1. Qué es el system catalog?
    1. El system catalog es el lugar donde un RDBMS guarda los metadatos del esquema. Los metadatos pueden incluir:
        1. Información sobre las tablas y columnas
        2. Views
        3. índices
        4. Usuarios y grupos de usuario.
        5. Triggers
        6. Funciones de agregación de finitas por el usuario.
2. Qué es un DDL (data definition language)?
    1. Es el lenguaje utilizado para definir la estructura de la base de datos o los esquemas.
        1. CREATE, ALTER, DROP, TRUNCATE, COMMENT, RENAME
3. Qué es un DML (data manipulation language)?
    1. Es el lenguaje utilizado para manipular los datos.
        1. SELECT, INSERT, DELETE, MERGE, CALL
4. Cuáles son los tipos de atributos?
    1. Composite vs Simple (atómicos):
        1. Los composite pueden ser divididos en sub-partes que representan atributos con significados independientes. Por ejemplo, dirección puede ser dividida en calle, ciudad, estado, etc. Estas sub-partes pueden estar divididas también incluso.
        2. Pueden utilizarse en los casos que se requiera el atributo como un conjunto de sub-partes o como entero.
        3. Los atributos atómicos son indivisibles.
5. Single-Valued vs Multivalued:
    1. Los single-valuad tienen un solo valor para un atributo. Ejemplo, edad.
    2. Los multivalued pueden tener más de un valor al mismo tiempo.
        1. Por ejemplo:
            1. Color de auto: azul, gris, negro.
            2. Hobbies: leer, correr, cantar.
6. Cuál es la diferencia entre atributos multivaluados y compuestos?
    1. Los atributos multivaluados tienen múltiples valores para el mismo atributo, por ejemplo, teléfono:
        1. 555-6554
        2. 555-1354
        3. 555-8214
    2. Los atributos compuestos tienen múltiples atributos, cada uno con su valor, por ejemplo, dirección, con calle y ciudad:
        1. Av de Mayo, Carhué
        2. Av 9 de Julio, Buenos Aires
7. Cuál es la diferencia entre los sistemas de bases de datos distribuidos homogéneos y heterogéneos?

    1. En las bases de datos distribuídas homogéneas todos los nodos de la misma utilizan el mismo software y acuerdan en cooperar para satisfacer las peticiones de los usuarios.
        1. Dentro del software:
            1. Cada nodo tiene el mismo software (DBMS) o compatible.
            2. Cada nodo tiene las mismas estructuras de datos o son compatibles.
    2. En las bases de datos heterogéneas, los diferentes nodos de la BD pueden utilizar diferente software y esquemas. Cada nodo puede no conocerse entre sí y pueden cooperar de manera limitada en satisfacer las peticiones de los usuarios.
8. Qué es una historia?
    1. Una historia es un orden parcial sobre un conjunto de operaciones que pertenecen a un conjunto de transacciones.
9. Cuándo una operación es conflictiva?
    1. Dos operaciones están en conflicto con respecto a un dato, si ambas operan sobre el mismo dato y al menos una de ellas escribe.
10. Cuándo dos historias son equivalentes (conflicto equivalentes)?
    1. Cuando dos historias H1, H2, están definidas sobre el mismo conjunto de transacciones y el orden de las operaciones conflictivas de las transacciones que no abortan es el mismo.
11. Cuándo una historia es serial? De un ejemplo
    1. Dado un par de transacciones Ti, Tj, y sus operaciones, una historia es serial si todas las operaciones de Ti se ejecutan antes de Tj (en la historia) o viceversa.
    2. Sean T1 = R1(X); W1(X); C1 y T2 = R2(X); W2(X); C2, H = R1(X); W1(X); C1; R2(X); W2(X);
12. Cuándo es una historia serializable? Dar un ejemplo de una historia serializable y una que no lo sea.
    1. Una historia es serializable cuando es conflicto equivalente a otra historia serial.
    2. Dados T1 = R1(Y); W1(Y); C1 y T2 = R2(X); W2(X); C2,
        1. H = R1(Y); R2(X) W2(X); W1(Y); W2(X); C1; C2. es serializable.
    3. Dados T1 = R1(Y); W1(X); C1 y T2 = R2(X); W2(Y); C2,
        1. H = R1(Y), R2(X); W1(X); W2(Y); C1; C2
            1. T1 -Y->T2; T2 -X->T1 (existe un ciclo en el grafo de precedencia), no es serializable.
13. En qué consiste el método para probar que una historia es serializable?
    1. Se construye el grafo de precedencia de la siguiente manera:
        1. Los nodos son cada transacción de la historia.
        2. Se crea un eje por cada operación en Ti que precede a otra operación de Tj y ambas están en conflicto.
        3. Se anota en los ejes el item sobre el cuál ocurre el conflicto.
    2. Si el grafo no tiene ciclos, entonces es serializable.
14. Qué quiere decir que una transacción lee de otra?

