# Titulación: Grado en Ingeniería Informática, Ingeniería en

**Sistemas de Información e InfoAde** **Curso: 2024-2025. Convocatoria Ordinaria de Mayo** **Asignatura: Bases de Datos Avanzadas – Laboratorio**

Practica 1: Arquitectura PostgreSQL y

almacenamiento físico

**ALUMNO 1:**

## Nombre y Apellidos:

Álvaro Paniagua Cortijo

**DNI:**

06593675N

**ALUMNO 2:**

## Nombre y Apellidos:

Cristina Martínez Toledo

**DNI:**

09109126E

## Fecha:

04/02/2025

**Profesor Responsable:** Óscar Gutierrez

Mediante la entrega de este fichero los alumnos aseguran que cumplen con la normativa de autoría de trabajos de la Universidad de Alcalá, y declaran éste como un trabajo original y propio.

En caso de ser detectada copia, se calificará la asignatura como Suspensa – Cero.

Es obligatorio proporcionar una explicación a lo que está ocurriendo en PostgreSQL cuando así se indica en la cuestión. No solo vale poner un pantallazo. La ausencia de una explicación hará que sea invalidada esa cuestión.

# Plazos

Trabajo de Laboratorio: semana 27 enero (GISI), 3 febrero, 10 febrero, 17 febrero, 24 de febrero y 3 marzo (GII/INFOADE)

Entrega de práctica: día 9 de marzo. Aula Virtual

Documento a entregar: un fichero con formato ZIP con las respuestas a las cuestiones planteadas en este fichero pdf (prohibido modificar el fichero o entregar otro documento diferente con respuestas y/o imágenes), así como los ficheros de log de postgresql relacionados con la resolución de la práctica y el fichero con el código de generación de los datos de la cuestión 1. El fichero se deberá llamar: **DNIdelosAlumnos\_PL1.zip**

**AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.**

# Introducción

En esta primera práctica se introduce el sistema gestor de bases de datos PostgreSQL (17.2 la última). Está compuesto básicamente de un motor servidor y de una serie de clientes que acceden al servidor y de otras herramientas externas. En esta primera práctica se entrará a fondo en la arquitectura de PostgreSQL, sobre todo en el almacenamiento físico de los datos y del acceso a los mismos. Antes de comenzar es obligatorio configurar lo que se comenta en la cuestión. Hay que resolver la práctica con consultas SQL.

Cuestión 0. Configurar el fichero de Error Reporting and Logging de PostgreSQL para que aparezcan recogidas las sentencias SQL DDL (Lenguaje de Definición de Datos) + DML (Lenguaje de Manipulación de Datos) generadas en dicho fichero. No se pide activar todas las sentencias. No activar la duración de la consulta. También se debe de configurar el log para que en el comienzo de la línea de registro de la información del log (“line prefix”) aparezcan vuestros DNI’s y el nombre del host con su puerto. ¿Cómo se ha realizado la configuración?

se ha modificado la linea:

log\_line\_prefix = '%m [%p] 09109126E 06593675N %r '

## Organización de Archivos en PostgreSQL

Cuestión 1. Crear una nueva Base de Datos que se llame **PL1**. Después crear una tabla

**productos** con los siguientes campos:

* producto\_id: que debe ser el identificador del producto comenzando por 1.
* nombre: guarda el nombre del producto.
* stock: guarda la cantidad de cada producto en el almacén.
* precio: guarda el precio de cada producto.

Crear un programa que permita generar 25 millones de registros en un fichero de texto que pueda ser cargado en la tabla (preferiblemente en Python) con las siguientes propiedades para los siguientes campos, cuyos valores se deben generar aleatoriamente.

* stock: deben ser valores aleatorios entre 0 y 20000.
* precio: deben ser valores aleatorios entre 10 € y 5000 €.

Cargar los datos en la tabla y localizar los ficheros relacionados con la tabla. ¿cómo se localizan? ¿Cuánto ocupan? ¿por qué?

Cuestión 2. Calcular teóricamente el tamaño en bloques que ocupa la relación **productos** tal y como se realiza en clase de teoría. ¿Concuerda con el tamaño en bloques que nos proporciona PostgreSQL? ¿Cuál es el factor de bloque medio real de la tabla **productos**? ¿Por qué? Realizar una consulta SQL que obtenga ese valor y comparar con el factor de bloque teórico.

