Grafos

23.	Para representar grafos irrestrictos son más performantes las representaciones dinámicas que las estáticas.	**F**	*Los grafos irrestrictos son más performantes si se implementan en forma estática, ya que de lo contrario tendrían que estar pidiendo espacio a memoria en forma continua.*
59.	Para reducir espacio al representar un grafo siempre es más conveniente la forma dinámica que estática.	**V**	*La forma dinámica ocupa menor cantidad de espacio que la estática, ya que la estática pide todo el espacio de entrada, sin aprovecharlo del todo.*
66.	La única estructura de datos estática capaz de representar cualquier grafo es una matriz.	**V**	*La matriz podría ser de adyacencia o de incidencia.*
107.	Todo grafo de grado dos es binario.	**F**	*Contraejemplo: un grafo reflexivo.*
113.	Dado el grafo G = {(E; P); E({a}), P({a; a}) se puede decir que es un árbol.	**F**	*No se puede decir que es un árbol porque es reflexivo.*

Árboles

Árbol-B → árbol de búsqueda.
AB → árbol binario.
ABB → árbol binario de búsqueda.
AVL → ABB en donde la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho, 1.

3.	El método de Árbol B no es aplicable a archivos con grandes volúmenes de datos.	**F**	*El método de Árbol B sí es aplicable a grandes volúmenes de datos: es lo que utilizan los sistemas operativos para manejar sus sistemas de directorio y las bases de datos para manejar los índices.*
11.	En un índice de un DBMS, armado en un Árbol B, el tiempo de acceso a la información depende en parte del tamaño de la clave almacenada.	**F**	*El tiempo de acceso en un árbol-B depende del orden y de la altura.*
33.	Dado el árbol {(c, a); (c, b); (c, d); (c, e)} su barrido simétrico es a, b, c, d, e.	**F**	*No se puede hacer un barrido simétrico en un árbol no binario.*
42.	Si un árbol está balanceado entonces está completo.	**V**	
43.	Un ABB siempre es más rápido que una lista para ordenar un conjunto de valores.	**F**	*Las velocidades de ordenamiento de un ABB y de una lista dependen del algoritmo de ordenamiento y de los datos.*
44.	Un AB con cuatro nodos nunca puede ser completo.	**F**	*Contraejemplo: {(1, 2), (1, 3), (2, 4)}.*
47.	Debido a que el crecimiento de un árbol es exponencial en base al ancho del mismo, los tiempos de búsqueda en el mismo son siempre logarítmicos.	**F**	*Los tiempos de búsqueda sobre un árbol dependen de cosas como su balanceo.*
50.	Un árbol de expresión siempre es completo.	-	*Depende de lo que se interprete por "completo":* *· Si se piensa por el lado de que todos los nodos no maximales tienen que tener el mismo grado, es completo.* *· Si se piensa por el lado de que para un árbol de profundidad h, todos los nodos hasta h-1 tienen que tener grado 2 y el nivel h se completa de izquierda a derecha, entonces no es completo.*
52.	Un vector es una representación computacional estática que puede almacenar un árbol.	**V**	
54.	En un árbol de expresión los nodos maximales siempre son los operadores.	**F**	*Los nodos maximales en un árbol de expresión son operandos.*