    1. Una transacción Y lee un valor C de otra transacción X cuando X fue la última transacción en escribir X antes de que Y la leyera y no abortó antes de que la leyera.
15. Cuándo una historia H es recuperable? Dar un ejemplo de cuando no lo sea.
    1. Una historia H es recuperable si toda transacción Ti de H hace commit después de que todas las transacciones de las que lee hicieron commit.
    2. W1(X); R2(X); W2(Y);C2; A1 => T2 lee X de T1, realiza un commit (C2) pero luego T1 aborta A1, quedándonos con una ejecución errónea.
16. Cuándo una historia H es ACA (avoid cascading aborts)?
    1. Una historia H es ACA cuando las transacciones sólo leen de otras transacciones cuando estas hicieron commit.
17. Cuándo una historia es Strict (ST)
    1. Una historia es ST si ninguna transacción lee o escribe un ítem antes de que la última transacción que lo escribió haya hecho commit.
18. Describir locking
    1. Un lock es una variable asociada con algún dato (item de dato) que determina que operaciones pueden realizarse sobre estos.
    2. Locking binario
        1. El más simple, las transacciones se pueden pensar como un conjunto de operaciones ordenadas de locks y unlocks.
        2. Es necesario tomar el lock de un ítem antes de leerlo o escribirlo. También es necesario desbloquear el lock al no necesitarlo.
        3. Una transacción que quiera operar sobre un ítem que está lockeado por otra transacción debe esperar a que se libere.
    3. Locking ternario
        1. Los locks ahora son de lectura o de escritura, para permitir múltiples lecturas.
    4. Update lock
        1. Se utiliza un update lock para garantizar lectura a una transacción sobre un ítem, y si este quiere actualizar el lock a escritura, puede.
    5. Locking y serializabilidad
        1. 2PL (Two phase locking)
            1. Una transacción satisface 2PL si todo lock ocurre antes del primer unlock en la transacción.
            2. Si toda transacción T en H respeta 2PL, entonces es serializable.
        2. 2PLE (Two phase locking estricto)
            1. Una transacción satisface 2PLE si satisface 2PL y además no libera ninguno de sus locks de escritura antes de hacer commit o abort.
        3. 2PLR (Riguroso)
            1. Una transacción satisface 2PLR si es 2PL y no libera ninguno de sus locks de escritura o lectura antes de hacer commit o abort.

19. Describir el método de timestamping en el modelo de concurrencia y recuperabilidad optimista.
    1. El planificador da un timestamp a cada transacción. Este timestamp determina el orden de serializabilidad de las mismas.
    2. Se le asocia a cada elemento de la base de datos, dos elementos y un bit:
        1. RT(X), determina el timestamp de la transacción que leyó de X que tiene mayor timestamp.
        2. WT(X), determina el timestamp más alto que escribió X.
        3. CT(X), es verdadero si la transacción que escribió más recientemente X commiteó.
    3. El planificador debe admitir una operación, abortar o hacer rollback de la transacción que quiere realizar la operación según el timestamp y una serie de reglas:
        1. Si se recibe un read, el timestamp de la transacción que lee debe ser mayor o igual al timestamp de la transacción que escribió (el dato). Si esto no pasa, se tiene un **read too late**.
            1. Además, si CT'(X) es true, conceder la lectura, caso contrario, demorar T hasta que CT'(X) sea verdadero o T' aborte.
        2. Si se recibe un write (sobre un dato), el timestamp de la transacción que escribe debe ser mayor o igual del timestamp de RT(X) y mayor o igual que el timestamp de la transacción registrado en WT(X). Así se previenen reads y lost updates.
    4. Si las condiciones no se cumplen, se puede abortar o hacer rollback de la transacción que realiza una operación no realizable (físicamente no realizable) y reiniciarse con un nuevo timestamp.
    5. Cuando la operación es físicamente realizable, se actualizan los valores de RT, WT y CT según corresponda.
    6. Si el planificador recibe abort o rollback:
        1. Cada transacción que estaba esperando por un elemento x que T escribió debe repetir el intento de lectura o escritura y verificar si ahora el intento es legal.
20. Describir el método de timestamping multi-versión en el modelo de concurrencia y recuperabilidad optimista.
    1. Similar a el modelo de timestamp, pero se mantienen diferentes versiones de los ítems de las bases de datos con cada escritura.
    2. Permite lecturas que abortarían si fueran hechas en casos donde los ítems fueron escritos por transacciones posteriores, leyendo la versión del ítem adecuado según el timestamp de la transacción.
    3. Cuando ocurre una escritura, se crea una versión de la misma (Xt), asociada al timestamp de la transacción que la realizó.
    4. Los tiempos de lectura ahora se asocian a las diferentes versiones de escritura.
    5. Cuando se realiza una lectura, se busca la versión de X (Xt) tal que TS(S) >= t, y que no exista un t' tal que TS(S) > t' > t.

