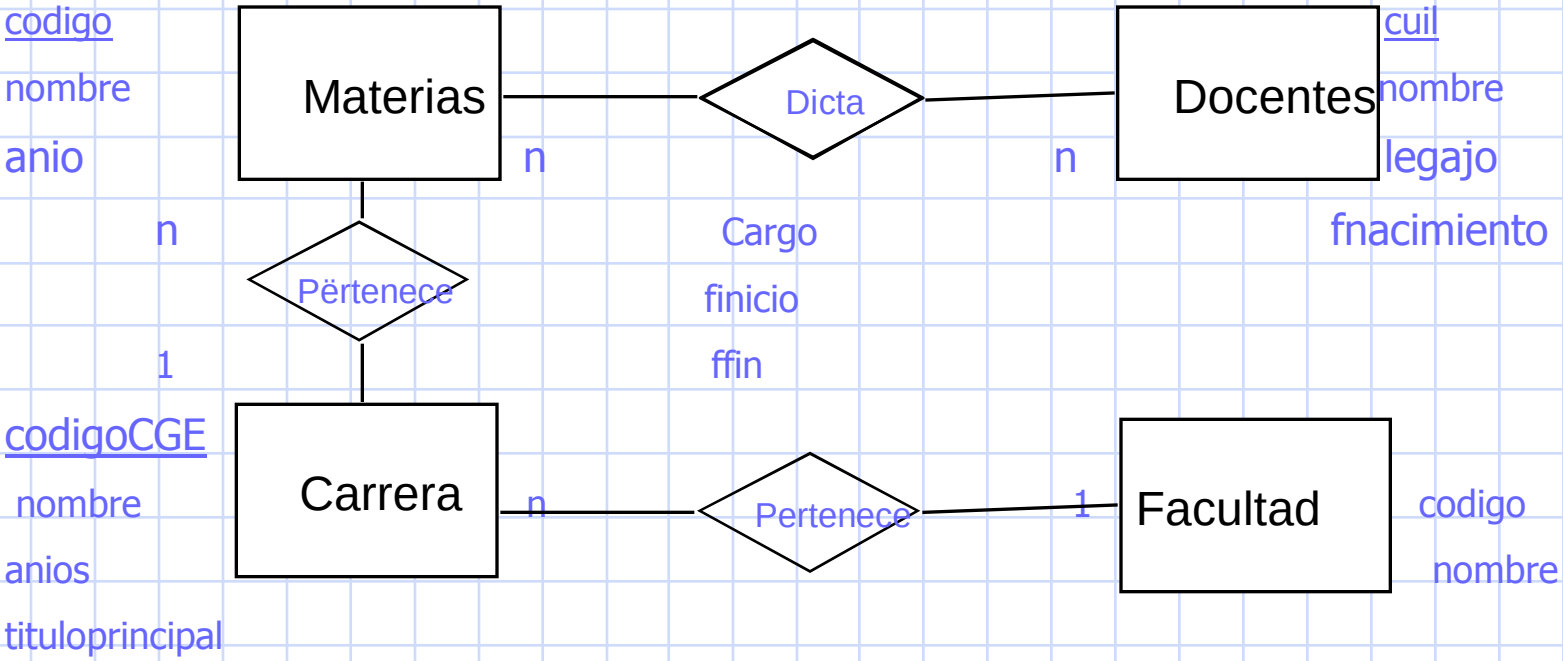
Sintaxis

DROP VIEW *nombre_view;*

Ej: ***DROP VIEW*** *vproductos2;*

Caso de Prueba

Unidades Académicas



4. Borre la vista VMATERIASNOMBRE.

Tipos de VISTAS s/Actualización

- Vistas actualizables. ~~(en postgreSQL con trigger INSTEAD OF)~~
- Vistas no actualizables.

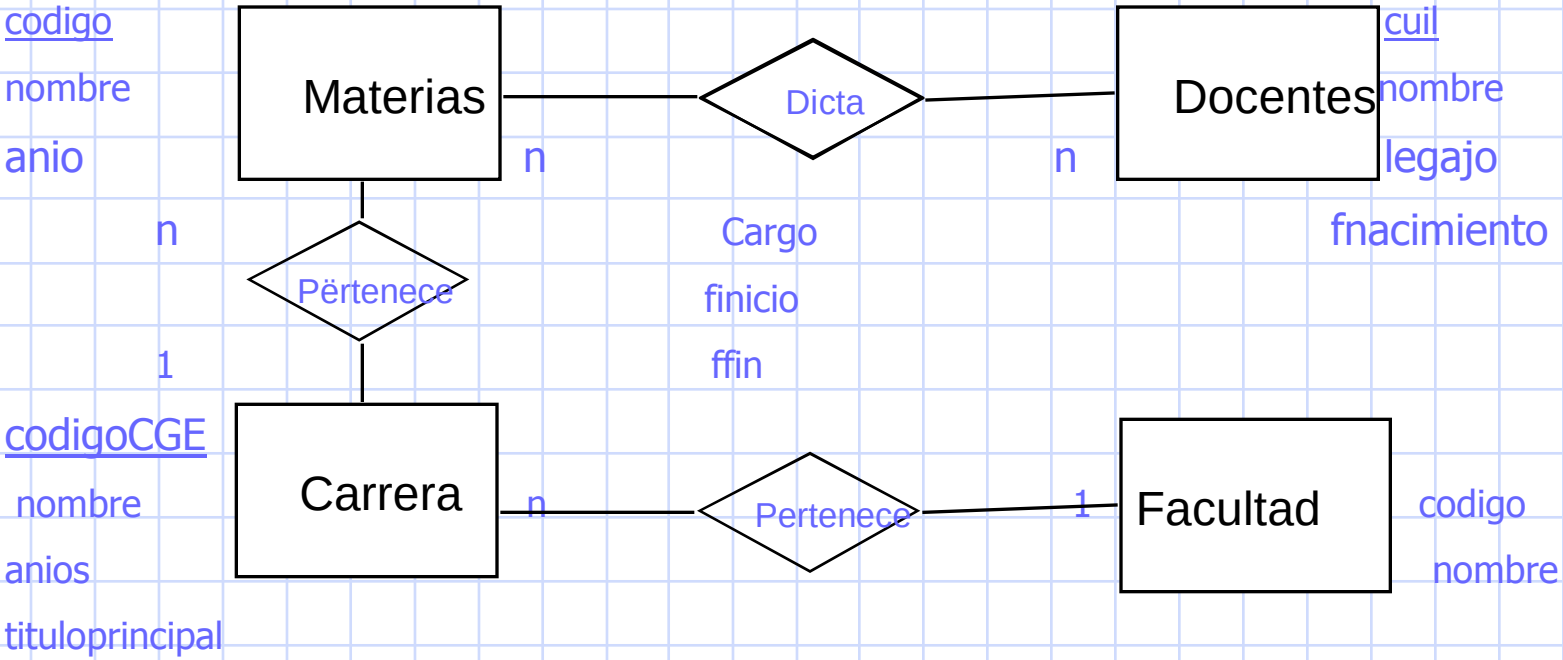
- Condiciones para ser actualizables:

  - Involucrar en el FROM una sola tabla (directa o indirectamente-contar aca JOIN también).
  - SELECT sin distinct y funciones de agregación.
  - El SELECT no debe tener tampoco GROUP BY.
  - Los campos no incluidos se insertan como NULL, no deben tener restricción NOT NULL.

Caso de Prueba

4. Agregue la filaVMATERIAS2DOANIO con (20,'PRACTICA PROFESIONAL',2).
5. Agregue la fila VMATERIA2DOANIO con (5,'PROGRAMACION 1',1). Analice que sucede.

Unidades Académicas



6. Genere una vista VMATANIOS con cantidad de materias por año (ANIO y CANTIDAD).
7. Inserte año 3 con 6 en cantidad sobre VMATANIOS. Analice.
8. Borre VMATANIOS. Analice.

Vistas definidas en función de otras

Idea

Se puede crear una vista de una vista, siempre que no sean recursivas.

Nota: *Los motores traducen las vistas de vistas a vistas de tablas.*

Caso de Prueba - Funcionamiento

¿Como trabajan las vistas en una consulta, como son sus planes de ejecución? Dada:

```
CREATE TABLE mueblesyutiles (
nroInventario     integer  primary key,
descripcion       varchar(60),
abreviatura       varchar(8),
tipo              varchar(1) check (tipo in ('M','H','O'))
);
```

1) Cree una vista Vherramientas para el tipo 'H', y una Vutiloficina para'O' .

2) Cree una vista Vherramientasr (por reducida) para Vherramientas proyectando nroInventario y abreviatura .

Caso de Prueba - Funcionamiento

- **¿Como trabajan las vistas en una consulta, como son sus planes de ejecución? Dada:**

1,2) Solución:

```
CREATE VIEW Vherramientas AS
     SELECT * FROM mueblesyutiles
     WHERE tipo = 'H';
```

```
CREATE VIEW Vherraredu AS
     SELECT nroInventario,
            abreviatura
      FROM vherramientas;
```

3) Veamos los planes de ejecución de la tabla y de ambas vistas.

```
SELECT *
FROM mueblesyutiles;
```

```
SELECT *
 FROM vherramientas
```

```
SELECT *
 FROM vherraredu
```

Los 3 query arman el mismo plan de ejecución.

QUERY PLAN text
Seq Scan on mueblesyutiles (cost=0.00..13.80 rows=380 width=184)

mueblesyutiles

Vistas Materializadas

Sintaxis

CREATE MATERIALIZED VIEW *nombre_view*
[(view_col [, view_col])]
AS *<select>*
[WITH CHECK OPTION]*;*

- Implementadas en postgreSQL a partir de 9.3,
  - No son 100 % dinámicas, se actualizan con REFRESH MATERIALIZED VIEW nombre_view.
  - se pueden implementar codificando nuestros propios triggers de mantenimiento de vistas. (Con instead of)

Vistas del SISTEMA

Todas las bases cuentan con herramientas para monitorear, conexiones, sesiones, memoria, uso de procesador y otros recursos.

La idea es que el monitoreo de la base se realice utilizando DML(select).

PostgreSQL: cuenta con las vistas pg. Ej: pg_settings, pg_stats, pg_stat_activity, pg_stat_database, etc.*