Tamaño de variables:

int = 4Bytes.

varchar(18) = caracteres ascii(1Byte por caracter). cabecera de registro = 23Bytes

longitud del 1 registro = 4\*3+18+23= 53 Bytes

con show block\_size, se obtiene una cantidad de 8192 Bytes de longitud de bloque

el grado de ocupación es de 100% por defecto, por lo que se ocuparan la longitud de bloque como se usan 24 bytes de control por bloque, de los 8192 B , se usan 8168 B para datos (Butil). el factor de bloque será: 8168/53 (redondea hacia abajo) = 154 registros

el Nº de bloques totales será: 25M/154 (redondeo hacia arriba) = 162.338 bloques

Mediante consultas de PostgresSQL, se obtinen los siguientes datos:

Con esta consulta:

SELECT relname AS table\_name, relpages AS bloques FROM pg\_class WHERE relname='tabla25M';

se obtiene un total de 183872 bloques.

Con la consulta:

SELECT 8192 / avg(pg\_column\_size(t)) AS block\_factor FROM tabla25M t;

se obtiene un factor de bloque de 146

Se observó, que si se calculaban el numero de registros sin añadir las cabeceras de registro, los cálculos mostraban que el numero de bloques era aproximadamente la mitad de bloques actuales, esto se debía a que la cabecera es casi del mismo tamaño que la longitud de registro. Esto tambien se observa en el factor de bloque el cual es mayor que el que usa postgresSQL.

Aun así el número total de bloques y el factor de bloque entre PostrgresSQL y los cálculos no coinciden del todo, por el uso por parte de PostgresSQL de mas datos en los registros como punteros u otros datos.

Cuestión 3. Realizar una consulta que muestre los productos que tengan un precio de 3000 €. ¿Cuántas tuplas se obtienen y cuántos bloques se leen del disco por Postgres?

¿Por qué? Comparar con los resultados obtenidos al aplicar el método visto en teoría. (usar funciones de “ The Cumulative Statistics System”)

Como precio se trata de un campo no clave, el valor V(precio) = 4990, que son todos los valores distintos que puede tomar el campo. 5000 (máx)-10(min) = 4990.

Teniendo en cuanta la equiprobabilidad de todos los valores del campo precio. El numero de registros será de 25.000.000/4990 = 5010 registro de media.

Mientras que como se trata de un campo no clave, el cual no está ordenado se deberán leer todos los 25.000.00 de registros para comprobar todos y cada uno de ellos de manera secuencial.

Mediante la consulta:

SELECT pg\_stat\_reset();

SELECT \* FROM tabla25M WHERE precio = 3000;

SELECT heap\_blks\_read, heap\_blks\_hit FROM pg\_statio\_user\_tables WHERE relname =’tabla25M';

se obtiene un total de 4965 registros con precio = 3000

De todos los bloques se han leido:

183872bloques en memoria

0 bloques de caché

Cuestión 4. Volver a realizar la consulta de la cuestión 3 de nuevo. ¿Cuántas tuplas se obtienen y cuántos bloques se leen del disco por Postgres ahora? ¿Por qué? Comparar con la cuestión anterior

Si se vuelve a ejecutar la misma consulta:

SELECT \* FROM tabla25M WHERE precio = 3000;

SELECT heap\_blks\_read, heap\_blks\_hit FROM pg\_statio\_user\_tables WHERE relname = ‘tabla25M';

se obtiene el mismo numero de registros con precio 3000

De todos los bloques se han leido:

183392 bloques en memoria

480 bloques de caché

Si los sumas te salen 183872 bloques que coincide con la búsqueda secuencial y el numero de bloques del apartado anterior de bloques leidos

cada vez que se ejecute la misma consulta el numero de bloques leídos desde el caché aumenta y el de memoria disminuye. Pero el numero de bloques totales sigue siendo el mismo.

Si se reinicia postgresSQL y se ejecuta otra vez la consulta, el numero de registros leídos desde caché varia.

(explicar por qué varía)

Cuestión 5. Crear una tabla **productos2** cuyas tuplas estén ordenadas físicamente por el campo precio de menor a mayor y que tenga la misma información. Cargar el mismo fichero de datos creado en la cuestión1. Indicar el proceso de generación de dicha tabla ordenada. ¿Cuántos bloques ocupa la tabla ahora? ¿Hay algún cambio? ¿Por qué?

se ha creado la tabla productos2 mediante el siguiente código:

create table if not exists productos2( id\_producto1 int,

nombre1 varchar(18), stock1 int,

precio1 int,

constraint ID1 primary key(id\_producto1)

);

y la consulta para ordenar los datos de menor a mayor y cargarlos en la tabla: insert into productos2(id\_producto1, nombre1, stock1, precio1)

select \* from tabla25M order by precio asc;

El número de bloques es de 183824, al ser la misma cantidad de información pero con un orden distinto, por lo que hay bloques que contendrán alguna tupla mas que otros, por lo que hay menos bloques en la tabla que en la original.

Cuestión 6. Repetir la cuestión 3 sobre la tabla **productos2** y comparar los resultados obtenidos indicando las conclusiones obtenidas. Relacionarlo con lo visto en teoría.

Como precio se trata de un campo no clave, el valor V(precio) = 4990, que son todos los valores distintos que puede tomar el campo. 5000 (máx)-10(min) = 4990.

Teniendo en cuanta la equiprobabilidad de todos los valores del campo precio. El numero de registros será de 25.000.000/4990 = 5010 registro de media.

Mientras que como se trata de un campo no clave, el cual está ordenado, se puede hacer busuqeqda secuencial o busqueda binaria:

Secuencial: leer los 182824 bloques

Binaria: log2(182824)+ (nºregistros a recuperar/factor de bloque) = 18 + (5010/154)

BB= 51 bloques

Mediante la consulta:

SELECT pg\_stat\_reset();

SELECT \* FROM tabla25M WHERE precio = 3000;

SELECT heap\_blks\_read, heap\_blks\_hit FROM pg\_statio\_user\_tables WHERE relname = 'productos2';

se obtiene un total de 4965 registros con precio = 3000

De todos los bloques se han leido:

183824 bloques en memoria

0 bloques de caché

Cuestión 7. Borrar 4000000 tuplas de la tabla **productos** de manera aleatoria usando el valor del campo producto\_id. ¿Qué es lo que ocurre físicamente en la base de datos?

¿Se observa algún cambio en el tamaño de la tabla y estructuras asociadas a ella? ¿Por qué? Adjuntar el código de borrado.

Borra las tuplas de datos de los bloques, pero el espacio que ocupan se mantienen porque los registros y los bloques no se reorganizan automáticamente dentro de la base de datos, originando huecos dentro de los bloques o también conocidos como tuplas muertas.

Estas tuplas "muertas", son utilizadas para la futura inserción de datos dentro de la bbdd, lo que provoca que no aumente el tamaño hasta que se llenen los datos borrados.

Debido a esto, el tamaño de la tabla no varia y se mantienen los 183872 bloques de datos. Se han borrado gracias a este código:

DELETE FROM tabla25M

WHERE id\_producto

IN (SELECT id\_producto FROM tabla25M ORDER BY random() LIMIT 4000000);

(Control de versiones(MVCC) filas eliminadas no visibles)

Cuestión 8. Insertar un nuevo producto en la tabla **productos**. ¿dónde se inserta físicamente la nueva tupla? ¿Por qué? (Mirar apartado System Columns)

Mediante la consulta:

insert into tabla25M(id\_producto,nombre,stock,precio)

values(25000001,'producto\_25000001',250,20);

insertamos el nuevo dato, el cual se inserta en el primer espacio libre de la tabla, el cual con la consulta:

SELECT ctid, \* FROM tabla25M WHERE id\_producto = 25000001;

esta, nos indica que se encuentra en (183823,60) , es decir, se ha colocado en el bloque numero 183823 y en la posición 60 dentro del bloque

Esto se debe a que la tupla en la posición (183823,60) que había sido borrada en el apartado anterior, ha sido sobrescrita por la nueva.

Cuestión 9. En la situación anterior, ¿Qué operaciones se pueden aplicar a la base de datos **PL1** para optimizar el rendimiento de esta? Aplicarlas de tal manera que se recupere el mayor espacio posible. Comentar cuál es el resultado final y qué es lo que ocurre físicamente.

Con el uso de VACUUM FULL, se reajusta la tabla, liberando el espacio ocupado por las tuplas muertas.

Por ello si se hace la consulta:

SELECT relname AS table\_name, relpages AS bloques FROM pg\_class WHERE relname='tabla25M';

se obtiene un total de bloques 154412

que es menor que los 183872 anteriores

físicamente se han reorganizado y reescrito todas y cada una de las tuplas de la tabla. Con este método, el siguiente dato a insertar, se localiza al final de las tuplas anteriores.

Cuestión 10. Crear una nueva tabla denominada **productos3** con los mismos campos que la cuestión 1 y que esté particionada por el campo precio definido en rangos de 500€. Insertar los datos del fichero de datos generado en la cuestión 1. Explicar el proceso seguido y comentar qué es lo que ha ocurrido físicamente en la base de datos. ¿Cuándo será útil el particionamiento? ¿Cuántos bloques ocupa cada una de las particiones?¿Por qué? Comparar con el número bloques que se obtendría teóricamente utilizando el procedimiento visto en teoría.

Cuestión 11. Repetir la cuestión 3 sobre la tabla **productos3** y comparar los resultados obtenidos con lo visto anteriormente en las tablas **productos** y **productos2** obteniendo conclusiones sobre el método de partición.

sefsf

## Indexación de PostgreSQL

PostgreSQL soporta indexación definida por el usuario para ayudar a acelerar ciertas consultas. Entre otros tipos de índices soporta árboles y hash. En este apartado se va a trabajar sobre ambos tipos de índices, pudiendo observar cómo se organizan internamente y su funcionamiento.

Cuestión 12. Borrar todas las tablas **productos, productos2** y **productos3**. Crear una nueva tabla que se llama **productos** como en la cuestión 1 y que tenga cargados todos los datos del fichero de texto generado.

fsef

Cuestión 13. Crear un índice de tipo árbol para producto\_id. ¿Dónde se almacena físicamente ese índice? ¿Qué tamaño tiene? ¿Cuántos bloques tiene? ¿Cuántos niveles tiene? ¿Cuántos bloques tiene por nivel? ¿Cuántas tuplas tiene un bloque de cada nivel? Indicar el procedimiento seguido e incluir el código SQL utilizado.

fsef

Cuestión 14. Determinar el tamaño de bloques y el número de niveles que teóricamente tendría de acuerdo con lo visto en teoría. Comparar los resultados obtenidos teóricamente con los resultados obtenidos en la cuestión 13.

sefsf

Cuestión 15. Crear un índice de tipo hash para el campo producto\_id. ¿Dónde se almacena físicamente ese índice? ¿Qué tamaño tiene? ¿Cuántos bloques tiene?

¿Cuántos cajones tiene? ¿Cuántas tuplas tiene de media un cajón? Indicar el procedimiento seguido e incluir el código SQL utilizado.

sefesf

Cuestión 16. Determinar el tamaño de bloques y el número de cajones que teóricamente tendría de acuerdo con lo visto en teoría. Comparar los resultados obtenidos teóricamente con los resultados obtenidos en la cuestión 15.

sefsf

Cuestión 17. Crear un índice de tipo árbol para el campo precio. ¿Dónde se almacena físicamente ese índice? ¿Qué tamaño tiene? ¿Cuántos bloques tiene? ¿Cuántos niveles tiene? ¿Cuántos bloques tiene por nivel? ¿Cuántas tuplas tiene un bloque de cada nivel? Indicar el procedimiento seguido e incluir el código SQL utilizado.

sefse

Cuestión 18. Determinar el tamaño de bloques y número de niveles que teóricamente tendría de acuerdo con lo visto en teoría. Comparar los resultados obtenidos teóricamente con los resultados obtenidos en la cuestión 17.

sefsef

Cuestión 19. Crear un índice de tipo hash para el campo precio. ¿Dónde se almacena físicamente ese índice? ¿Qué tamaño tiene? ¿Cuántos bloques tiene? ¿Cuántos cajones tiene? ¿Cuántas tuplas tiene de media un cajón? Indicar el procedimiento seguido e incluir el código SQL utilizado.

sefsef

Cuestión 20. Determinar el tamaño de bloques y número de cajones que teóricamente tendría de acuerdo con lo visto en teoría. Comparar los resultados obtenidos teóricamente con los resultados obtenidos en la cuestión 17.

sefse

Cuestión 21. ¿Qué conclusiones se puede obtener de la gestión y organización de PostgreSQL sobre los dos índices árbol y hash que se han creado y han sido analizados?

¿Por qué? Comparar con lo visto en teoría.

sefsef

## Monitorización de la actividad de la base de datos

En este último apartado se mostrará el acceso a los datos con una serie de consultas sobre la tabla original. En este apartado se pretende mostrar cómo es el acceso a los datos para diferentes tipos de consultas.

PostgreSQL suministra varias vistas estadísticas que se pueden usar para monitorizar los bloques leídos (tipo statio de la sección The Cumulative Statistics System) de cada una de las estructuras creadas en la base de datos. En este apartado se deben usar esas vistas y está prohibido el uso de otro comando para este fin (Table 28.2).

Para ello, borrar todas las tablas creadas y volver a crear la tabla **productos** como en la cuestión 1. Cargar los datos que se encuentran originalmente en el fichero generado en la cuestión 1.

Cuestión 22. Crear un índice primario tipo árbol sobre el campo precio. También crear un índice hash sobre el campo producto\_id y otro sobre precio. ¿Cuál ha sido el proceso seguido para crear cada tipo de índice? Incluir el código SQL utilizado para ello.

sefse

Cuestión 23. Para cada una de las consultas que se muestran a continuación, ¿Qué información se puede obtener de los datos monitorizados por la base de datos al realizar la consulta? Comentar cómo se ha realizado la resolución de la consulta.

¿Cuántos bloques se han leído de cada estructura? ¿Por qué? Comparar con lo visto en teoría. Importante, reinicializar los datos recolectados de la actividad de la base de datos antes de lanzar cada consulta. Se recuerda que solo se pueden usar vistas sobre las estadísticas de la base de datos (usar funciones de “The Cumulative Statistics System”).

1. Mostrar la información de las tuplas con 2000 €.

sefsef

1. Mostrar la información de la tupla con producto\_id igual a 60000.

sefse

1. Mostrar los datos de los productos que tienen un producto\_id entre 180000 y

sefse

200000.

1. Contar el número de productos que valen más de 4000 €.

sefse

1. Mostrar el número total de unidades de los productos que tienen el mismo precio.

sefse

1. Insertar un nuevo producto en la tabla productos con un precio de 1000 €.

fesf

1. Actualizar el precio del producto insertado anteriormente para cambiar de 1000

€ a 2000 €.

fesf

Cuestión 24. Borrar los índices creados en la cuestión 20. Crear un índice multiclave tipo árbol sobre los campos stock y precio. Incluir el código SQL utilizado para ello.

fsefe

Cuestión 25. Para cada una de las consultas que se muestran a continuación, ¿Qué información se puede obtener de los datos monitorizados por la base de datos al realizar la consulta? Comentar cómo se ha realizado la resolución de la consulta.

¿Cuántos bloques se han leído de cada estructura? ¿Por qué? Importante, reinicializar los datos recolectados de la actividad de la base de datos antes de lanzar cada consulta:

1. Mostrar los productos que tienen un stock de 100 unidades y un precio de 1000

€.

fefes

1. Mostrar los productos que tienen un stock de 100 unidades o un precio de 1000

€.

fefe

1. Mostrar los productos que tienen un stock de 100 unidades.

sefsf

1. Mostrar los productos que tienen un precio de 1000 €.

fefsef

Cuestión 26. Crear la tabla **productos3** particionada por medio de una función de asociación h = stock mod 10. Para cada una de las consultas que se muestran a continuación, ¿Qué información se puede obtener de los datos monitorizados por la base de datos al realizar la consulta? Comentar cómo se ha realizado la resolución de la consulta. ¿Cuántos bloques se han leído de cada estructura? ¿Por qué? Comparar con la teoría. Importante, reinicializar los datos recolectados de la actividad de la base de datos antes de lanzar cada consulta.

1. Mostrar el número de productos con un stock de 200.

fefesf

1. Mostrar los productos que tienen un stock de 100 o 200 o 300.

hh

1. Mostrar los productos con un stock de más de 100 y menos de 200.

iiiu

Cuestión 27. A la vista de los resultados obtenidos de este apartado, comentar las conclusiones que se pueden obtener del acceso de PostgreSQL a los datos almacenados en disco.

iuhiuiuhiugiu

# Bibliografía (PostgreSQL 16)

* + Capítulo 1: Getting Started.
  + Capítulo 5: 5.6 System Columns.
  + Capítulo 5: 5.12 Table Partitioning.
  + Capítulo 11: Indexes.
  + Capítulo 19: Server Configuration.
  + Capítulo 24: Routine Database Maintenance Tasks.
  + Capítulo 27: Monitoring Database Activity. The statistics Collector
  + Capítulo 27.6: Monitoring Disk Usage. Determining Disk Usage
  + Capítulo 51: System Catalogs
  + Capítulo 64.1: B-Tree Indexes
  + Capítulo 64.6: Hash Indexes
  + Capítulo 65: Database Physical Storage
  + Capítulo VI.II: PostgresSQL Client Applications.
  + Capítulo VI.III: PostgresSQL Server Applications.
  + Apéndice F: Additional Supplied Modules. Pageinspect, pgstatutuple
  + Apéndice G: Additional Supplied Programs. Oid2name