6. Una transacción T que quiere escribir debe hacer rollback cuando existe alguna Xt,tr tal que TS(T') > t y TS(T') < tr. Es decir, no se puede escribir si exista una transacción creó una versión anterior de X y alguna transacción que la leyó tiene un tiempo de lectura mayor al de la transacción que quiere escribir (ya que debería haber leído la modificación de la que quiere escribir).
7. Se puede borrar cualquier Xt que tenga tiempo menor a toda transacción activa.

21. Describir el método de concurrencia y recuperabilidad optimista de validación
    1. Se tiene para cada transacción:
        1. Rs(T): elementos leídos por T
        2. Ws(T): elementos escritos por T
    2. Las transacciones se ejecutan en 3 fases:
        1. Se lee desde la BD todos los elementos en Rs(T)
        2. Se validan las operaciones de Rs(T) y Ws(T) con la de otras transacciones. Si la validación falla, se ejecuta un rollback y se comienza nuevamente.
        3. Se escriben los elementos de Ws en la BD.
    3. Validación:
        1. Se verifica que Rs(T) ∩ Ws(U) es vacío para cualquier transacción U validada previamente que no finalizó antes de que T comience.
        2. Se verifica que Ws(T) ∩ Ws(U) es vacío para cualquier transacción U validada previamente y que no finalizó antes de que T sea validada.
    4. El planificador mantiene los conjuntos de start (transacciones no validadas), val (transacciones validadas pero no escritas), end (transacciones escritas), con los cuales valida las transacciones.
22. Describir los problemas de serialización es sistemas concurrentes
    1. Dirty read
        1. Una transacción T1 lee de otra transacción T2 (y commitea), pero T1 aborta o hace rollback, dejando un valor incorrecto escrito por T2.
    2. Lost update
        1. Dos transacciones, T1 y T2, T1 ocurre antes que T2.
        2. T1 lee un valor de X, T2 lee el mismo valor, T1 actualiza X según el valor leído anteriormente y T2 hace lo mismo. T2 debería haber leído el valor actualizado por T1 para determinar su actualización.
    3. Phantom read
        1. Ocurre cuando dentro de una transacción, dos operaciones idénticas de selección (sin modificar lo que se lee) arrojan distintas tuplas.
    4. Incorrect summary
        1. Ocurre cuando una transacción T1 está actualizando valores que otra transacción T2 está utilizando para calcular una función de agregación de suma. Algunos valores serán modificados antes de ser sumados y otros no, obteniendo un valor incorrecto.
    5. Unrepeatable read

        1. Ocurre cuando dentro de una transacción, dos reads (uno atrás del otro o sin modificar los datos), leen diferentes valores del mismo dato.
23. Qué es una dependencia funcional?
    1. Se dice que X determina funcionalmente a Y (X->Y) o que Y depende funcionalmente de X si para todo par t1, t2 tal que t1(x) = t2(x), entonces t1(y) = t2(y).
24. Quién establece las dependencias funcionales?
    1. El administrador de la BD.
25. Qué es la clausura de un atributo?
    1. La clausura de un atributo con respecto a F es el conjunto de todos los atributos A tal que X -> A. Es decir, todos los atributos a los que X determina funcionalmente.
26. Cómo se calcula la clausura de un atributo?
    1. Se puede utilizar el siguiente algoritmo:
        1. X0 = X
        2. Xi+1 = Xi unión conjunto de atributos A tal que hay alguna dependencia funcional Y -> Z en F, A está en Z e Y está incluida en Xi.
            1. Xi+1 = Xi unión el conjunto de atributos Z tal que algún subconjunto Y de Xi determine funcionalmente a Z.
27. Cómo se define la clausura de F conjunto de DF?
    1. El conjunto de todas las dependencias funcionales que se pueden inferir de F aplicando los axiomas de Armstrong.
28. Cuál es la diferencia entre superclave y clave?
    1. Dado X -> R, X es clave <=> no existe Y incluído en X, Y != X.
29. Cómo se calcula una clave candidata?
    1. Hay que calcular la clausura de todos los atributos y conjuntos de ellos.
        1. Una manera de hacerlo más rápido es calcular la clausura de los atributos que no figuran nunca en los lados derechos, ya que algún atributo debería determinarlo funcionalmente o este atributo determina funcionalmente a los demás.
        2. Si lo anterior no funciona, se calcula la clausura de un atributo X, si esta no es igual a R, se agrega otro atributo hasta conseguir que sea igual.
30. Cuándo dos conjuntos de dependencias funcionales son equivalentes?
    1. Dado F y G conjuntos de DF, estos son equivalentes si F+ = G+ o si F |= G y G |= F
31. Qué es un cubrimiento minimal?
    1. Un cubrimiento minimal de F (conjunto de dependencias funcionales) es un conjunto de DF Fm tal que Fm es equivalente a F y además:
        1. Todo lado derechos de las DF de Fm tienen un único atributo
        2. Todo lado izquierdo es reducido (no hay atributos redundantes)
        3. No contiene DF redundantes (generalmente por transitividad).
    2. Ej: cubrimiento minimal de R(A, B, C, D) dado F = {A->BD, B->C, C->D, BC->D}