64.	En la implementación de un Árbol B, todos los nodos de datos que contienen claves se encuentran en el mismo nivel.	**V**	
67.	El Árbol B garantiza un número de niveles menor que otros árboles.	**F**	*Los árboles-B no garantizan menos niveles. Los niveles dependen de cosas como la cantidad de datos, el orden, el grado de completitud de un nodo, …*
70.	Si un AB está completo y balanceado todas las hojas están en el mismo nivel	-	*Depende lo que se entienda por completo.* *· Si es un árbol cuyos nodos no maximales tienen grado dos, es FALSO.* *· Si es un árbol que tiene una profundidad h y que todos los nodos hasta h-1 tienen grado dos y los que están en h se colocan de izquierda a derecha, es VERDADERO.*
78.	Un árbol siempre tiene más punteros que elementos de datos.	**F**	*Un árbol implementado con un vector no tiene punteros.*
87.	Un ABB recorrido en orden simétrico (infijo) siempre devuelve un conjunto de valores ordenados.	**V**	*Un ABB recorrido en forma infija devuelve un conjunto de valores ordenados en forma ascendente.*
96.	El árbol de expresión siempre está balanceado en su raíz.	**F**	*Contraejemplo: a + (b + c) * d.*
99.	La cantidad de nodos en un árbol de expresión siempre es par.	**F**	*Contraejemplo: {(x, 1), (y, 2)}.*
106.	Es posible implementar el concepto de ABB con un vector.	**V**	
112.	Sobre un árbol n-ario con n>2, se pueden realizar los barridos pre-orden, simétrico, post-orden y por niveles.	**F**	*En un árbol n-ario con n>2 solamente se puede realizar el barrido por niveles.*
105.	Cuando un ABB se basa en un AVL su orden de complejidad es el mejor n log2 n.	**F**	*El orden de complejidad de un árbol AVL es O(log n).*
115.	El orden de complejidad de un ABB es similar al del Árbol B.	**F**	*El orden de complejidad de un árbol-B es mejor al de un ABB.*

Árboles – Métodos de Clasificación

56.	El orden de complejidad de un Árbol B siempre es mejor que el orden de complejidad del *Quicksort*.	**F**	*El orden de complejidad puede variar de acuerdo a cómo vengan ordenados los datos*
55.	El orden de complejidad del *Quicksort* puede variar dependiendo de cómo vengan ordenados los datos.	**F**	*El orden de complejidad del Quicksort depende de la elección del pivote. En el mejor de los casos, se elige el pivote que divide a la lista en dos partes iguales. Pero en el peor de los casos, se elige el pivote que divide a la lista en una gran lista y una muy pequeña.*
57.	El algoritmo *Quicksort* tiene un promedio un grado de complejidad O(n log n) pero en determinada circunstancia puede tener grado de complejidad $O(n^2)$ y ser el peor de todos los métodos de clasificación.	**V**	*Si bien es cierto que en el peor de los casos su orden de complejidad es $O(n^2)$, no es el peor algoritmo ya que, por ejemplo, el algoritmo de la burbuja también se comporta como $O(n^2)$.*
94.	Si tengo un conjunto de datos tendiendo a ordenados, el algoritmo de *Quicksort* es el más eficiente para su ordenamiento total.	**F**	*Quicksort puede llegar a ser bueno según la elección del pivote. Otro bueno podría ser Heapsort, que es in situ y con orden O(n · log n).*
53.	El algoritmo *Quicksort* siempre tiene el mismo orden de complejidad que el algoritmo de *Heapsort*.	**F**	*El orden de complejidad de los algoritmos depende de cómo vengan ordenados los datos. Quicksort en el peor de los casos es $O(n^2)$, mientras que Heapsort es O(n log n).*
97.	El *Heapsort* tiene peor rendimiento si los datos ya vienen ordenados.	**F**	*El rendimiento del Heapsort es siempre constante y es O(n log n).*

Hashing – Tablas de Hash - Funciones de hash

12. La técnica de *hashing* puede generar muchas lecturas secuenciales para un valor clave *hash* cuando hay alto grado de repetición de claves de usuario.	**V**	
62. La técnica de *hashing* abierto puede generar muchas lecturas secuenciales para un valor "clave *hash*" cuando hay alto grado de repetición de claves de usuario.	**-**	
77. La técnica de *hashing* no puede ser implementada para uso de caché, ya que no garantiza el acceso de alta velocidad.	**-**	
84. Una tabla de *hash* permite desarrollar un mecanismo indexado para recuperación de claves únicas.	**V**	*También se permite para claves duplicadas.*
45. La implementación de la cantidad de entradas para claves en una tabla de *hash* es dinámica.	**F**	*La implementación de las claves de hash pueden ser dinámicas o estáticas.*
109. La implementación de la cantidad de entradas para una tabla de *hash* es dinámica.	**F**	*Puede ser dinámica o estática.*
32. Las funciones de *hash* no poseen funciones inversas.	**F**	*Hay funciones hash que permiten la inversibilidad.*
40. Si una función de *hash* no posee una buena dispersión, se van a producir muchas colisiones.	**V**	
37. La implementación de un *hash* permite obtener a partir de una entrada "x" una salida "y" única y reversible.	**F**	*La función de hash, ante una entrada determinada, devuelve siempre la misma salida. Hay situaciones en las que no se puede recuperar la entrada teniendo la salida.*

Hashing vs Árbol-B

9. La técnica de *hashing* es más rápida para el acceso a los datos que el Árbol B.	**-**	*Hashing es más veloz para accesos directos.* *Árbol B es más veloz para accesos secuenciales.*
10. La técnica de *hashing* es más rápida para el tratamiento de claves duplicadas que el Árbol B.	**-**	
68. La técnica de *hashing* es menos performante que el Árbol B para el manejo de claves duplicadas.	**V**	
74. Todos los DBMS utilizan la técnica de *hashing* para el armado de sus índices.	**(F)**	*No sólo se usa la técnica de hashing, sino también la de los árboles-B.*
100. La técnica de *hashing* es más performante que Árbol B en la búsqueda de una clave existente en particular.	**V**	*Hasing es más veloz para accesos directos.*

Algoritmo de Huffman

20.	La compactación por algoritmo de Huffman permite redefinir el almacenamiento lógico de símbolos de tal manera que la pérdida de información sea despreciable.	**F**	*La compactación por algoritmo de Huffman no pierde información, ya que es sin pérdida.*
48.	El método de compresión de Huffman es sin pérdida, por eso no es recomendable para compactar imágenes o video.	**V**	*Para reducir el tamaño de imágenes o videos, el algoritmo de Huffman no me servirá (es sin pérdida).*
83.	La compresión que se logra mediante el algoritmo de Huffman es mayor cuando la variedad de caracteres diferentes que aparecen es menor.	**V**	*Con menor cantidad de caracteres distintos, menor espacio ocupará cada carácter.*
21.	Un árbol de Huffman es siempre completo.	**V**	
35.	En un árbol de Huffman, si el código del caracter "m" es 1011, entonces no puede ningún otro caracter poseer el código 1011111	**V**	*Solamente los maximales tienen código… Si es maximal, no puede tener hijos.*
101.	El algoritmo de Huffman obtiene los códigos comprimidos parseando un AB balanceado.	**F**	*El árbol en el que se basa Huffman no está necesariamente balanceado.*
104.	El árbol en el que se basa Huffman es principal derecho balanceado.	**F**	*El árbol en el que se basa Huffman no está necesariamente balanceado.*
108.	En el algoritmo de Huffman, la cantidad de nodos es la siguiente: (total de hojas * 2) 1.	**V**	
38.	La reexpresión de caracteres al aplicar Huffman implica la disminución de 8 bits para la expresión de todos los caracteres.	**F**	*Cuando aplicamos Huffman, puede haber caracteres que terminen pesando más de 8 bits.*
69.	Para comprimir en el algoritmo de Huffman, se debe leer en un ciclo cada carácter del archivo a comprimir y acceder al árbol desde la raíz para llegar a la hoja que contiene el carácter. Si desciendo por un hijo izquierdo, agrego un 0 como bit del código comprimido. Si desciendo por un hijo derecho, agrego un 1 como bit del código comprimido.	**F**	*Para comprimir, se empieza a buscar desde el nodo maximal hacia la raíz.* *Para descomprimir, se empieza a buscar desde la raíz hasta el nodo maximal.*
110.	Si una palabra es capicúa, su código de Huffman también lo es.	**F**	*Contraejemplo: el código Huffman de una palabra que empieza y termina con la misma letra, cuyo código es 001, será: 001…001.*

Bases de Datos – Objetos de BD – RDBMS

1.	Un *constraint* de tipo CHECK siempre puede ser reemplazado por un *trigger*.	**V**	
7.	El checksum es una de las técnicas utilizadas para corroborar la integridad de los datos.	**V**	
8.	En un modelo de DB OLTP el concepto de transacción está asociado a la atomicidad de procesamiento.	**F**	*El concepto de transacción no solamente está asociado a la atomicidad, sino también a la consistencia, al aislamiento y a la durabilidad (ACID).*
13.	Una vista en un RDBMS permite restringir el acceso a modelo y tener una vista simplificada del mismo.	**V**	*Las vistas se usan para generar abstracciones más sencillas del modelo de datos. Además, pueden usarse para compartir código de una manera más segura ya que el cliente no tiene acceso al modelo de datos sino a la vista.*
14.	Las claves foráneas brindan integridad relacional al modelo de datos y es su principal función.	**V**	
15.	Las claves foráneas eliminan redundancia de datos.	**F**	*Aunque sean claves foráneas, puede haber redundancia de datos para mejorar la performance.*
16.	Los índices aseguran unicidad de claves.	**F**	
22.	Mediante la utilización de *triggers* se puede simular la integridad referencial entre tablas de diferentes bases de datos.	**V**	
24.	En PL-SQL, la única diferencia entre una función y un *stored procedure* es que puede retornar valores.	**F**	*No es la única diferencia.*
25.	Las únicas restricciones posibles a aplicar en el modelo relacional son por tabla.	**F**	*También se pueden aplicar restricciones a nivel de columna y de BD, por ejemplo.*
27.	Un índice en una BD relacional es una restricción al modelo físico.	**F**	*Un índice es independiente del modelo físico y del modelo lógico.*
28.	Una tabla no puede tener dos claves foráneas que referencien a la misma tabla.	**F**	*Contraejemplo: un registro de una tabla Persona podría tener una clave foránea a la tabla Persona que simbolice la relación madre y una que simbolice la relación padre.*
30.	Una clave primaria y una clave foránea son restricciones al modelo físico.	**F**	*La clave primaria y la clave foránea son restricciones al modelo lógico.*
31.	La vista conceptual o lógica se define mediante un esquema conceptual. Este esquema conceptual se escribe en DDL. Contiene definiciones del contenido de la base, tipos de datos, restricciones, reglas de integridad, etc.	**V**	
39.	La ejecución de una query sin filas de resultado dentro de un *trigger* genera la cancelación de la transacción.	**F**	*El trigger continúa ejecutando incluso ante una consulta que no devuelva filas.*
46.	La única forma de definir una restricción de integridad sobre una columna de una tabla es mediante la restricción CHECK.	**F**	*Puedo usar triggers también.*
41.	Nunca es posible ejecutar la operación de INSERT sobre una vista.	**F**	*Se puede hacer un INSERT sobre una vista.*
49.	En los RDBMS no está permitido hacer INSERTs sobre una vista.	**F**	*Se pueden hacer INSERTs sobre una vista.*

51.	Luego de ejecutar una sentencia SQL para crear una tabla, si se ejecuta un rollback la tabla queda dropeada.	-	*Para que quede dropeada, el CREATE TABLE se debería haber ejecutado dentro de una transacción.*
60.	Siempre es conveniente tener una tabla indexada por su clave primaria.	**V**	*Ya que es uno de los campos que más se utiliza para comparar.*
61.	Una *constraint* es más eficiente que un *trigger* para validar el dominio de un atributo.	**V**	*Una constraint se ejecuta en mucho menos tiempo que un trigger.*
63.	Una vista es el equivalente a una consulta estática de una o más tablas.	**F**	*Una vista es una consulta dinámica ya que cada vez que se usa, consulta las tablas. Es decir, puedo ejecutar la misma vista en dos momentos distintos y que me retornen dos resultados diferentes.*
71.	La acción que ejecuta un *trigger* y el evento que lo dispara siempre se ejecutan de manera atómica.	**V**	*La acción y el evento de un trigger conforman una transacción.*
79.	La normalización aplicada al diseño de un modelo de datos relacional nos permite desarrollar un modelo de manera estructurada, independiente de la performance que ese modelo pueda llegar a obtener.	**F**	*La normalización no es independiente de la performance. En algunas ocasiones se desnormaliza para poder mejorar la performance.*
81.	Los *triggers* son las funciones con las que cuenta un RDBMS que permite controlar la integridad referencial.	**V**	
82.	En un RDBMS, una función creada por el usuario puede modificar el contenido de una tabla de la base donde fue creada.	**F**	*Una función no puede tener efecto colateral.*
85.	Un DBMS que soporta atomicidad, consistencia, aislamiento y durabilidad puede ser considerado transaccional.	**V**	*Toda transacción cumple con las siglas ACID (atomicidad, consistencia, aislamiento y durabilidad).*
86.	Las tablas INSERTED y DELETED pueden ser utilizados en *stored procedures* y funciones siempre que estos sean invocados dentro de un *trigger*.	**F**	*Los contextos de las funciones, de los stored procedures y de los triggers son distintos, por eso no las podrán usar a menos que se la pase por parámetro.*
88.	No se pueden crear índices en las vistas.	**V**	
90.	Si se tiene una vista que relaciona varias tablas y una de ellas posee un índice, el motor no puede utilizarlo para generar una consulta más performante.	**F**	*Las vistas pueden usar los índices de las tablas a las que referencian, pero no pueden crearse índices sobre esas vistas.*
91.	Se puede ejecutar un *stored procedure* en una consulta SELECT.	**F**	
92.	Si se desea ocultar columnas de una tabla a una aplicación, el mejor objeto que podemos usar es un sinónimo.	**F**	*Para este caso, si bien emplear un sinónimo no está mal, la vista es el mejor objeto a usar.*
98.	Si se desea que no se puedan eliminar registro de una tabla de auditoría, una opción es crear un *trigger* que lo impida.	**V**	*Un trigger sobre un DELETE podría anular la eliminación del registro.*
103.	Gracias a los índices se puede asegurar la integridad referencial.	**V**	*Los índices aseguran valores únicos para las filas almacenadas.* *De todas modos, no son los únicos objetos que aseguran la integridad referencial, ya que existen las FKs.*
114.	La única manera para asegurar unicidad de los campos es con una clave primaria o con UNIQUE	**F**	*También se pueden usar triggers.*

Bussiness Intelligence (BDs multidimensionales – OLTP-OLAP – modelo STAR – Data Warehouse – Data Mining)

2.	Hipercubo y Multicubo son dos técnicas utilizadas para el armado de la capa interna de una Base de Datos Multidimensional.	**V**	
4. **72.**	Un modelo OLAP posee al menos una tabla de hechos con campos precalculados.	**F**	*Un modelo OLAP no contiene tablas. El que contiene tablas es el Data Warehouse.*
5.	En un modelo OLAP se puede aplicar normalización en cualquier tabla que no sea la tabla de hechos.	**F**	*Un modelo OLAP no contiene tablas. El que contiene tablas es el Data Warehouse. Y se puede aplicar normalización en las tablas de dimensión.*
6.	En un modelo OLAP se puede aplicar normalización en cualquier tabla de dimensión.	**V**	
19.	El modelo OLAP no se puede implementar si no se tiene una BD Multidimensional.	**F**	*El modelo OLAP se puede implementar sin una BD multidimensional, pero es conveniente no hacerlo.*
75.	En el modelo OLAP no es aplicable la normalización de datos.	**F**	*Existen dos modelos: el modelo STAR (no normalizado) y el modelo SNOWFLAKE (normalizado).*
76.	No es posible obtener el mismo resultado de una consulta OLAP a partir del OLTP que dio origen al mismo.	**F**	
36.	El modelo STAR no cumple con al menos una forma normal.	**V**	*En el modelo STAR la tabla de dimensiones no están normalizadas.*
65.	Una tabla de dimensión en el modelo STAR es el equivalente a una cara en un cubo de información multidimensional.	**F**	*Un atributo en la tabla de dimensión se corresponde con una cara en el cubo.*
73.	Las técnicas más comunes utilizadas en *Data Mining* son las redes neuronales, los árboles de decisión y las reglas de inducción.	**V**	
80.	*Data Mining* son las técnicas y algoritmos utilizados para encontrar información y relaciones ocultas en un *Data Warehouse*.	**V**	

Consultas SQL

17.	Para poder realizar UNION entre dos consultas, las mismas deben tener la misma cantidad de columnas en el mismo orden e igual tipo de dato por cada una.	**V**	*El UNION requiere que las dos consultas devuelvan la misma cantidad de columnas con el mismo tipo de datos.*
18.	Un SELECT con 3 tablas en el FROM consume igual recursos que 3 SELECT con una tabla cada una.	**F**	*Tres consultas consumen más recursos que una consulta a tres tablas.*
26.	En SQL, una subconsulta ubicada en el WHERE siempre debe retornar una fila y una columna.	**F**	*Una subconsulta en el WHERE podría retornar más de una columna si se lo usa con el operador IN.*
29.	Si en una consulta SELECT, hay al menos una función de grupo (COUNT, SUM, AVG, …) siempre debe colocarse la cláusula GROUP BY en dicho SELECT.	**F**	*Se puede hacer SELECT COUNT (*) FROM Libros.*
34.	Un *stored procedure* posee la siguiente sentencia en su cuerpo: **DELETE FROM tl_funciones WHERE tlstatus = 'Finalizada'** Si al ejecutar el procedimiento no existe ningún registro que cumpla con la citada condición, entonces la ejecución se cancela.	**F**	
89.	Luego de ejecutar la siguiente instrucción: **CREATE TABLE miTablita (campo1 VARCHAR(5), campo2 VARCHAR(5))** y poblar la tabla, la siguiente igualdad va a ser siempre verdadera para cualquier set de datos posibles: **SELECT COUNT(campo1) FROM miTablita = SELECT COUNT(campo2) FROM miTablita**.	**F**	*Como COUNT no cuenta los NULLs cuando es aplicado sobre un campo, si el número de NULLs en cada campo es diferente, las consultas van a ser distintas.*
93.	Las siguientes consultas nunca devuelve el mismo resultado a excepción de la primera que siempre devuelve nada: **SELECT 1 FROM tablita WHERE campo1 = 1 OR campo2 = 2** **SELECT 1 FROM tablita WHERE campo1 = 1 UNION SELECT 1 FROM tablita WHERE campo2 = 2**	**F**	*La primera no devuelve NULL, sino un registro 1 por cada registro donde su campo1=1 o bien su campo2=2.*
95.	Si una columna posee la *constraint* UNIQUE, entonces una sola fila como máximo puede contener NULL en dicha columna.	**V**	
102.	Si existe un SELECT dentro de una cláusula HAVING, este debe retornar una fila y una columna.	**F**	*Podría retornar más de una fila si se lo usa con el operador IN.*
111.	Las siguientes consultas devuelven lo mismo sin importar el set de datos: **SELECT COUNT(campo1) FROM tabla = SELECT COUNT(*) FROM tabla**. La "tabla" sólo tiene la columna campo1.	**F**	*No devuelven lo mismo.* *COUNT(campo1) no cuenta los valores nulos mientras que COUNT(*) sí cuenta los valores nulos.*

- En el SELECT, el DISTINCT tiene en cuenta los NULLs.
- En el GROUP BY, los NULLs se procesan como un grupo.
- Para las funciones agregadas:
    - En una tabla vacía, COUNT → devuelve 0.
      En una tabla vacía, AVG/MAX/MIN/SUM → devuelve NULL.
    - COUNT(*) → considera tanto los NULLs como los valores repetidos.
      AVG/MAX/MIN/SUM(*) → no existe.
    - COUNT(unCampo) → no considera los NULLs, pero sí considera los valores repetidos.
      AVG/MAX/MIN/SUM(unCampo) → no consideran los NULLs, pero sí consideran los valores repetidos.
    - COUNT(DISTINCT unCampo) → no considera ni los NULLs ni los valores repetidos.
      AVG/MAX/MIN/SUM(DISTINCT unCampo) → no consideran ni los NULLs ni los repetidos.