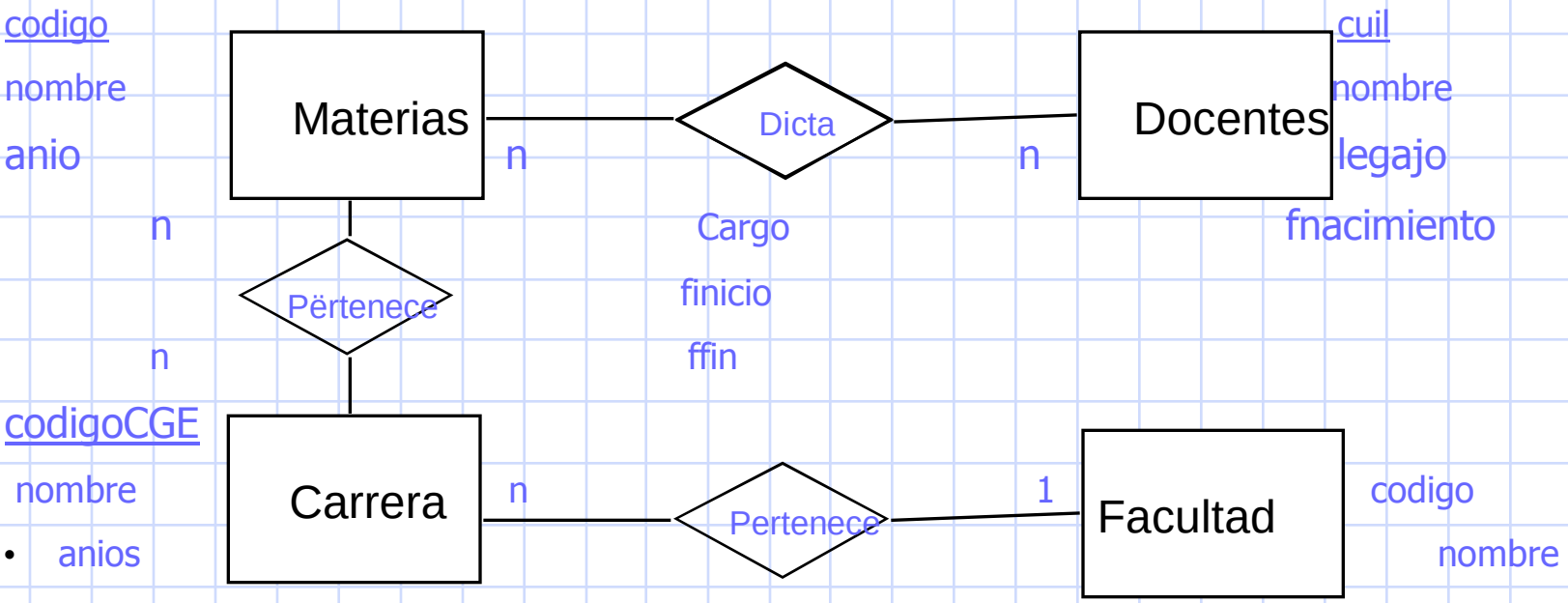
Firebird: cuenta con las vistas MON$.*

Ej: Mon$database; Mon$transactions

Oracle: cuenta con las vistas V$*. Ej: v$database;

Caso de Prueba

Unidades Académicas



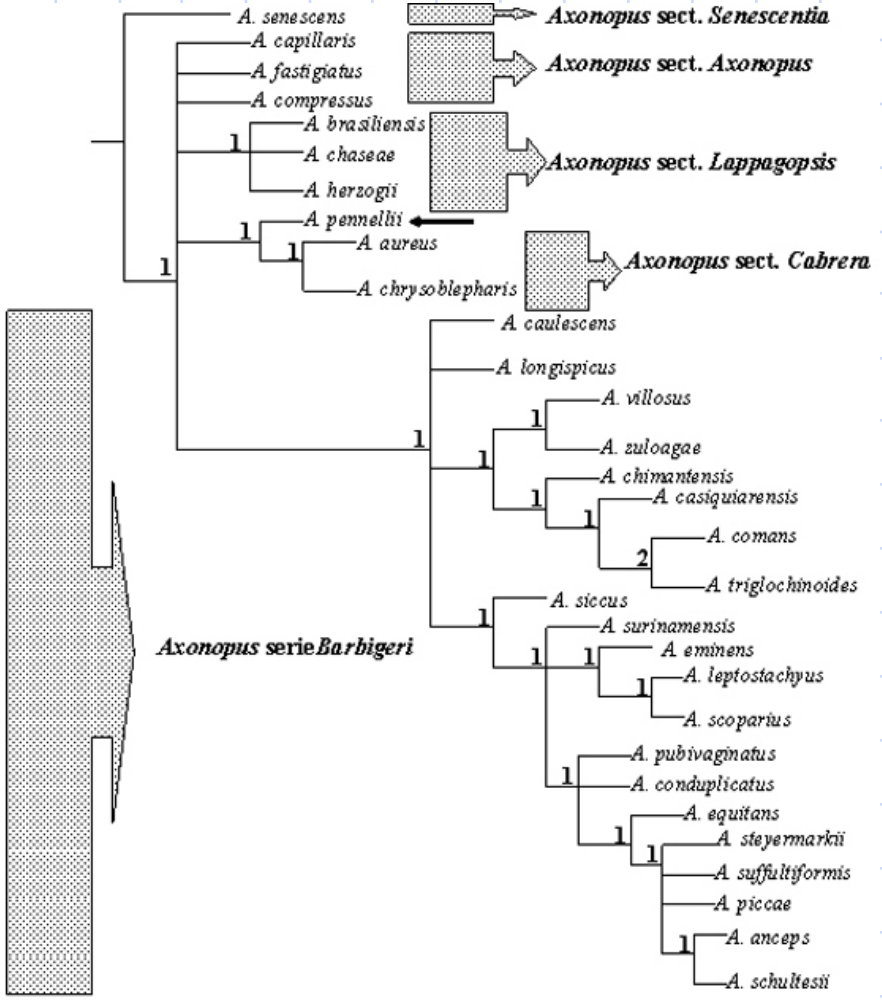
9. Genere la tabla CARRERAS { (101,'Ing Electronica',6),(12,'Tsp',2),(8,'ICivil',6)}

10. Genere la tabla CARRERAMTERIAS (pertenece) y agregue 2 relaciones.

11. Cree la vista VFILAS con una fila por cada tabla y la catidad de filas para cada una de ellas.

12. Cree la vista materializada VMFILAS idem a la anterior.

INDICES


A. senescens
A. capillaris
A. fastigiatus
A. compressus
A. brasiliensis
A. chaseae
A. herzogii
A. pennellii
A. aureus
A. chrysoblepharis
A. caulescens
A. longispicus
A. villosus
A. zuloagae
A. chimantensis
A. casiquiarensis
A. comans
A. triglochinoides
A. siccus
A. surinamensis
A. eminens
A. leptostachyus
A. scoparius
A. pubivaginatus
A. conduplicatus
A. equitans
A. steyermarkii
A. suffultiformis
A. piccae
A. anceps
A. schultesii
Axonopus sect. Senescentia
Axonopus sect. Axonopus
Axonopus sect. Lappagopsis
Axonopus sect. Cabrera
Axonopus serie Barbigeri

Que es un Índice?

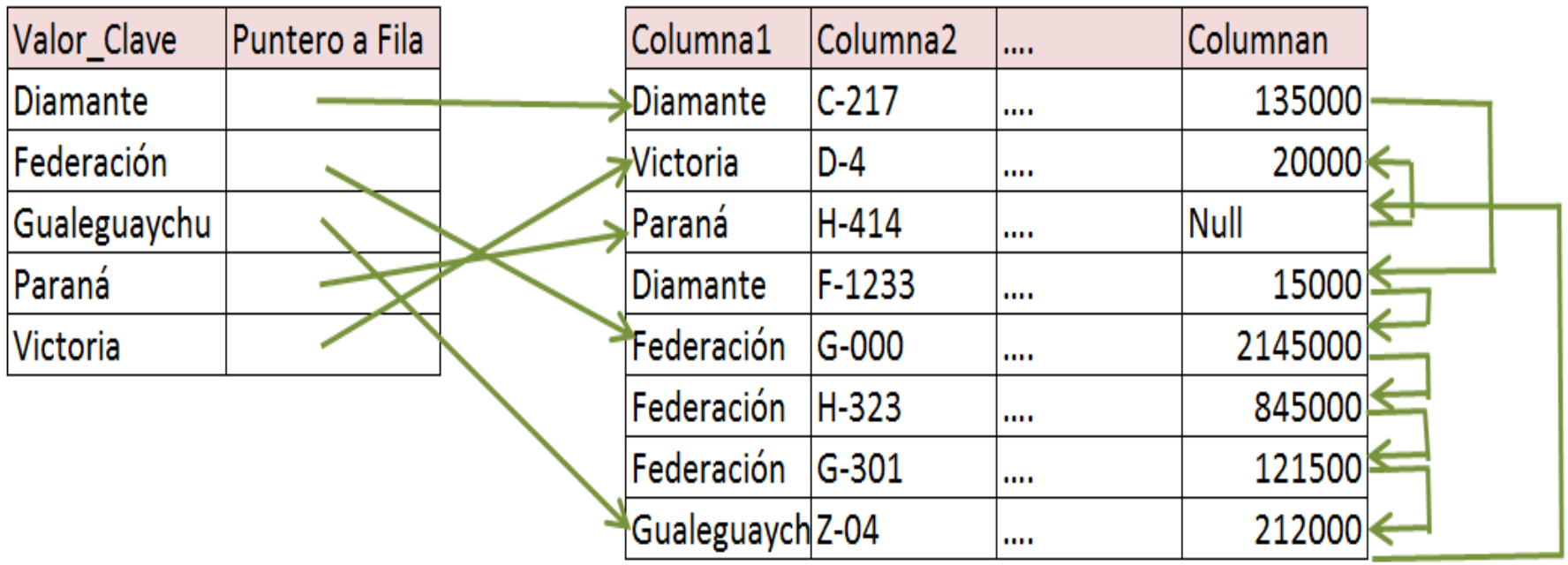
Tiene el funcionamiento similar al índice de un libro, se lee primero el índice y de aquí se obtiene el puntero al dato.

Indice

Valor_Clave	Puntero a Fila
Diamante	
Federación	
Gualeguaychu	
Paraná	
Victoria	

Tabla

Columna1	Columna2	....	Columnan
Diamante	C-217	....	135000
Victoria	D-4	....	20000
Paraná	H-414	....	Null
Diamante	F-1233	....	15000
Federación	G-000	....	2145000
Federación	H-323	....	845000
Federación	G-301	....	121500
Gualeguaych	Z-04	....	212000



Concepto

El **índice** de una **Tabla** es una estructura de datos que mejora la velocidad de las **select**, por medio de identificador de cada fila de una tabla, permitiendo un rápido acceso.

Al aumentar drásticamente la velocidad de acceso, se suelen usar, sobre aquellos columnas sobre los cuales se hacen frecuentes búsquedas o frecuentes ordenamientos.

Evitan la lectura de toda la tabla para evaluar la restricción sobre cada fila.

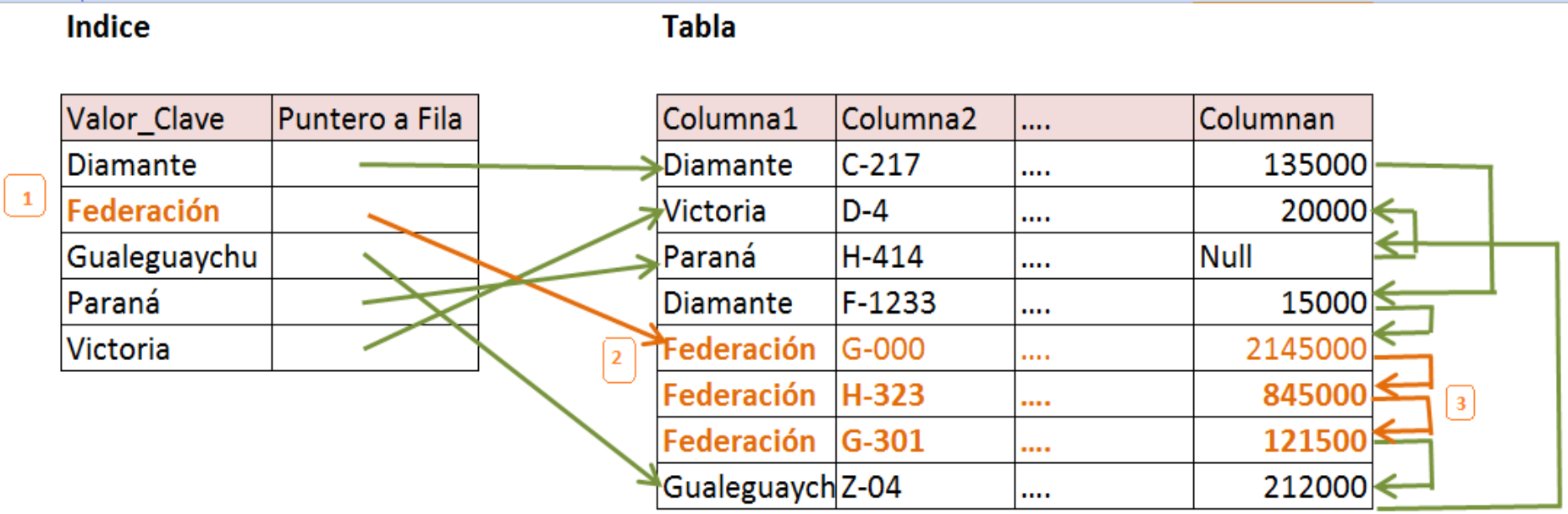
Características Relevantes

- **Son Caminos de acceso que evita leer una tabla entera cuando se usa un select con where u order by.**
- NO se usan explicitamente en el select, lo utiliza el motor (optimizador).
- Transfieren el costo computacional del select restricto a las sentencias Insert / Update / Delete;
- Internamente se implementan con árboles Btree.

Caracteristicas Relevantes

Ejemplo de acceso a

select * from tabla where clave = 'Federación'



Estrategia:

1. Acceso al Indice y localización de la clave.

2. Acceso a la 1er Fila que tiene la clave.

3. Acceso a las restantes filas por el link del indice.

Características Relevantes

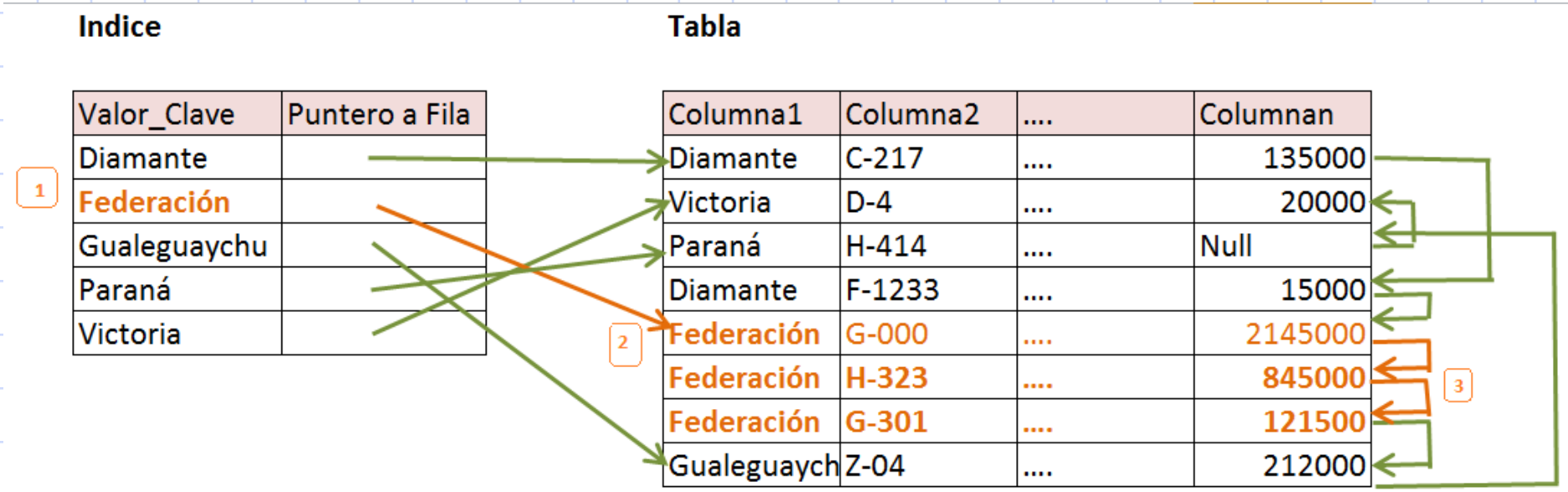
- Son Caminos de acceso que evita leer una tabla entera cuando se usa un select con where u order by**.**
- **NO se usan explicitamente en el select, lo utiliza el motor (optimizador).**
- Transfieren el costo computacional del select restricto a las sentencias Insert / Update / Delete;
- Internamente se implementan con árboles Btree.

Caracteristicas Relevantes

Ejemplo de acceso a

select * from tabla where clave = 'Federación'

No se explicita el indice en la consulta. El optimizador arma un plan de ejecución de consulta escoge el indice mas apropiado.

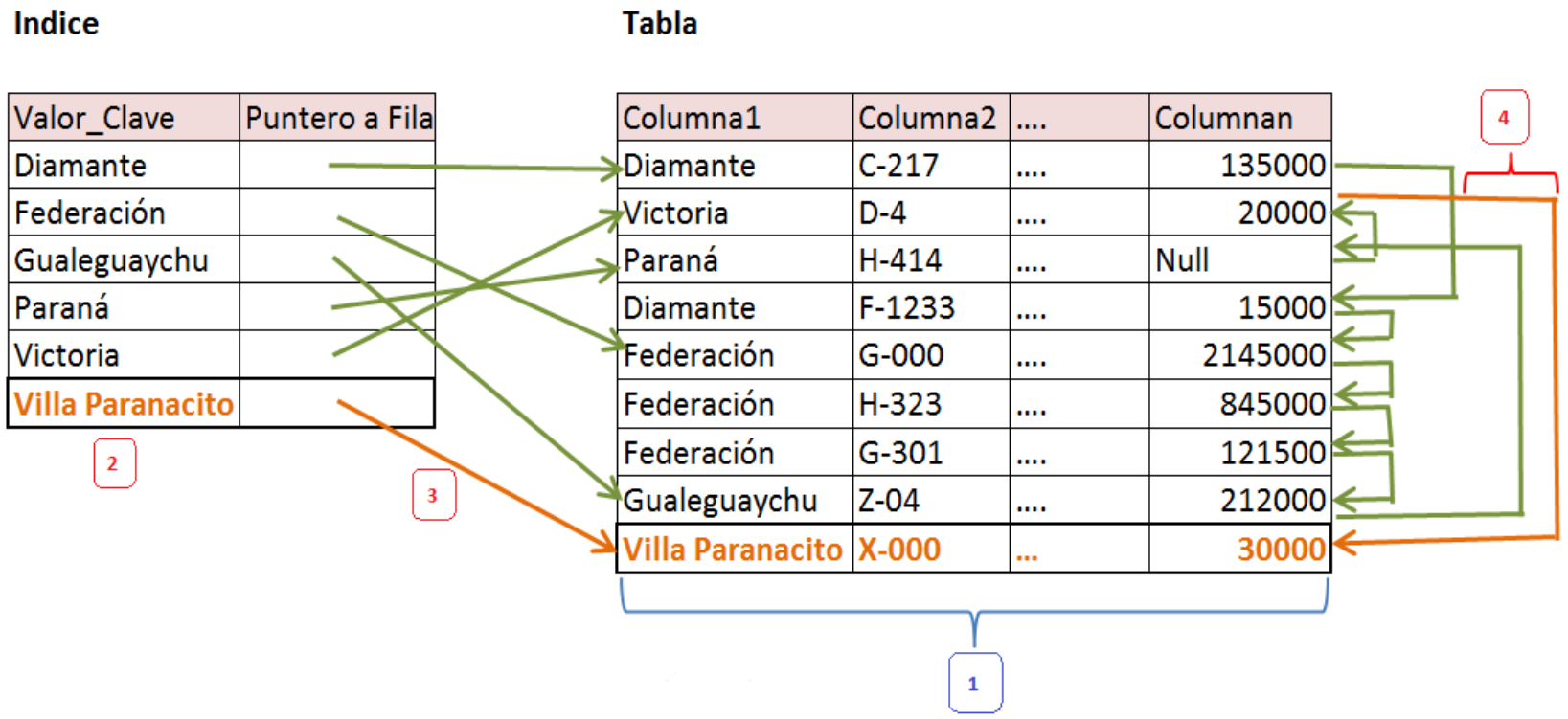


Características Relevantes

- Son Caminos de acceso que evita leer una tabla entera cuando se usa un select con where u order by.
- No se usan explicitamente en el select, lo utiliza el motor (optimizador).
- **Transfieren el costo computacional del select restricto a las sentencias Insert / Update / Delete;**
- Internamente se implementan con árboles Btree.

Características Relevantes

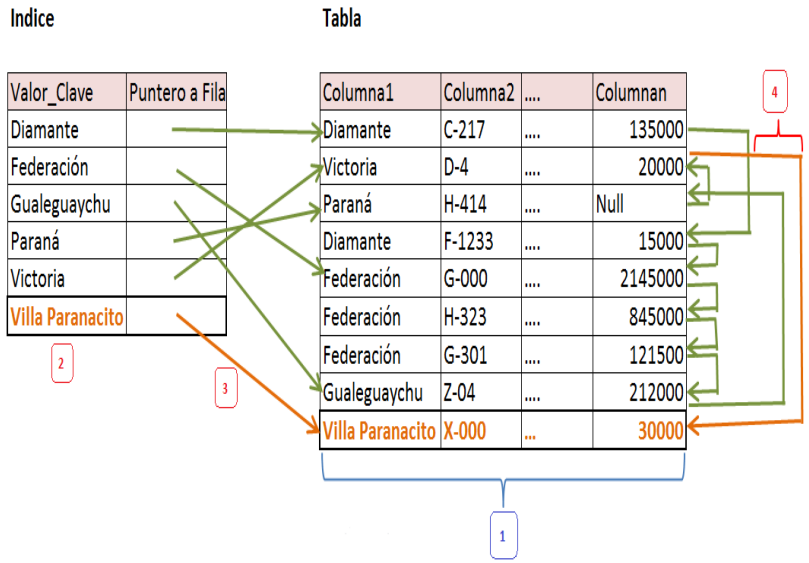
Actualización por agregado de Fila



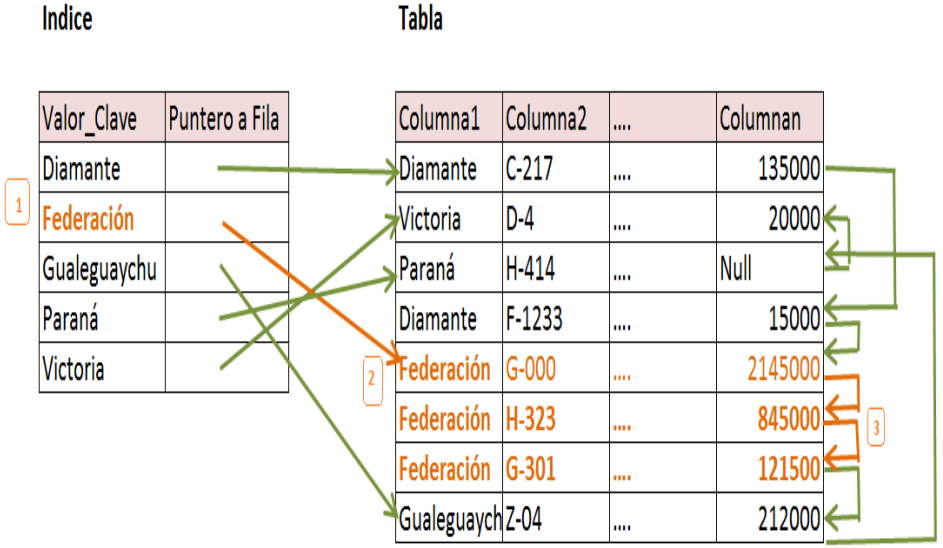
Los pasos 2 a 4 constituyen el sobrecosto trasladado a la actualización.

Características Relevantes

Actualización



Consulta



Insert-Update-Delete

\+ **Lento** *por act. Indice*

Select por la clave – u Order By

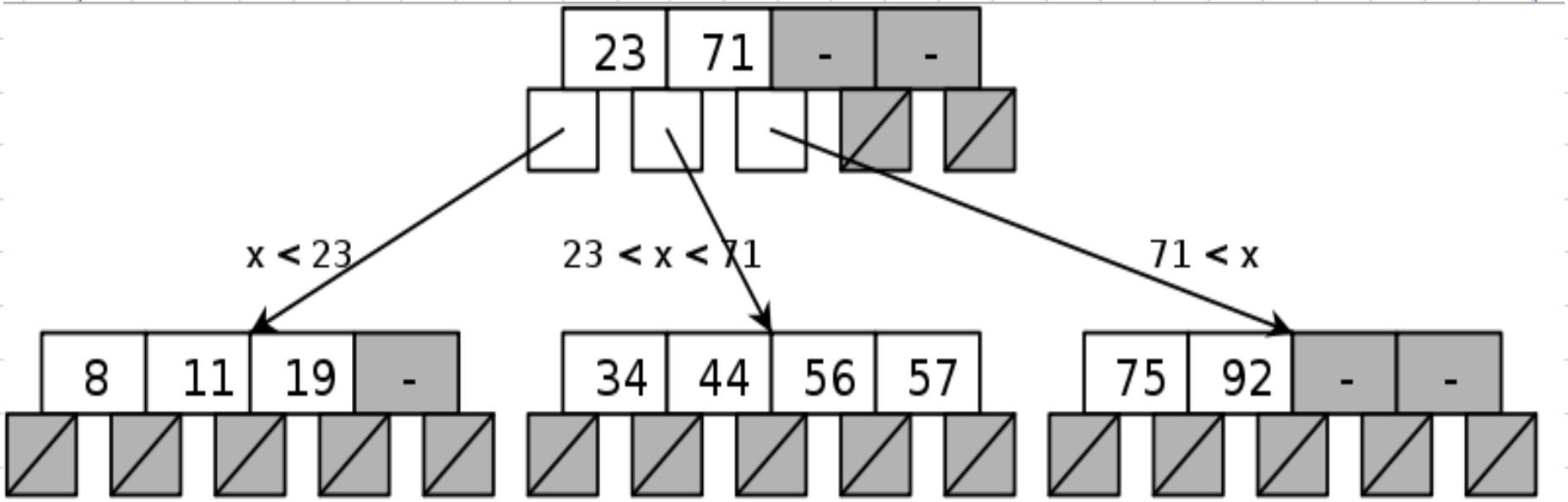
\+ **Rapido** *por uso Indice*

Características Relevantes

- Son Caminos de acceso que evita leer una tabla entera cuando se usa un select con where u order by.
- No se usan explicitamente en el select, lo utiliza el motor (optimizador).
- Transfieren el costo computacional del select restricto a las sentencias Insert / Update / Delete.
- **Internamente se implementan con árboles B y B+.**

Características Relevantes

Implementan con arboles B y B+, ej árbol B



Cada nodo del árbol posee hasta 4 claves y 5 enlaces a nodos hijos.

Por cada clave se asocia un link a las fila que contiene la clave.

Estructura de un Índice

- Una Indice esta definida por:
  - Un nombre (único para toda la base).
  - Una Condición si admite valores únicos o no (Unique). Default no únicos. *Opcional*
  - Una tabla o vista materializada.
  - Un conjunto de pares columnas que definen la clave del indice.
    - Alternativamente se puede cambiar el orden Ascendente (Asc) por descendente a cada columna (Desc).

Ejemplo de Índice

```
create index productosfamilianombre

  on productos    (familia desc, nombre);
```

Index CREATE

Sintaxis

CREATE [UNIQUE] INDEX *nombre_idx*
ON *nom_tabla*
[(col_name1 or expresion1 [asc/desc],
col_name2 or expresion2 [asc/desc], ...])]

En postgre solo nom_tabla o vista materializada.

Nombre de las indice:

- Hasta 64 caracteres (letras Nros y “_”)
- Deben comenzar con una letra
- No pueden ser palabras reservadas.

Index DROP

Sintaxis

DROP INDEX *nombre_idx*

Ej:

DROP INDEX *personas_dni;*

Índices Implícitos

- Una **PRIMARY KEY** genera un índice único.
- Una col_constraint o table_constraint **UNIQUE** genera un índice único.

Caso de Estudio 1

Una empresa cuenta con información de sus empleados en varias tablas donde la principal tiene la siguiente estructura:

```
CREATE TABLE empleados (
legajo    integer  primary key,
nombre    varchar(40), -- redundante con personas - c
   red.controlada
idcargo   integer,
idpersona integer,
idbancoctaingreso integer,          -- información confidencial
cbuctaingreso     varchar(24),      -- información confidencial
nrointerno        smallint,
mail              varchar(60),
celular           integer,          -- información confidencial
fecha_ingreso     date
);
```

Cree una vista que contenga solo (legajo, nombre, mail e interno) con el nombre `vcontactosemp`.

Caso de Estudio 1 - Soluciones

Una empresa cuenta con información de sus empleados en varias tablas donde la principal tiene la siguiente estructura:

```
CREATE TABLE empleados (
legajo    integer  primary key,
nombre    varchar(40), -- redundante con personas - c
   red.controlada
idcargo   integer,
idpersona integer,
idbancoctaingreso integer,          -- información confidencial
cbuctaingreso     varchar(24),      -- información confidencial
nrointerno        smallint,      mail              varchar(60),
celular           integer,       fecha_ingreso     date
);
```

Cree una vista que contenga solo (legajo, nombre, mail e interno) con el nombre vcontactosemp.

```
CREATE OR REPLACE VIEW vcontactosemp AS
 SELECT empleados.legajo, empleados.nombre, empleados.mail,
    empleados.nrointerno   FROM empleados;
```

Caso de Estudio 2

La tabla bancos contiene además:

```
CREATE TABLE bancos (
Id          integer  primary key,
nombre      varchar(40),
Direccioncasamatriz varchar(100)
);

ALTER TABLE empleados ADD CONSTRAINT fkbancos FOREIGN
  KEY  (idbancoctaingreso) REFERENCES bancos (id);
```

Cree una vista que contenga el nombre del empleado, legajo, nombrebanco, nroctaingreso denominada `vcuentasingresos` .

Caso de Estudio 2 - Soluciones

La tabla bancos contiene además:

```
CREATE TABLE bancos (
Id          integer  primary key,
nombre      varchar(40),
Direccioncasamatriz varchar(100)
);
ALTER TABLE empleados ADD CONSTRAINT fkbancos FOREIGN
   KEY  (idbancoctaingreso) REFERENCES bancos (id);
```

Cree una vista que contenga el nombre del empleado, legajo, nombrebanco, nroctaingreso denominada `vcuentasingresos` .

```
create view vcuentasingreso as select e.legajo,
   e.nombre, e.cbuctaingreso, b.nombre as
   banconombre    from empleados e join bancos b
   on e.idbancoctaingreso = b.id
```

Caso de Estudio 3

Un banco posee una tabla de cuentas corrientes y otra donde se registran sus movimientos:

```
CREATE TABLE cuentascorrientes (
Id        integer  primary key,
nombre    varchar(40)
--….
);

CREATE TABLE movscuentascorrientes (
Idcuentacorriente integer,
Fecha date,
Debitocredito integer,      -- 1=debito(deposito,etc), -1=credito(extrac,etc).
Importe       numeric(14,2)
);
```

Se desea crear una vista materializada con toda la información de la tabla cuenta corriente, mas los campos, fecha ultimo movimiento y saldo (el cual se obtiene por suma algebraica de importe utilizando debitocredito).

Caso de Estudio 3 - Soluciones

Un banco posee una tabla de cuentas corrientes y otra donde se registran sus movimientos:

```
CREATE TABLE cuentascorrientes (
Id        integer  primary key,
nombre    varchar(40)
--….
);

CREATE TABLE movscuentascorrientes (
Idcuentacorriente integer,
Fecha date,
Debitocredito integer,      -- 1=debito(deposito,etc), -1=credito(extrac,etc).
Importe       numeric(14,2)
);
```

Se desea crear una vista materializada con toda la información de la tabla cuenta corriente, mas los campos, fecha ultimo movimiento y saldo (el cual se obtiene por suma algebraica de importe utilizando debitocredito).

Lo veremos cuando tratemos trigger.

Caso de Estudio 4

- Una empresa cuenta con información de sus empleados en varias tablas donde la principal tiene la siguiente estructura:

```
CREATE TABLE empleados (
legajo    integer  primary key,
nombre    varchar(40), -- redundante con personas - c
   red.controlada
idcargo   integer,
idpersona integer,
idbancoctaingreso integer,          -- información confidencial
cbuctaingreso     varchar(24),      -- información confidencial
nrointerno        smallint,
mail              varchar(60),
celular           integer,          -- información confidencial
fecha_ingreso     date
);
```

1) Cree un indice unico para celular que no admita valores repetidos .

2)Cree un indice unico por idcargo y nombre.

Caso de Estudio 4 - Soluciones

- Una empresa cuenta con información de sus empleados en varias tablas donde la principal tiene la siguiente estructura:

```
CREATE TABLE empleados (
legajo    integer  primary key,
nombre    varchar(40), -- redundante con personas - c red.controlada
idcargo   integer,
idpersona integer,
idbancoctaingreso integer,          -- información confidencial
cbuctaingreso     varchar(24),      -- información confidencial
nrointerno        smallint,
mail              varchar(60),
celular           integer,          -- información confidencial
fecha_ingreso     date
);
```

1) Cree un indice unico para celular que no admita valores repetidos .

```
CREATE UNIQUE INDEX celulares  ON empleados  (celular);
```

2)Cree un indice unico por idcargo y nombre.

```
CREATE UNIQUE INDEX emp_idcargo_nombre  ON empleados (idcargo,nombre);
```

Caso de Estudio 5

- Se crea una tabla de cargos que se relaciona con personas.idcargo:

```
CREATE TABLE cargos (
idcargo       integer  primary key,
descripcion    varchar(40),
abreviatura    varchar(8),
jerarquia      varchar(20)
);
```

1) Cree una constraint para jerarquia que limite las opciones a “auxiliar”,”encargado”,”jefe area”,”gerente area”,”gerente general” .

2)Cree una regla de integridad referencial entre cargos y empleados, proponga el criterio de propagación y actualización.

3) Cree una vista que se llame vempleadoscargos con ambas.

4) Cree un indice por cargos. Jerarquia asc y cargos.descripcion desc.

Caso de Estudio 5 - Soluciones

1) Cree una constraint para jerarquia que limite las opciones a “auxiliar”,”encargado”,”jefe area”,”gerente area”,”gerente general” .

```
alter table cargos add constraint cargos_jerarquia
   check(jerarquia in ('auxiliar','encargado','jefe
   area','gerente area','gerente general'));
```

2)Cree una regla de integridad referencial entre cargos y empleados, proponga el criterio de propagación y actualización.

```
alter table empleados add constraint empleados_cargos_fk
   foreign key (idcargo) references cargos (idcargo) on
   delete set null on update set null;
```

3) Cree una vista que se llame vempleadoscargos con ambas.

```
create view vempleadoscargos as select e.*, c.descripcion
   as cargo_descripcion, c.abreviatura as
   cargo_abreviatura, c.jerarquia as cargo_jerarquia from
   empleados e join cargos c on c.idcargo = e.idcargo;
```

4) Cree un indice por cargos.jerarquia asc y cargos.descripcion desc.

```
Create  index  cargos_jerarquia_nombre on cargos
   (jerarquia asc, descripcion desc);
```

Fuentes

Libro: Fundamentos de Bases de Datos
Autor: Silberschatz / Korth / Sudarshan
Editorial: Mc Graw Hill

Libro: Introducción a los SISTEMAS DE BASES DE DATOS
Autor: C.J. Date
Editorial: Addison Wesley

http://www.firebird.com.mx/descargas/documentos/tema_3-ddl.pdf
http://www.postgresql.org/docs/9.1/static/sql-createindex.html
http://www.postgresql.org/docs/9.1/static/sql-createview.html