1. A ->B, A-> D, B -> C, C->D, BC->D (lado izquierdo tiene atributos redundantes)
2. A->B, A->D, B->C, C->D, C->D (C->D duplicada, A->D redundante)
3. A->B, B->C, C->D es el cubrimiento minimal de F.

32. Qué problemas podría presentar un esquema?
    1. Anomalías de inserción
        1. Cuando se tienen atributos no pertinentes o aglomerados en una relación, es posible que en la inserción algunos de estos deban llevar null.
    2. Anomalías de actualización
        1. Debido a la redundancia, se debe actualizar varias veces el mismo valor en distintos lugares.
    3. Anomalías de borrado
        1. Puede ocurrir pérdida de información debido a que queremos borrar algo específico, pero el esquema de relación tiene más información.
    4. Redundancia de información
        1. Ejemplo: El nombre del alumno se repite por cada materia en la que se inscribe y por cada examen que rinda.
33. Cómo puedo saber si una descomposición binaria es con pérdida de información?
    1. Una descomposición ρ de R, ρ = (R1,R2) es sin pérdida de información con respecto a un conjunto de dependencias funcionales F, <=>
        1. La dependencia funcional (R1 ∩ R2)-> (R1 – R2) está en F+ **o**
        2. La dependencia funcional (R1 ∩ R2)-> (R2 – R1) está en F+
    2. De manera más sencilla, tiene que conservarse un conjunto de atributos que comparten ambas nuevas relaciones que determina funcionalmente a la otra, de manera que el join natural no produzca tuplas espurias.
34. Cómo puedo saber si cualquier descomposición es con pérdida de información?
    1. Algoritmo de tableau:
        1. Se arma una matriz donde cada columna son los atributos de R y las filas son las subrelaciones de la descomposición.
        2. En cada valor de la matriz se pone aj, con j número de columna, si la descomposición contiene ese atributo, o bij si la descomposición no tiene ese atributo.
        3. Para cada dependencia funcional, se toma el lado izquierdo de la DF, para todas las filas que coincidan con su lado izquierdo (mismo valores en la misma columna), se igualan los lados derechos de la DF.
        4. Se repite hasta que no haya cambios.
        5. Si hay una fila con todos símbolos distinguidos (aj), entonces es sin pérdida de información.
35. Qué es la proyección de un conjunto de dependencias funcionales sobre un conjunto de atributos?

1. Dados R, y ρ = (R1, R2,...., Rk) decimos que la proyección de F sobre un conjunto de atributos Z (Π Z (F)) es el conjunto de dependencias funcionales X->Y Є F+ tal que XY ⊆ Z.
2. Es decir, es el subconjunto de dependencias funcionales que pertenecen a F+ y pueden formarse con el subconjunto de atributos Z.

36. Cómo puedo saber si una descomposición conserva las dependencias funcionales?
    1. Una descomposición ρ preserva F, si la unión de todas las dependencias funcionales en Π Ri (F) para i= 1, 2, ..., k implican F , o sea:
        1. F+=(Ui=1,k ΠRi(F))+
    2. Si la unión de la proyección de F sobre el conjunto de atributos de cada Ri es F+.
37. Dar un ejemplo de una descomposición con pérdida de dependencias funcionales
    1. R(A, B, C), F = {AC->B, B->C}, ρ = {R1(AB), R2(BC)}
        1. F+ = {AC->B, AC->C, AC->A, AC->AC, B->B, B->C, A->A, C->C}
        2. Proyección de F sobre AB = {B->B, B->C, A->A}
        3. Proyección de F sobre BC = {B->B, B->C, C->C}
        4. Se pierde AC->B, existe pérdida de dependencias funcionales.
38. Cómo determino que una descomposición es sin pérdida de DF?
    1. Algoritmo para determinar si es con pérdida de DF sin calcular F+
    2. