



<http://www.eqsoft.net>

Correo: informes@eqsoft.net

Teléfonos: (51)(1)-5645424/997244926/997003957



PostgreSQL Como funciona una Base de Datos por dentro

Ernesto Quiñones Azcárate
ernesto@eqsoft.net



<http://www.eqsoft.net>

Correo: informes@eqsoft.net

Teléfonos: (51)(1)-5645424/997244926/997003957

Agenda

1. Una corta introducción
2. Una DBMS por dentro
3. Almacenamiento y organización de los datos
4. Los índices
5. Como se procesa un Query
6. Concurrencia : Transacciones y bloqueos



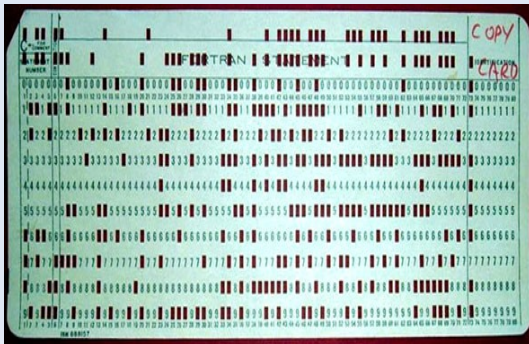
<http://www.eqsoft.net>

Correo: informes@eqsoft.net

Teléfonos: (51)(1)-5645424/997244926/997003957

1. Una corta introducción

- Las bases de datos como concepto nacen en los 1960.
- La primera vez que se uso el termino fue en un proyecto del ejercito norteamericano en una fecha no determinada entre los 1950 y principios de los 1960.
- Las bases de datos libres y las privativas tienen casi el mismo tiempo de existencia (<http://tinyurl.com/l5fern>).
- Existen diversos tipos de bases de datos:
 - Planas
 - Relacionales
 - Objeto-relaciones
 - Orientadas a objetos
 - XML
 - Orientadas a data “sin forma”, ejemplo: La web
 - Datos gráficos, espaciales, etc.
 - ...muchas otras por venir





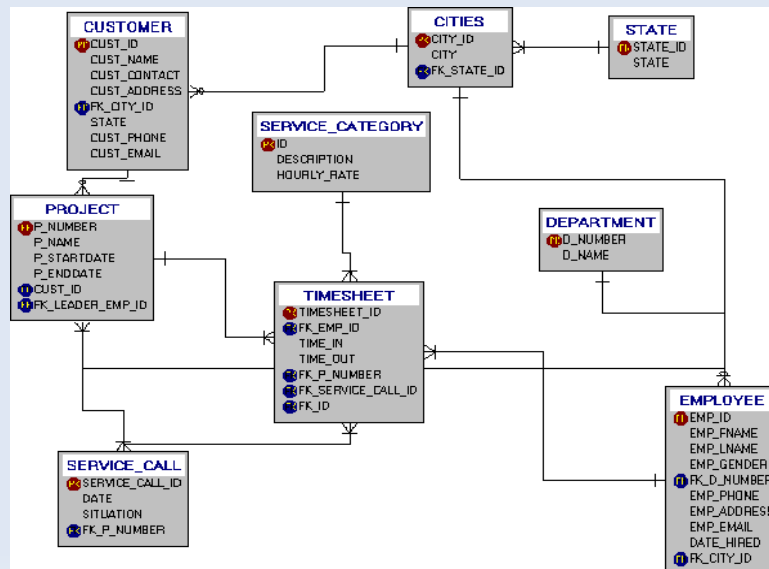
<http://www.eqsoft.net>

Correo: informes@eqsoft.net

Teléfonos: (51)(1)-5645424/997244926/997003957

1. Una corta introducción

- Actualmente se genera mas información de la que se puede almacenar (<http://tinyurl.com/lwuvzd>)
- La mayoría de esta data “no tiene forma”, son fotos, videos, web, etc.
- Sin embargo aún necesitamos guardar este tipo de datos:

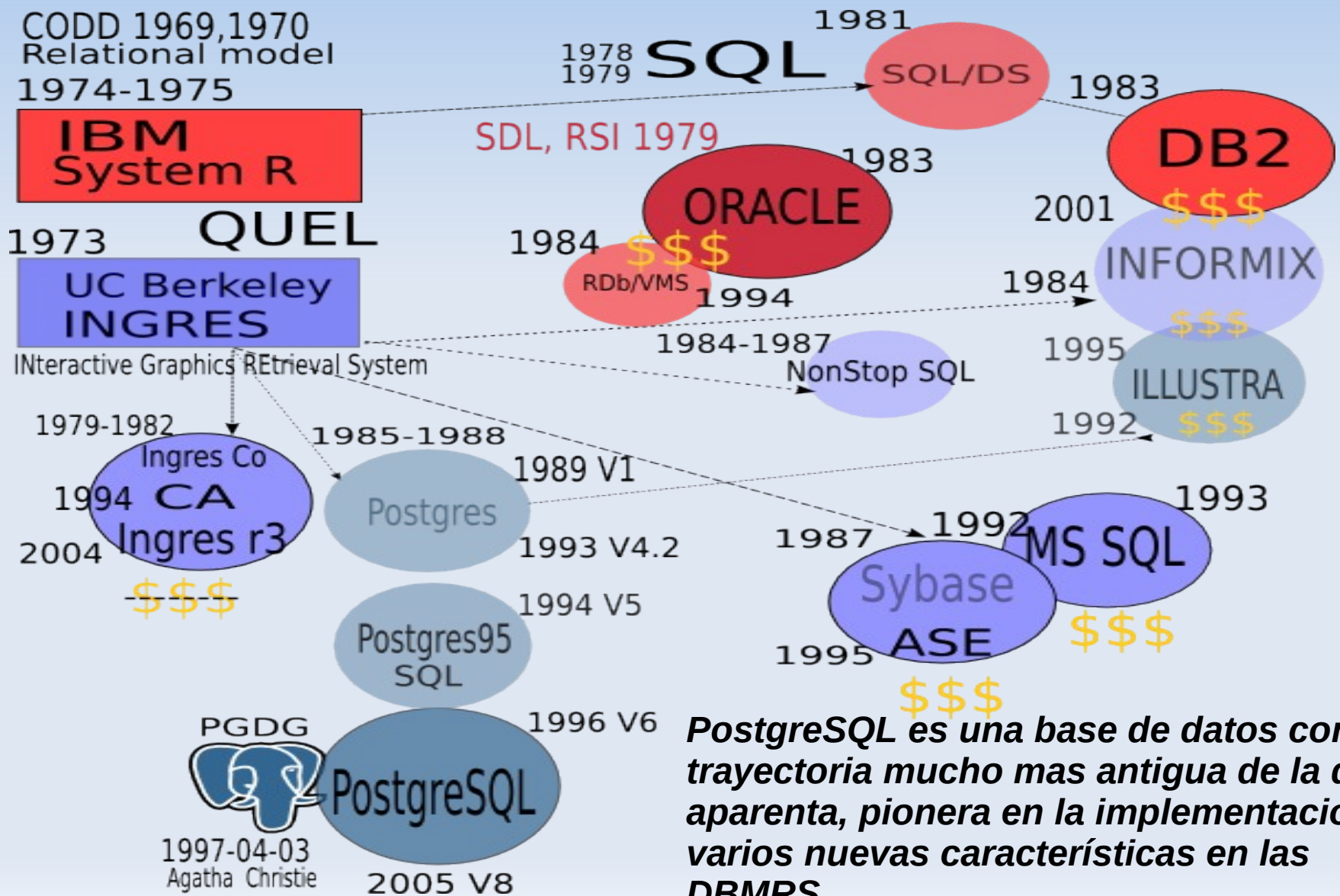


RICARDO GITTO		Factura	
Representante Exclusivo en Bell Ville y Zona de:		N° 0001-00 000112	
PISOS BLANGINO			
Mosaicos Compactos - Lozetas Graníticas			
Pisos Marmolados - Umbrales - Escalones - Mesadas			
Bv. ASCASUBI 582 - BELL VILLE (Cha.)		C.U.I.T. 20-11622202-8	
Tel. 03534 - 422958 - Cel. 15599883		Ing. Brutos: 202-03005-0	
RESPONSABLE MONOTRIBUTO		Inicio Activ.: 01 - 10 - 04	
Señor (es):		Tel.:	
Domicilio:			
Responsable Inscripto <input type="checkbox"/>		C.U.I.T. N°	
IVA: Exento <input type="checkbox"/> No resp. <input type="checkbox"/> Corrent, Final <input type="checkbox"/>		Ing. Brutos	
Responsable Monotributo <input type="checkbox"/>		REMITO N°	
Cond. de Venta: CONTADO <input type="checkbox"/> CTA. CTE. <input type="checkbox"/>			
CANT.	DETALLE	P. UNIT.	IMPORTE
32 m²	Lozetas 16/16x16	25.00	800.00
50 m²	Lozetas 16/16x16	17.00	850.00
			895.00
PAGADO			
RICARDO GITTO			
VENDEDOR			
Juan B.N. Blangino			
Son Pesos:		TOTAL \$ 895.00	
IMPRESA MENARDI de Ernesto R. Menardi - Amaghoro 135 Bell Ville		FECHA DE IMPRESION	
C.U.I.T. 20-10050966-1 Ing. Brutos: 202-101-7-6 Municipalidad M-0227		28 - 03 - 2007	
ORIGINAL (Blanco) DUPLICADO (Color)		N° 101 # 200	

- Y para ella existe este tipo de base de datos.



1. Una corta introducción





2. Una DBMS por dentro

PostgreSQL tiene una arquitectura que involucra muchos estilos, en su nivel mas alto es un esquema clásico cliente-servidor, mientras que el acceso a la data es un esquema en capas.

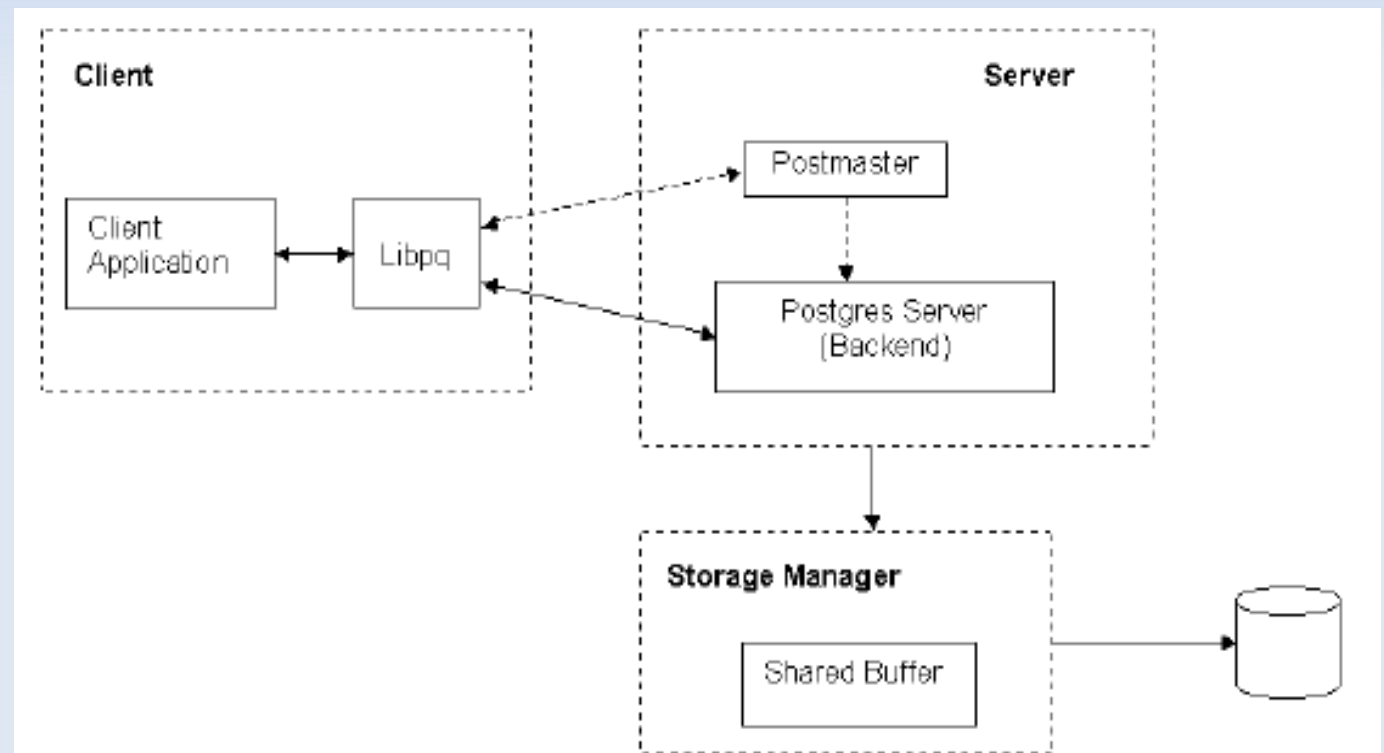


Figure 1 PostgreSQL System Concept Architecture



2. Una DBMS por dentro

- El Libpq es el responsable de manipular las comunicaciones entre la aplicación cliente y el postmaster (servicio del PostgreSQL en el servidor).**
- El server esta compuesto por 2 grandes subsistemas, el “Postmaster” que es el responsable de aceptar las comunicaciones con el cliente y autentificar y dar acceso. El “Postgre” se encarga de la administración de los querys y comandos enviados por el cliente. PostgreSQL trabaja bajo el concepto de “process per user”, eso significa un solo procesos cliente por conexión. Tanto el Postmaster como el Postgre deben estar junto en el mismo servidor siempre.**
- El Storage Manager es responsable de la administración general de almacenamiento de los datos, controla todos los trabajos del back-end incluido la administración del buffer, archivos, bloqueos y control de la consistencia de la información.**



<http://www.eqsoft.net>

Correo: informes@eqsoft.net

Teléfonos: (51)(1)-5645424/997244926/997003957

3. Almacenamiento y organización de datos

La data siempre se va a guardar en “disco” (esto puede no ser literalmente un HD).

Esto genera un intenso trabajo de I/O, cuando leemos la data la sacamos del “disco” para pasarla a la RAM, cuando escribimos la bajamos de la RAM al “disco”.

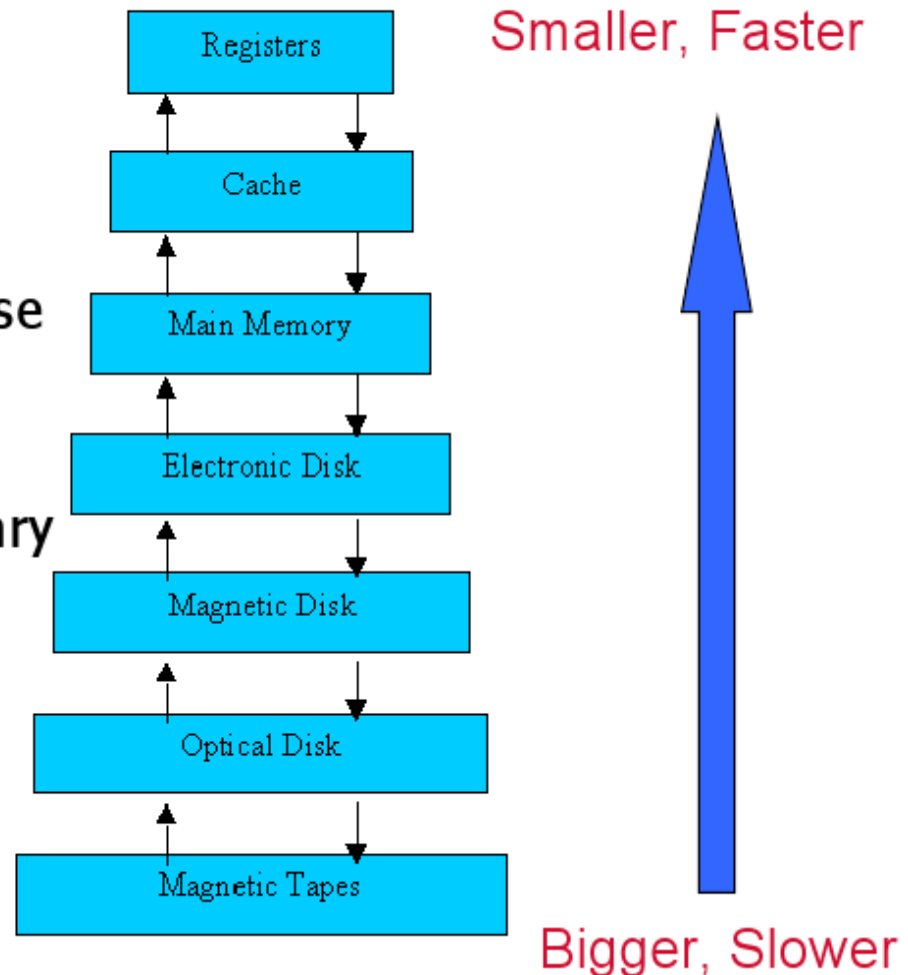


3. Almacenamiento y Organización de los Datos



The Storage Hierarchy

- Main memory (RAM) for currently used data.
- Disk for the main database (secondary storage).
- Tapes for archiving older versions of the data (tertiary storage).





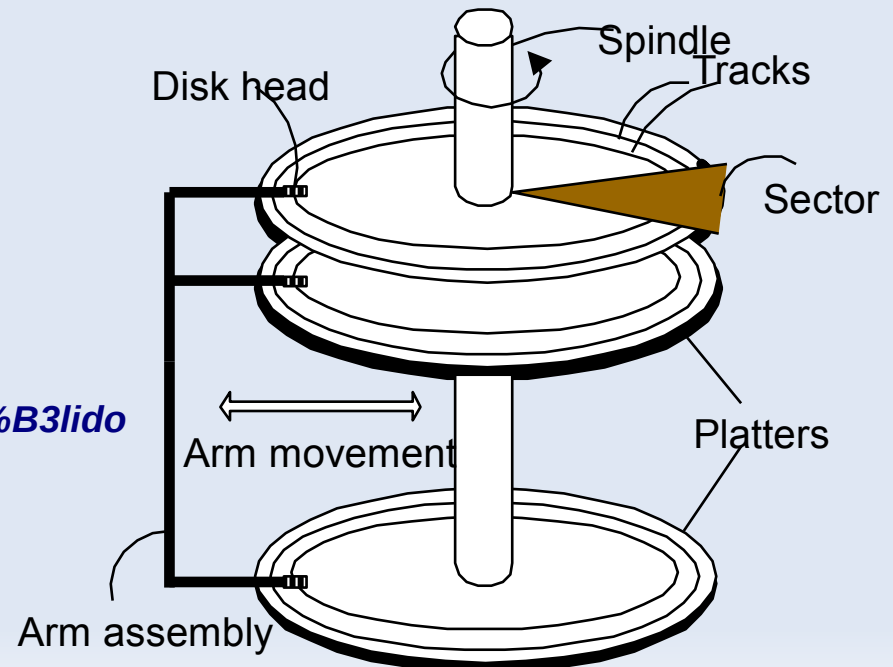
3. Almacenamiento y organización de datos

La data en cualquier DBMS se almacena en pequeños bloques de disco llamadas “páginas”.

Estás “páginas” se guardan en un disco en diferentes posiciones físicas, mucha dispersión creará una baja performance en la dbms, en sistemas de almacenamiento como los HD (osea casi el 99%) esto es un gran problema.

Afortunadamente ahora existen Soluciones basadas en Discos de Estado Solido que ayudan con Este problema.

http://es.wikipedia.org/wiki/Unidad_de_estado_s%C3%B3lido





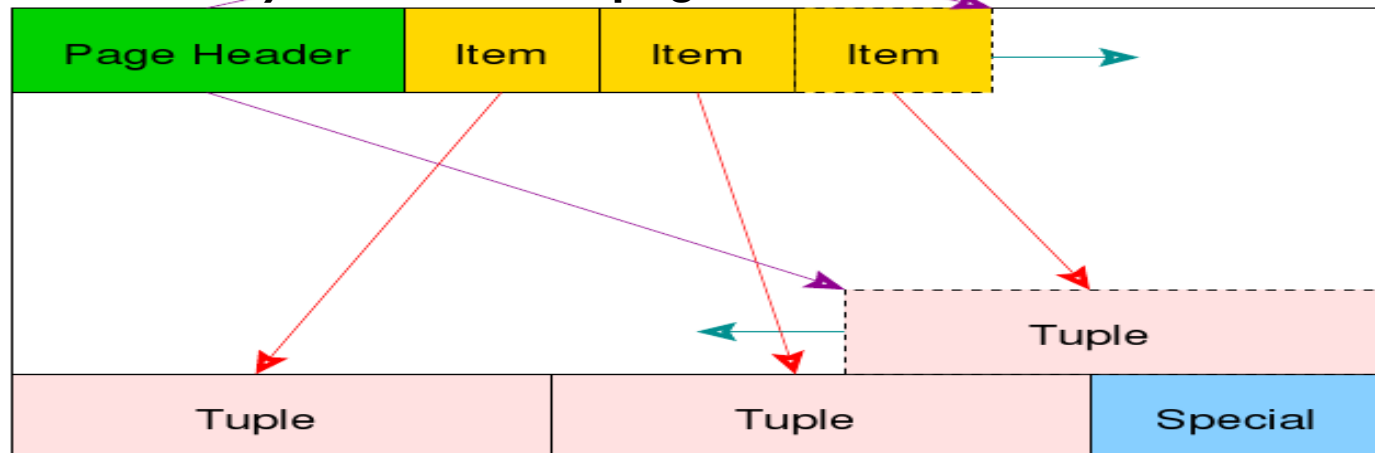
3. Almacenamiento y organización de datos

El tamaño de una página en PostgreSQL puede ser tan pequeño como 8k (por defecto) hasta un máximo de 32k y no se permite que un tupla pueda ser mas grande que una página de tamaño.

Cuando se necesita guardar data muy grande (un video por ejemplo) la data es comprimida y partida en pequeñas “filas” que se guardan en una tabla paralela, esto es transparente para el usuario (<http://www.postgresql.org/docs/8.4/interactive/storage-toast.html>).

Las páginas contienen “items” los cuales apuntan a tuplas o entradas de índices junto con metadata.

Para el caso de PostgreSQL las operaciones de R/W primero se consulta al Buffer Manager (memoria RAM) si contiene la página.





3. Almacenamiento y organización de datos

PostgreSQL posee un solo “Storage Manager” (MySQL tiene 5 o más por ejemplo), este esta compuesto por varios módulos que proveen administración de las transacciones y acceso a los objetos de la base de datos.

Los módulos se programaron bajo 3 lineamientos bien claros:

- Manejar transacciones sin necesidad de escribir código complejo de recuperación en caso de caídas.**
- Mantener versiones históricas de la data bajo el concepto de “graba una vez, lee muchas veces”.**
- Tomar las ventajas que ofrece el hardware especializado como multiprocesadores, memoria no volátil, etc.**



3. Almacenamiento y organización de datos

Los módulos que componen el Storage Manager son:

- **Transaction System**
- **Relational Storage**
- **Time Management**
- **Concurrency Control y Timestamp Management**
- **Record Acces**

PostgreSQL siempre esta añadiendo data, la data modificada o borrada realmente no se modifica o se borra, las páginas donde ellas están almacenadas se marca como “no visible” y se inserta un nuevo registro completo con un clon de toda la data (como se maneja esto en detalle se explica mas adelante).

Esto hace que la base de datos ocupe mucho espacio y afecta el “tiempo de acceso” a la data.



3. Almacenamiento y organización de datos

Ejemplo de un Vacuum Full

	Plato 1	Plato 2	Plato 3																																													
Antes de un Vacuum	<table><tr><td>1</td><td>2</td><td>E</td></tr><tr><td>E</td><td>3</td><td>4</td></tr><tr><td>5</td><td>6</td><td>7</td></tr><tr><td>E</td><td>OF</td><td>OF</td></tr><tr><td>SD</td><td>SD</td><td>E</td></tr></table>	1	2	E	E	3	4	5	6	7	E	OF	OF	SD	SD	E	<table><tr><td>E</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>8</td><td>OF</td></tr><tr><td>OF</td><td>9</td><td>OF</td></tr><tr><td>OF</td><td>10</td><td>OF</td></tr></table>	E	SD	SD	SD	SD	SD	SD	8	OF	OF	9	OF	OF	10	OF	<table><tr><td>11</td><td>OF</td><td>OF</td></tr><tr><td>SD</td><td>12</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>13</td><td>E</td><td>E</td></tr><tr><td>E</td><td>E</td><td>E</td></tr></table>	11	OF	OF	SD	12	SD	SD	SD	SD	13	E	E	E	E	E
	1	2	E																																													
	E	3	4																																													
	5	6	7																																													
	E	OF	OF																																													
SD	SD	E																																														
E	SD	SD																																														
SD	SD	SD																																														
SD	8	OF																																														
OF	9	OF																																														
OF	10	OF																																														
11	OF	OF																																														
SD	12	SD																																														
SD	SD	SD																																														
13	E	E																																														
E	E	E																																														
Despues de un Vacuum	<table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr><tr><td>10</td><td>OF</td><td>OF</td></tr><tr><td>11</td><td>12</td><td>13</td></tr></table>	1	2	3	4	5	6	7	8	9	10	OF	OF	11	12	13	<table><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>OF</td></tr><tr><td>OF</td><td>SD</td><td>OF</td></tr><tr><td>OF</td><td>SD</td><td>OF</td></tr></table>	SD	SD	SD	SD	SD	SD	SD	SD	OF	OF	SD	OF	OF	SD	OF	<table><tr><td>SD</td><td>OF</td><td>OF</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr></table>	SD	OF	OF	SD	SD	SD	SD	SD	SD	SD	SD	SD	SD	SD	SD
	1	2	3																																													
	4	5	6																																													
	7	8	9																																													
	10	OF	OF																																													
11	12	13																																														
SD	SD	SD																																														
SD	SD	SD																																														
SD	SD	OF																																														
OF	SD	OF																																														
OF	SD	OF																																														
SD	OF	OF																																														
SD	SD	SD																																														
SD	SD	SD																																														
SD	SD	SD																																														
SD	SD	SD																																														

Existe un “tiempo de acceso” para llegar a la data (sea read o write) que depende de:

- Tiempo de búsqueda del OS en mover los brazos del disco duro.
- Tiempo de rotación de los discos para que el brazo encuentre la posición física donde esta la data.
- Tiempo de transferencia de R/W de la data del disco a la memoria.

Hay que buscar que reducir este tiempo para que el acceso a la data sea mas rápido.



3. Almacenamiento y organización de datos

Ejemplo de un Lazy Vacuum

Antes de un Vacuum	Plato 1	Plato 2	Plato 3																																													
	<table><tr><td>1</td><td>2</td><td>ED</td></tr><tr><td>ED</td><td>3</td><td>ED</td></tr><tr><td>OF</td><td>OF</td><td>OF</td></tr><tr><td>7</td><td>8</td><td>OF</td></tr><tr><td>OF</td><td>OF</td><td>SD</td></tr></table>	1	2	ED	ED	3	ED	OF	OF	OF	7	8	OF	OF	OF	SD	<table><tr><td>EI</td><td>EI</td><td>5</td></tr><tr><td>4</td><td>OF</td><td>OF</td></tr><tr><td>OF</td><td>OF</td><td>OF</td></tr><tr><td>13</td><td>14</td><td>6</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr></table>	EI	EI	5	4	OF	OF	OF	OF	OF	13	14	6	SD	SD	SD	<table><tr><td>10</td><td>12</td><td>11</td></tr><tr><td>ED</td><td>ED</td><td>SD</td></tr><tr><td>SD</td><td>EI</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>9</td><td>SD</td><td>SD</td></tr></table>	10	12	11	ED	ED	SD	SD	EI	SD	SD	SD	SD	9	SD	SD
	1	2	ED																																													
	ED	3	ED																																													
	OF	OF	OF																																													
7	8	OF																																														
OF	OF	SD																																														
EI	EI	5																																														
4	OF	OF																																														
OF	OF	OF																																														
13	14	6																																														
SD	SD	SD																																														
10	12	11																																														
ED	ED	SD																																														
SD	EI	SD																																														
SD	SD	SD																																														
9	SD	SD																																														
Despues de un Vacuum	Plato 1	Plato 2	Plato 3																																													
	<table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>OF</td><td>OF</td><td>OF</td></tr><tr><td>7</td><td>8</td><td>OF</td></tr><tr><td>OF</td><td>OF</td><td>9</td></tr></table>	1	2	3	4	5	6	OF	OF	OF	7	8	OF	OF	OF	9	<table><tr><td>EI</td><td>EI</td><td>10</td></tr><tr><td>11</td><td>OF</td><td>OF</td></tr><tr><td>OF</td><td>OF</td><td>OF</td></tr><tr><td>12</td><td>13</td><td>14</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr></table>	EI	EI	10	11	OF	OF	OF	OF	OF	12	13	14	SD	SD	SD	<table><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>EI</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr><tr><td>SD</td><td>SD</td><td>SD</td></tr></table>	SD	SD	SD	SD	SD	SD	SD	EI	SD	SD	SD	SD	SD	SD	SD
	1	2	3																																													
	4	5	6																																													
	OF	OF	OF																																													
7	8	OF																																														
OF	OF	9																																														
EI	EI	10																																														
11	OF	OF																																														
OF	OF	OF																																														
12	13	14																																														
SD	SD	SD																																														
SD	SD	SD																																														
SD	SD	SD																																														
SD	EI	SD																																														
SD	SD	SD																																														
SD	SD	SD																																														

La operación de Vacuum es importante porque nos ayuda a mejorar la performance del acceso a la data y la optimización del uso de espacio en disco.

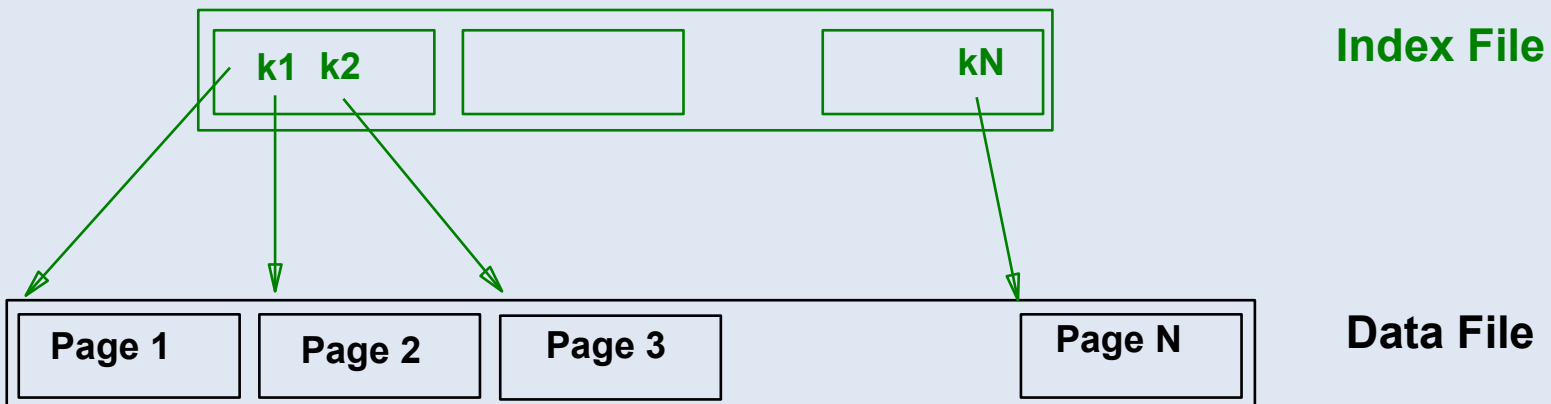
El método del Lazy Vacuum es más usado que el Vacuum Full.



4. Los índices

Cada tipo de búsqueda tienen un tipo de índice adecuado para trabajarla, básicamente un índice es un “archivo” donde esta parte de la data y estructura de una tabla con las “search key” de búsqueda.

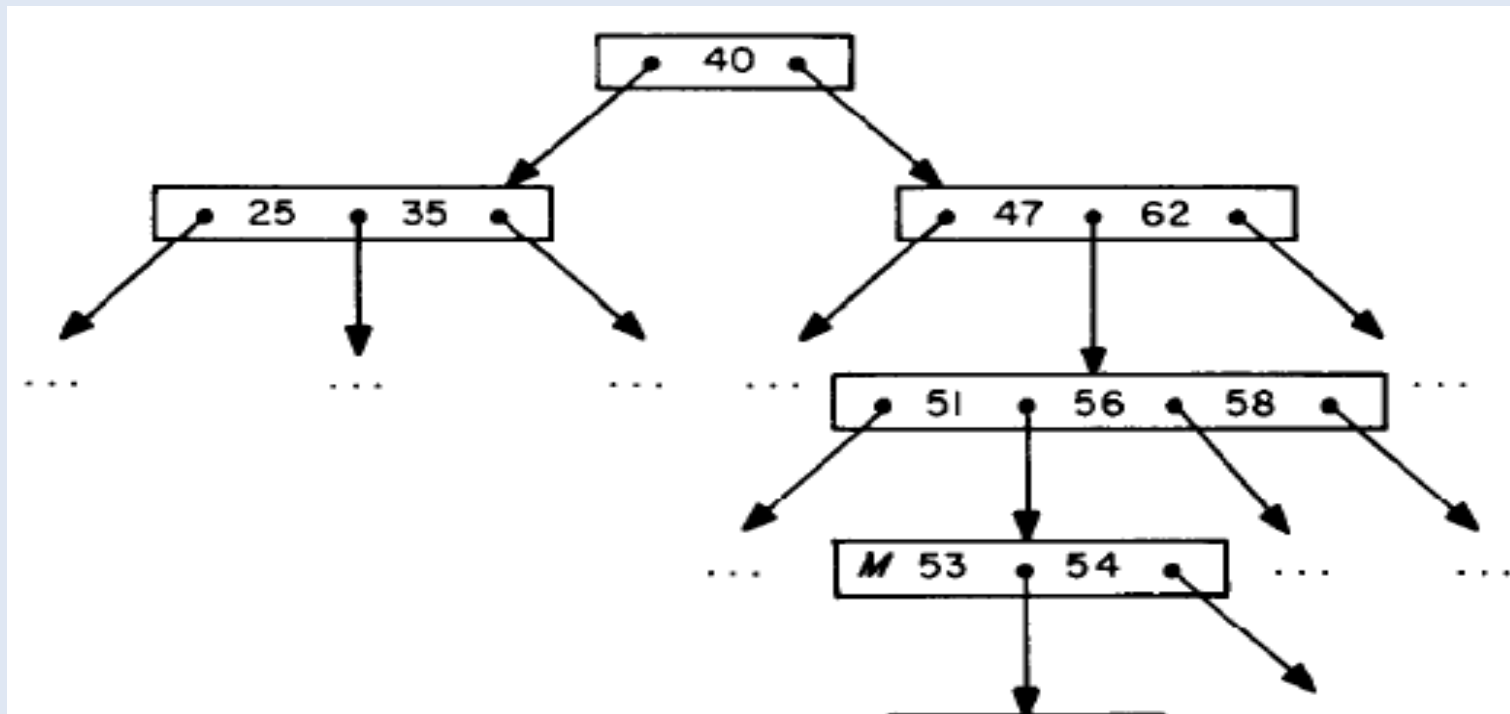
En simple como es un índice:





4. Los índices

La forma clásica es buscar por extremos y medios (búsqueda binaria), pero esto hace que sea altamente costosa la búsqueda, entonces es preferible organizar los índices en estructuras mas eficientes como los árboles.



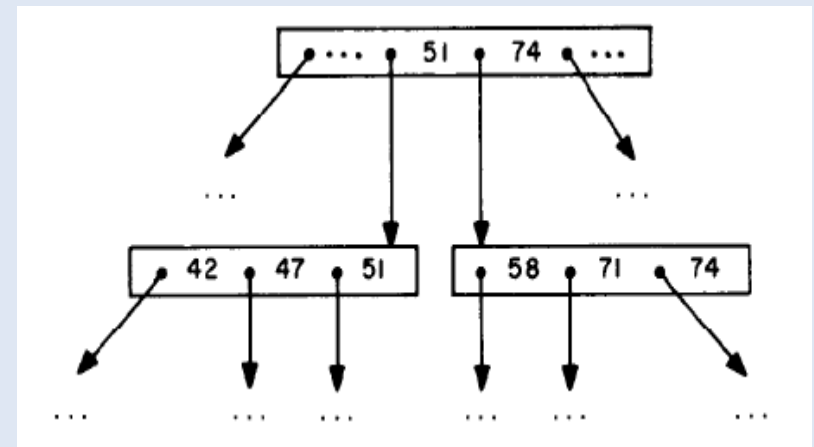
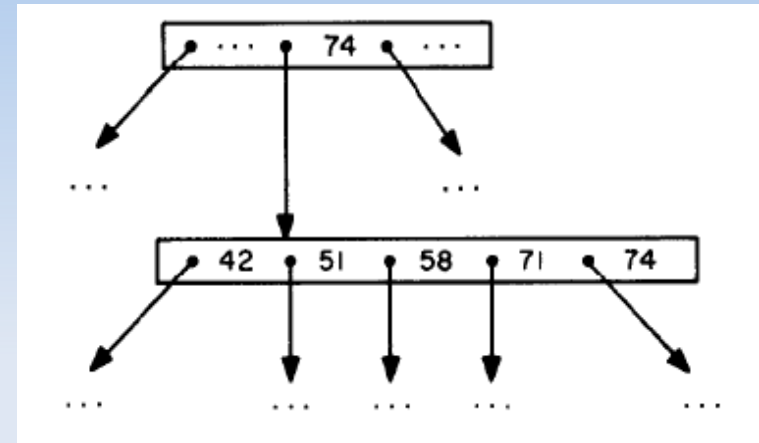


4. Los índices

La cantidad de elementos en un nodo es delimitado por una constante predefinida, exactamente no deben haber mas de 2 veces elementos que la constante.

Los nuevos elementos son insertados en el nodo que le corresponda según su valor, en caso de que el nodo haya copado su capacidad máxima de elementos entonces el nodo se divide en 2 partes iguales y se crea una “hoja” superior con los índices apropiados.

(ejemplo se inserta el elemento 47)



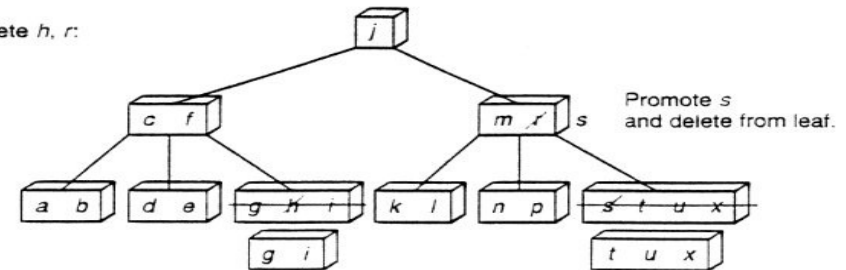


4. Los índices

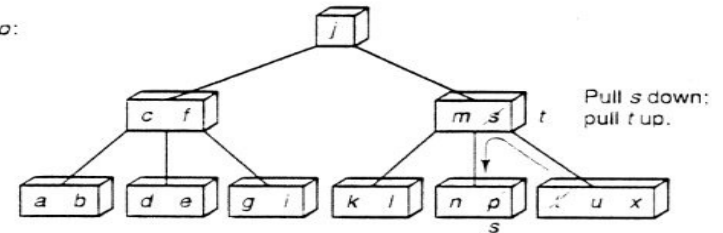
Para borrar un elemento se realiza la búsqueda del mismo, al llegar al nodo que lo contiene se bloquea.

El nodo se trabaja en “memoria” sacando el elemento a borrar y se reescribe totalmente el nodo, ocasionalmente el nodo queda con menos elementos la cantidad de la constante definida de elementos máximos.

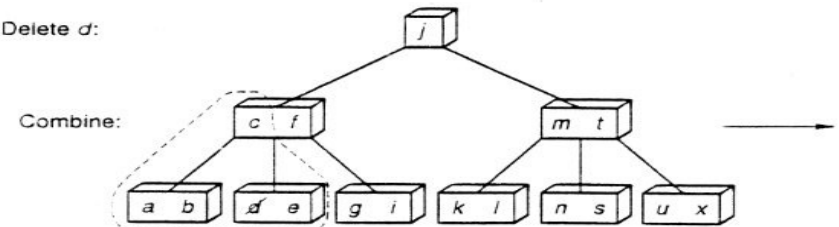
1. Delete *h*, *r*:



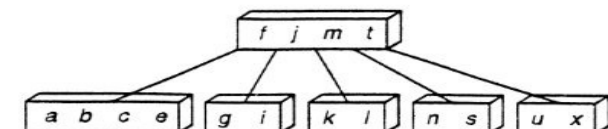
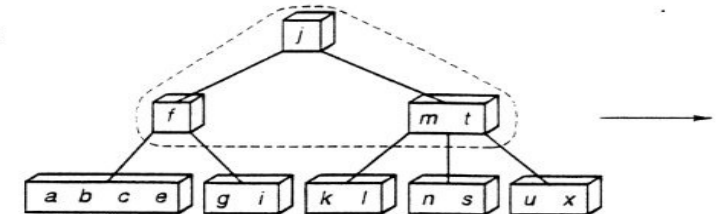
2. Delete *p*:



3. Delete *d*:



Combine:





4. Los índices

Características de los índices:

- Según la estructura de ordenamiento del mismo:
 - Tree-based (Btree, Rtree), hash-based, other (Tsearch2)
- Según el ordenamiento físico de la data
 - Clustered vs. Unclustered Indexes
- Según la asociación con la data :
 - primary vs. secondary indexes, manejo de duplicados
- Según la cantidad de columnas que incorpore en la “search key”
 - Multi-part key = “Composite Indexes”



4. Los índices

La estructura de ordenamiento del índice nos dice que tipo de selección soporta, en el caso de PostgreSQL tenemos los siguientes :

- B-tree (<, <=, =, >=, > y modificadores)
- Hash (= y sin soporte de NULL data)
- GiST (<<, &<, &>, >>, <<|, &<|, |&>, |>>, @>, <@, ~=, &&; estos son operadores para datos geométricos)
- GIN (<@, @>, =, &&; estos son operadores para datos tipo array y para “Full Text Searching” dentro de documentos a través de lexemas <http://es.wikipedia.org/wiki/Lexema>)

Rtree fue descontinuado a favor de GiST.

Las búsquedas Like e iLike solo usan B-tree si la búsqueda o el patrón es fijo al inicio, o sea lo usa si es *Campo like 'pat%'* pero no lo usa si es *Campo like '%pat'*.



4. Los índices

Los índices “clusterizados” o “no clusterizados”.

Un índice clusterizado es aquel donde la data esta ordenada o cercanamente ordenada físicamente con las entradas de la data del índice

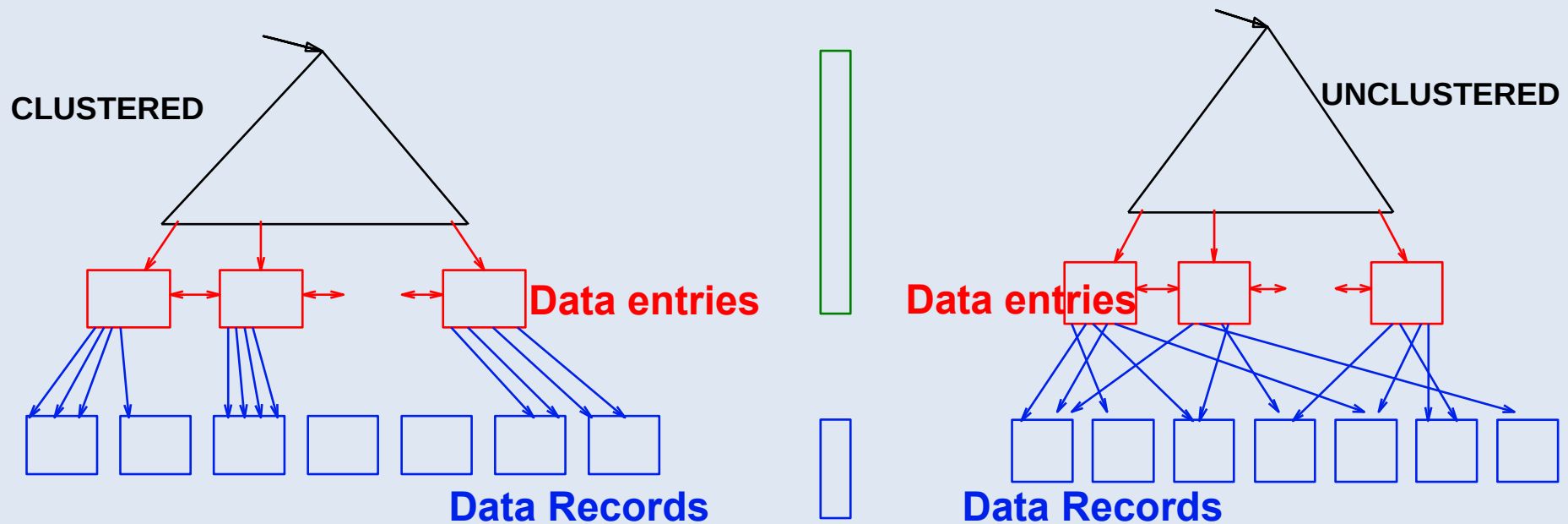
Una tabla solo puede estar clusterizada solo por un índice, no más.

Son altamente apreciados en búsquedas por rangos, pero su costo de mantenimiento es alto debido al reordenamiento que siempre tiene que hacerse a la data.



4. Los índices

Los índices “clusterizados” o “no clusterizados”.





<http://www.eqsoft.net>

Correo: informes@eqsoft.net

Teléfonos: (51)(1)-5645424/997244926/997003957

5. Como se procesa un Query

Una consulta simple:

```
select firstname  
from friend  
where age = 33;
```

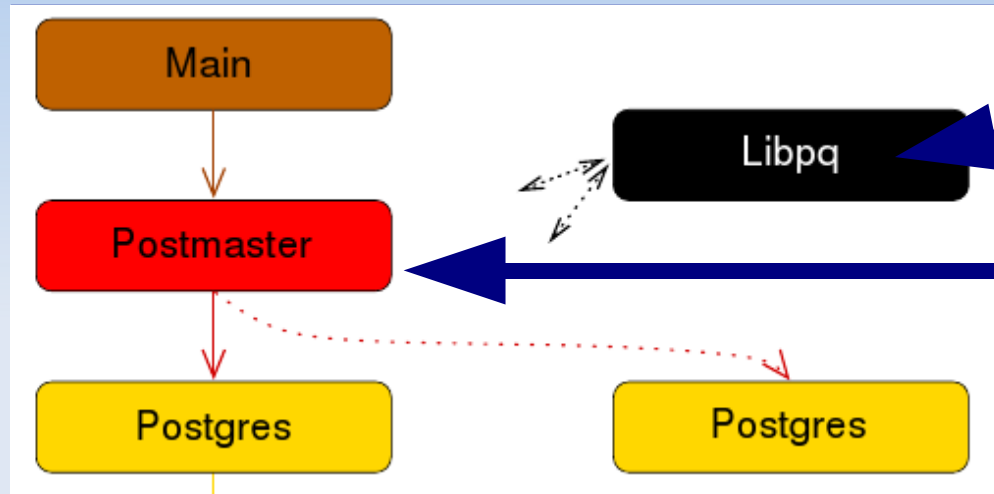
**¿Como hace la base de datos
para interpretar esto?**



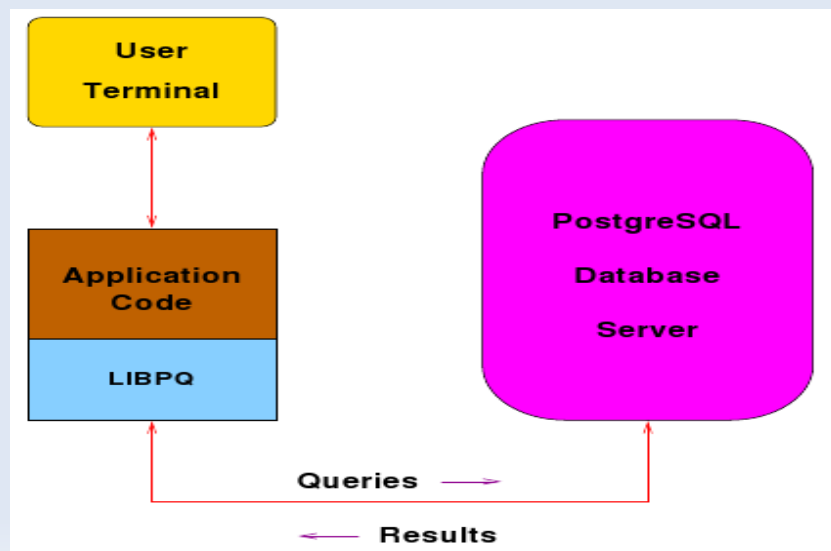
The diagram illustrates the PostgreSQL architecture. At the top, the **Main** process (brown box) connects to the **Postmaster** (red box). The **Postmaster** connects to the **Postgres** (yellow box). A dashed red arrow also points from the **Postmaster** to another **Postgres** box. The **Postgres** box connects to the **Parser** (dark blue box) inside a large cyan box. The **Parser** connects to the **Traffic Cop** (dark blue box), which then connects to the **Rewrite & Generate Paths** (dark blue box), followed by the **Choose Path & Generate Plan** (dark blue box), and finally the **Executor** (dark blue box). The **Traffic Cop** also connects to the **Utility Commands** (dark blue box) via a **utility** connection. The **Utility Commands** box has a feedback loop back to the **Parser**. The **Executor** connects to the **Utilities** (pink box) via a bidirectional connection. The **Utilities** box connects to the **Catalog** (pink box) via a bidirectional connection. The **Catalog** box connects to the **Storage Managers** (pink box) via a bidirectional connection. The **Storage Managers** box connects to the **Access Methods** (pink box) via a bidirectional connection. The **Access Methods** box connects to the **Nodes / Lists** (pink box) via a bidirectional connection. The **Nodes / Lists** box connects to the **Bootstrap** (purple box) at the bottom.



5. Como se procesa un Query



Esta parte es sencilla:
“El cliente postgresql” se comunica con el servicio del “postmaster” para pasarle una cadena de texto con el query.



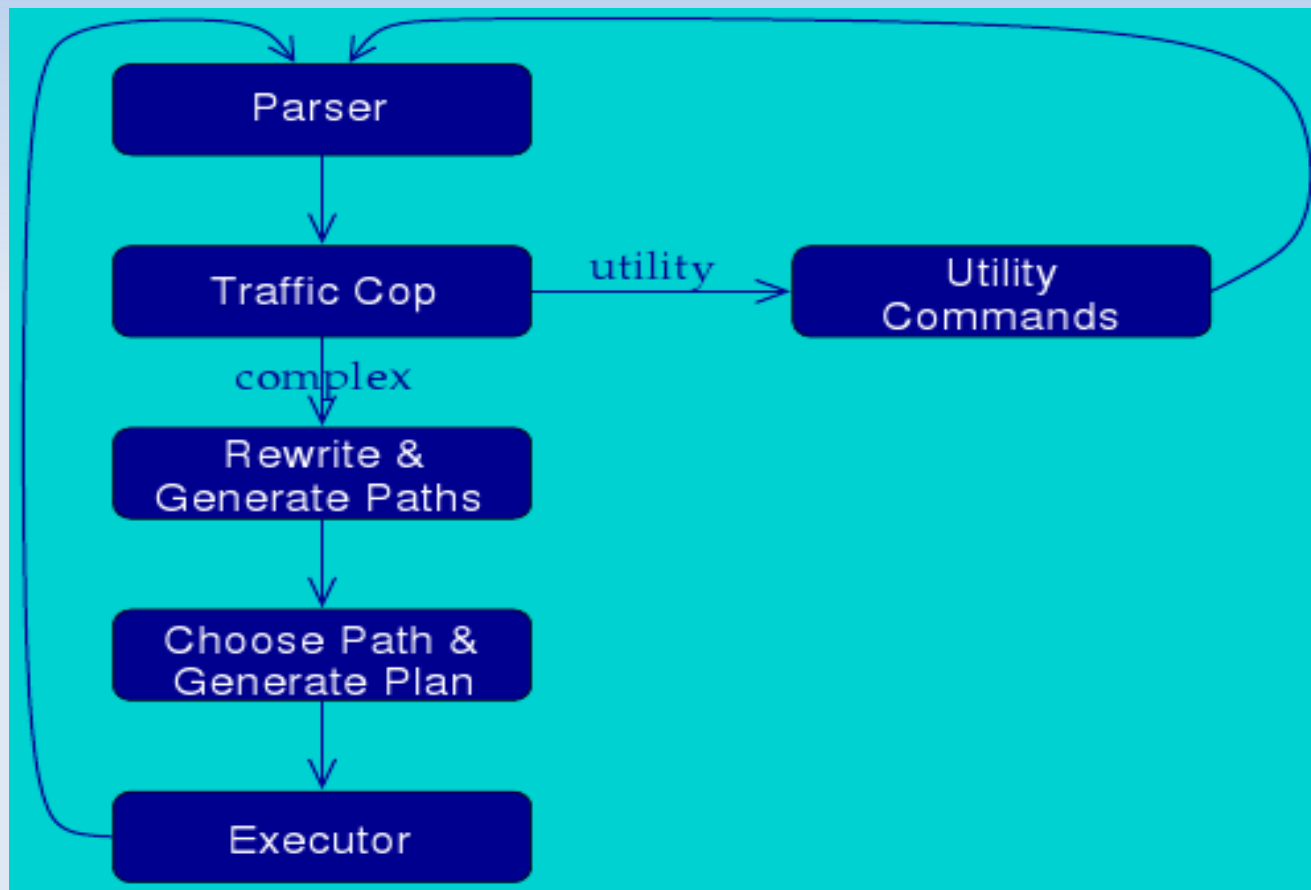
```
FindExec: found "/var/local/postgres/bin/postgres" using argv[0]
DEBUG: connection: host=[local] user=postgres database=test
DEBUG: InitPostgres
DEBUG: StartTransactionCommand
DEBUG: query: SELECT firstname
              FROM friend
              WHERE age = 33;

[ query is processed ]

DEBUG: ProcessQuery
DEBUG: CommitTransactionCommand
DEBUG: proc_exit(0)
DEBUG: shm_exit(0)
DEBUG: exit(0)
```



5. Como se procesa un Query



Aquí es donde empieza la acción



5. Como se procesa un Query

```
DEBUG: parse tree: { QUERY :command 1 :utility <> :resultRelation 0 :into <> :isPortal false :isBinary false :isTemp false :hasAggs false :hasSubLinks false :rtable ({ RTE :relname friend :relid 26912 :subquery <> :alias <> :eref { ATTR :relname friend :attrs ( "firstname" "lastname" "city" "state" "age" ) } :inh true :inFromCl true :checkForRead true :checkForWrite false :checkAsUser 0 ) } :jointree { FROMEXPR :fromlist ({ RANGETBLREF 1 ) } :quals { EXPR :typeOid 16 :opType op :oper { OPER :opno 96 :opid 0 :opresulttype 16 } :args ({ VAR :varno 1 :varattno 5 :vartype 23 :vartypmod -1 :varlevelsup 0 :varnoold 1 :varoattno 5 ) { CONST :consttype 23 :constlen 4 :constbyval true :constisnull false :constvalue 4 [ 33 0 0 0 ] } } ) } :rowMarks () :targetList ({ TARGETENTRY :resdom { RESDOM :resno 1 :restype 1042 :restypmod 19 :resname firstname :reskey 0 :reskeyop 0 :ressortgroupref 0 :resjunk false } :expr { VAR :varno 1 :varattno 1 :vartype 1042 :vartypmod 19 :varlevelsup 0 :varnoold 1 :varoattno 1 } } ) :groupClause <> :havingQual <> :distinctClause <> :sortClause <> :limitOffset <> :limitCount <> :setOperations <> :resultRelations () }
DEBUG: rewritten parse tree:
DEBUG: { QUERY :command 1 :utility <> :resultRelation 0 :into <> :isPortal false :isBinary false :isTemp false :hasAggs false :hasSubLinks false :rtable ({ RTE :relname friend :relid 26912 :subquery <> :alias <> :eref { ATTR :relname friend :attrs ( "firstname" "lastname" "city" "state" "age" ) } :inh true :inFromCl true :checkForRead true :checkForWrite false :checkAsUser 0 ) } :jointree { FROMEXPR :fromlist ({ RANGETBLREF 1 ) } :quals { EXPR :typeOid 16 :opType op :oper { OPER :opno 96 :opid 0 :opresulttype 16 } :args ({ VAR :varno 1 :varattno 5 :vartype 23 :vartypmod -1 :varlevelsup 0 :varnoold 1 :varoattno 5 ) { CONST :consttype 23 :constlen 4 :constbyval true :constisnull false :constvalue 4 [ 33 0 0 0 ] } } ) } :rowMarks () :targetList ({ TARGETENTRY :resdom { RESDOM :resno 1 :restype 1042 :restypmod 19 :resname firstname :reskey 0 :reskeyop 0 :ressortgroupref 0 :resjunk false } :expr { VAR :varno 1 :varattno 1 :vartype 1042 :vartypmod 19 :varlevelsup 0 :varnoold 1 :varoattno 1 } } ) :groupClause <> :havingQual <> :distinctClause <> :sortClause <> :limitOffset <> :limitCount <> :setOperations <> :resultRelations () }
DEBUG: plan: { SEQSCAN :startup_cost 0.00 :total_cost 22.50 :rows 10 :width 12 :qptargetlist ({ TARGETENTRY :resdom { RESDOM :resno 1 :restype 1042 :restypmod 19 :resname firstname :reskey 0 :reskeyop 0 :ressortgroupref 0 :resjunk false } :expr { VAR :varno 1 :varattno 1 :vartype 1042 :vartypmod 19 :varlevelsup 0 :varnoold 1 :varoattno 1 } } ) :qpqual ({ EXPR :typeOid 16 :opType op :oper { OPER :opno 96 :opid 65 :opresulttype 16 } :args ({ VAR :varno 1 :varattno 5 :vartype 23 :vartypmod -1 :varlevelsup 0 :varnoold 1 :varoattno 5 ) { CONST :consttype 23 :constlen 4 :constbyval true :constisnull false :constvalue 4 [ 33 0 0 0 ] } } ) } :lefttree <> :righttree <> :extprm () :locprm () :initplan <> :nprm 0 :scanrelid 1 }
```

El parser transforma el pequeño query en una serie de instrucciones que la base de datos pueda interpretar, por eso es importante escribir queries inteligentes.



5. Como se procesa un Query

Luego pasa por :

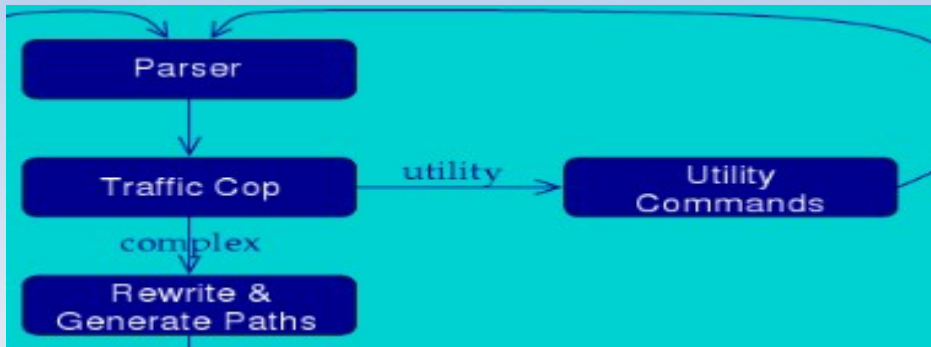
- *Un identificador de reglas de que lo escrito sea sintácticamente entendible, que los dígitos y los números sean reconocibles.*
- *Luego se descompone “palabra” a “palabra” el query para pasar a la estructura que le corresponde según el query, en este caso a la estructura de un “select”, esto se ve así:*

```
simple_select: SELECT opt_distinct target_list
into_clause from_clause where_clause
group_clause having_clause
{
    SelectStmt *n = makeNode(SelectStmt);
    n->distinctClause = $2;
    n->targetList = $3;
    n->istemp = (bool) ((Value *) lfirst($4))>val.ival;
    n->into = (char *) lnext($4);
    n->fromClause = $5;
    n->whereClause = $6;
    n->groupClause = $7;
    n->havingClause = $8;
    $$ = (Node *)n;
}
```

```
typedef struct SelectStmt
{
    NodeTag      type;
    /*
     * These fields are used only in "leaf" SelectStmts.
     */
    List          *distinctClause; /* NULL, list of DISTINCT ON exprs, or
     * lcons(NIL,NIL) for all (SELECT
     * DISTINCT) */
    char          *into;           /* name of table (for select into table) */
    bool          istemp;          /* into is a temp table? */
    List          *targetList;     /* the target list (of ResTarget) */
    List          *fromClause;     /* the FROM clause */
    Node          *whereClause;    /* WHERE qualification */
    List          *groupClause;    /* GROUP BY clauses */
    Node          *havingClause;   /* HAVING conditional-expression */
    /*
     * These fields are used in both "leaf" SelectStmts and upper-level
     * SelectStmts. portalname/binary may only be set at the top level.
     */
    List          *sortClause;     /* sort clause (a list of SortGroupBy's) */
    char          *portalname;     /* the portal (cursor) to create */
    bool          binary;          /* a binary (internal) portal? */
    Node          *limitOffset;    /* # of result tuples to skip */
    Node          *limitCount;     /* # of result tuples to return */
    List          *forUpdate;      /* FOR UPDATE clause */
    /*
     * These fields are used only in upper-level SelectStmts.
     */
    SetOperation op;               /* type of set op */
    bool          all;             /* ALL specified? */
    struct SelectStmt *larg;       /* left child */
    struct SelectStmt *rarg;       /* right child */
    /* Eventually add fields for CORRESPONDING spec here */
} SelectStmt;
```



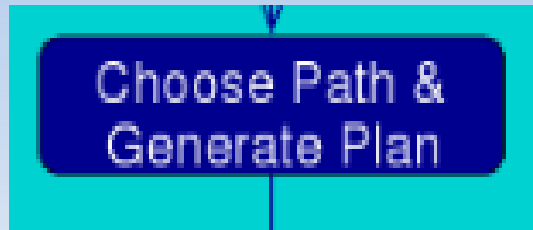
5. Como se procesa un Query



EL Traffic Cop contiene al controlador principal del proceso del PostgreSQL, además se encarga de las comunicaciones entre el Parser, Optimizer, Executor y /commands functions. Los querys complejos pasan al Rewriter (select, insert, etc.), lo que no, se pasa al Utility Commands, generalmente querys simples (alter, create, vacuum, etc.)



5. Como se procesa un Query



El “planner” es el encargado de generar el “plan de ejecución”, esto es estimar la mejor vía para resolver el query, maneja mediante formulas matemáticas avanzadas la forma de búsqueda de datos y la forma de resolver las relaciones entre tablas.


Luego que el planner a calculado el costo de todas las posibles vías de obtener la data escoge cual es el mejor y se lo pasa al “Executor”.



5. Como se procesa un Query

Métodos de búsqueda

Heap



D	D	D	D	D	D	D	D	D	D	D	D
A	A	A	A	A	A	A	A	A	A	A	A
T	T	T	T	T	T	T	T	T	T	T	T
A	A	A	A	A	A	A	A	A	A	A	A

Secuencial

Index

<	Key	=	>
---	-----	---	---

<	Key	=	>
---	-----	---	---

<	Key	=	>
---	-----	---	---

Heap

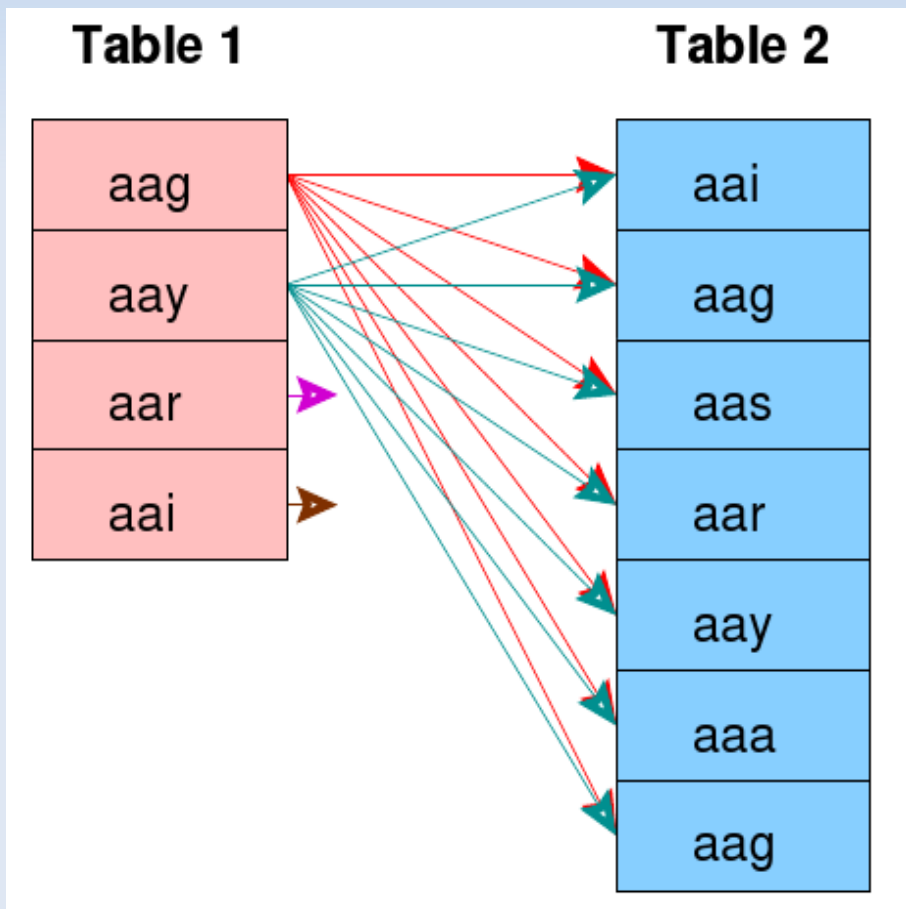
D	D	D	D	D	D	D	D	D	D	D	D
A	A	A	A	A	A	A	A	A	A	A	A
T	T	T	T	T	T	T	T	T	T	T	T
A	A	A	A	A	A	A	A	A	A	A	A

Indexada



5. Como se procesa un Query

Métodos para relacionar tablas



Nested loop join

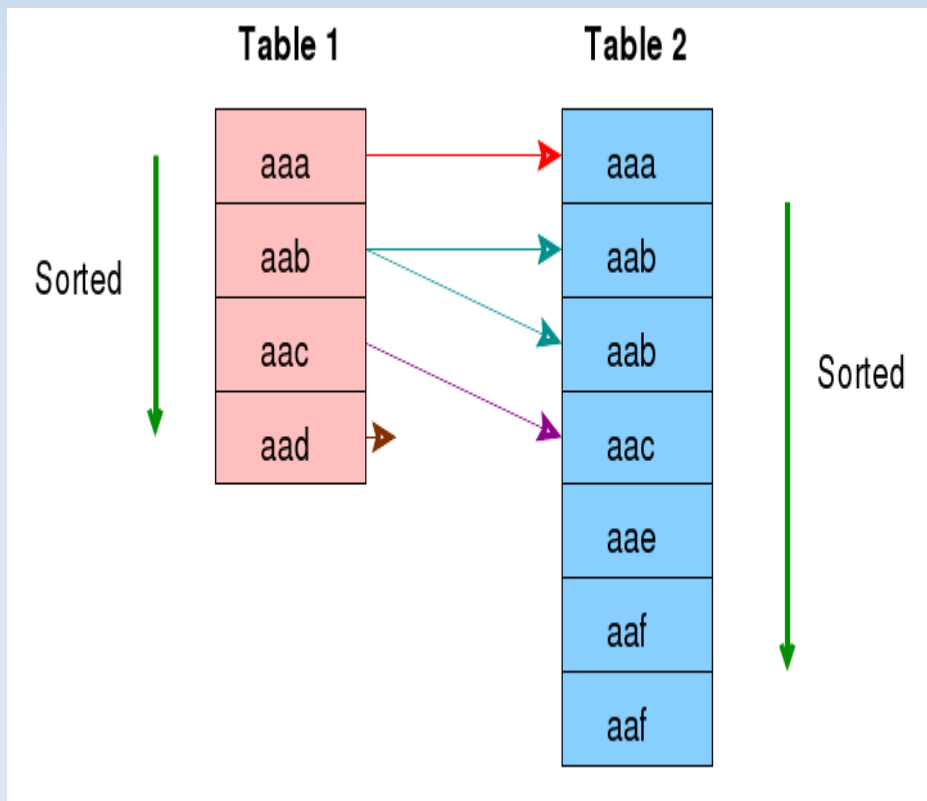
Consume mas recursos de memoria pero la cantidad de búsquedas a realizar es menor.

http://en.wikipedia.org/wiki/Nested_loop_join



5. Como se procesa un Query

Métodos para relacionar tablas



Merge join

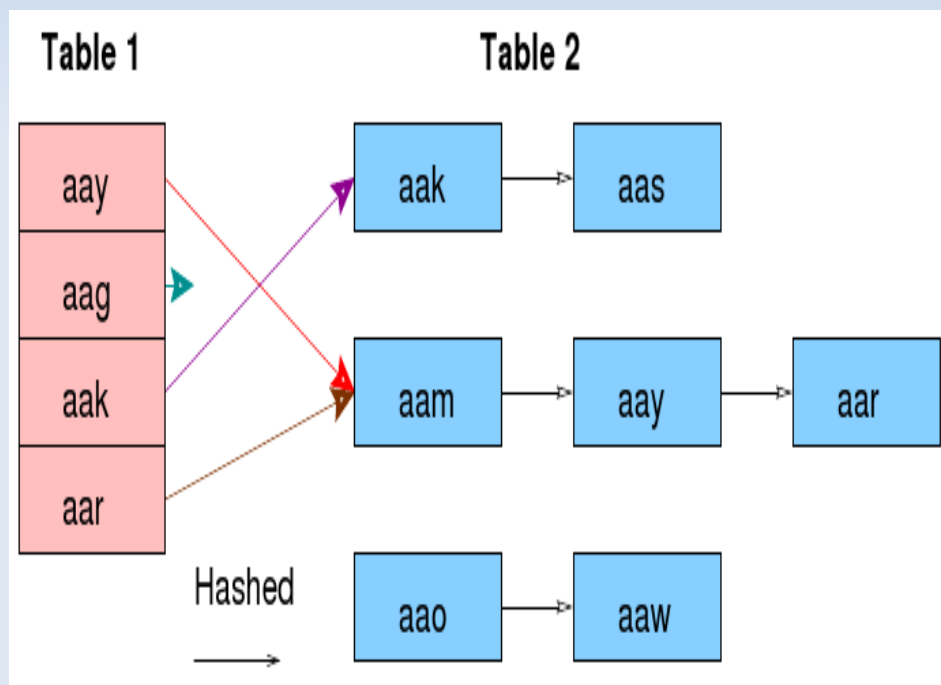
Requiere que la data este ordenada para ubicar las relaciones, el costo esta justamente en mantener la data ordenada.

http://en.wikipedia.org/wiki/Merge_join



5. Como se procesa un Query

Métodos para relacionar tablas



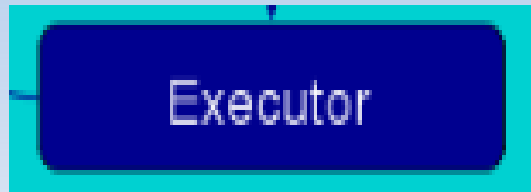
Hash join

Aparentemente sería la forma mas rápida de acceder a data gracias a la creación de tablas indexadas, pero limitada a una búsqueda de igualdad.

http://en.wikipedia.org/wiki/Hash_join



5. Como se procesa un Query



El “Executor” toma el plan de ejecución que el “planner” le entrega e inicia el procesamiento, ejecuta un “plan tree”.

Este “plan tree” tiene varios nodos de ejecución que se van ejecutando uno a uno y de cada uno de ellos se obtiene un set de datos (tuplas).

Los nodos tienen subnodos y otros a su vez otros subnodos, tantos como sea necesario.



5. Como se procesa un Query

```
(2 3 ): rows=575 width=76
path list:
HashJoin rows=575 cost=3.57..41.90
  clauses=(salesorder.part_id = part.part_id)
    SeqScan(2) rows=575 cost=0.00..13.75
    SeqScan(3) rows=126 cost=0.00..3.26
Nestloop rows=575 cost=0.00..1178.70
  SeqScan(2) rows=575 cost=0.00..13.75
  IdxScan(3) rows=126 cost=0.00..2.01
Nestloop rows=575 cost=0.00..1210.28
  pathkeys=((salesorder.customer_id, customer.customer_id) )
  IdxScan(2) rows=575 cost=0.00..45.33
    pathkeys=((salesorder.customer_id, customer.customer_id) )
    IdxScan(3) rows=126 cost=0.00..2.01

cheapest startup path:
Nestloop rows=575 cost=0.00..1178.70
  SeqScan(2) rows=575 cost=0.00..13.75
  IdxScan(3) rows=126 cost=0.00..2.01

cheapest total path:
HashJoin rows=575 cost=3.57..41.90
  clauses=(salesorder.part_id = part.part_id)
    SeqScan(2) rows=575 cost=0.00..13.75
    SeqScan(3) rows=126 cost=0.00..3.26
```



6. Concurrencia : Transacciones y Bloqueos

El control de concurrencia es el que asegura que muchos usuarios puedan acceder a la data al mismo tiempo.

Sin embargo al mismo tiempo se producen muchas operaciones de R/W, estas operaciones se conocen como transacciones.

Ninguna transacción debe ver el resultado de otras transacciones inconclusas, si esto no fuera así estaríamos leyendo datos inconsistentes.

Una transacción puede incluir dentro de si muchas operaciones en la base de datos.



6. Concurrencia : Transacciones y Bloqueos

Las transacciones deben cumplir el criterio ACID.

Consistency: la transacción solo termina si la data es consistente.

Isolation: la transacción es independiente de otras transacciones.

Atomicity: todas las acciones en la transacción se cumplen o no se cumple ninguna.

Durability: cuando la transacción termina el resultado de la misma es perdurable.



6. Concurrencia : Transacciones y Bloqueos

- PostgreSQL implementa el modelo MVCC (Multiversion Concurrency Control) desde la 8.3.
- Bajo MVCC las transacciones ven una imagen de la data al momento de iniciarla (para ello la data se versiona con un timestamp), esto protege la transacción de inconsistencia de data cuando llegan 2 operaciones de R/W sobre la misma.
- En simple el MVCC nunca modifica ó elimina la data, nuevas filas de información de van añadiendo conforme se crea o actualiza la data y se marca la anterior como “no visible”, cuando se desea eliminar un dato igualmente se añade una fila de data y se marca como “no visible” al mismo tiempo.
- La data nunca es “visible” por otros usuarios hasta que no sea “commiteada”.
- La principal ventaja es que las operaciones de R nunca bloquean a las de W, y viceversa, podemos obtener backups en caliente sin bloquear la db.
- Como desventaja: consumimos mas disco duro.



6. Concurrencia : Transacciones y Bloqueos

Tipo	MVCC			DATA	
	timestamp inicio	timestamp fin	visible	id	nombre
W	11:59:59	11:59:59	Y	01	Juan Perez
W	11:59:59	11:59:59	N	01	Juan Perez
W	12:00:05	12:00:05	Y	01	Juan Perez Paredes
W	11:59:59	11:59:59	N	01	Juan Perez
W	12:00:05	12:00:05	N	01	Juan Perez Paredes
W	12:10:01	12:10:01	N	01	Juan Perez Paredes

← delete

Tipo	MVCC			DATA	
	timestamp inicio	timestamp fin	visible	id	nombre
W	11:59:59	11:59:59	N	01	Juan Perez
R	12:01:01	12:01:01	Y	01	Juan Perez
W	12:01:01	12:01:01	Y	01	Juan Perez Paredes
W	11:59:59	11:59:59	N	01	Juan Perez
W	12:01:01	12:01:01	Y	01	Juan Perez Paredes
W	12:10:02			01	Luis Perez Paredes
R	12:10:04	12:10:04	Y	01	Juan Perez Paredes
W	11:59:59	11:59:59	N	01	Juan Perez
W	12:01:01	12:01:01	N	01	Juan Perez Paredes
W	12:10:02	12:10:07	Y	01	Luis Perez Paredes
R	12:10:04	12:10:04	N	01	Juan Perez Paredes
R	12:10:06	12:10:06	Y	01	Luis Perez Paredes

← supongamos que tomará 5 segundos en hacer commit

← se lee la transacción de las 12:01:01

← se lee la transacción de las 12:10:02



6. Concurrencia : Transacciones y Bloqueos

Tipo	MVCC			DATA	
	timestamp inicio	timestamp fin	visible	id	nombre
W	11:59:59	11:59:59	N	01	juan peres
R	12:01:01	12:01:01	Y	01	juan perez paredes
W	11:59:59	11:59:59	N	01	juan peres
R	12:01:01	12:01:01	Y	01	juan perez paredes
W	12:10:02			01	luis perez paredéz
W	12:10:03				pedro perez
W	11:59:59	11:59:59	Y	01	juan peres
R	12:01:01	12:01:01	N	01	juan perez paredes
W	12:10:02			01	luis perez paredéz
W	12:10:03				pedro perez
R	12:10:04	12:10:04	Y	01	juan perez paredes
W	11:59:59	11:59:59	N	01	juan peres
R	12:01:01	12:01:01	N	01	juan perez paredes
W	12:10:02	12:10:07	Y	01	luis perez paredéz
R	12:10:04	12:10:04	N	01	juan perez paredes
R	12:10:08	12:10:08	Y	01	luis perez paredéz
W	12:10:08			01	pedro perez

<-- demora 5 segundos

<-- se aborta y reiniciará después

<-- se lee la transacción de las 12:01:01

<-- se lee la transacción de las 12:01:01

<-- se lee la transacción de las 12:10:02

<-- ya esta apto para procesar



6. Concurrencia : Transacciones y Bloqueos

	Tipo	MVCC			DATA		
		timestamp inicio	timestamp fin	visible	id	nombre	
Opción 1	W	11:59:59	11:59:59	N	01	juan peres	
	R	12:01:01	12:01:01	Y	01	juan perez paredes	
	R	12:10:02			01	juan perez paredes	<-- select masivo con joins demora 10 segundos
	W	12:10:03			01	pedro perez	<-- se aborta y reiniciará después
	W	11:59:59	11:59:59	N	01	juan peres	
	R	12:01:01	12:01:01	N	01	juan perez paredes	
	R	12:10:02	12:10:12	Y	01	juan perez paredes	
	W	12:10:13			01	pedro perez	<-- ya esta apto para procesar
Opción 2							
	W	11:59:59	11:59:59	N	01	juan peres	
	R	12:01:01	12:01:01	N	01	juan perez paredes	
	R	12:10:02			01	juan perez paredes	<-- se lee la transacción de las 12:01:01
	RW	12:10:03	12:10:03	Y	01	pedro perez	<-- se copia el dato



6. Concurrencia : Transacciones y Bloqueos

Cuando 2 transacciones trabajan sobre el mismo objeto y al menos uno de ellos incluye operaciones de escritura entonces se produce un “conflicto”.

Cuando 2 transacciones hacen exactamente lo mismo puede que sean “serializadas” para optimizar el acceso a la data.

$$\begin{array}{cc} R(A) \ W(A) & R(B) \ W(B) \\ R(A) \ W(A) & R(B) \ W(B) \\ \equiv & \\ R(A) \ W(A) \ R(B) \ W(B) & \\ & R(A) \ W(A) \ R(B) \ W(B) \end{array}$$

Antes de que una transacción pueda ejecutar un R/W sobre un objeto en la db debe obtener un “bloqueo”.



6. Concurrencia : Transacciones y Bloqueos

Este bloqueo puede ser Compartido (Share) o Exclusivo (Exclusive), estos son administrados por un “Lock Manager”.

Existen varios tipos de bloqueos :

Mode	Used
Access Share Lock	SELECT
Row Share Lock	SELECT FOR UPDATE
Row Exclusive Lock	INSERT, UPDATE, DELETE
Share Lock	CREATE INDEX
Share Row Exclusive Lock	EXCLUSIVE MODE but allows ROW SHARE LOCK
Exclusive Lock	Blocks ROW SHARE LOCK and SELECT...FOR UPDATE
Access Exclusive Lock	ALTER TABLE, DROP TABLE, VACUUM, and unqualified LOCK TABLE



6. Concurrencia : Transacciones y Bloqueos

Si un bloqueo toma mucho tiempo en ejecutarse entonces se produce un Deadlock, el sistema lo muestra como un Timeout.

Un bloqueo puede darse a una tupla, una página o una tabla completa.

Existe una tabla de locks, antes de ejecutar uno nuevo se consulta esta tabla.



<http://www.eqsoft.net>

Correo: informes@eqsoft.net

Teléfonos: (51)(1)-5645424/997244926/997003957

Agradecimientos:

- ***Alvaro Herrera, desarrollador del proyecto PostgreSQL por las correcciones a este trabajo.***

Para leer más, todas las referencias bibliográficas bajo las que se hizo estas diapositivas están aquí:

<http://tinyurl.com/y9who7m>

Gracias por su tiempo.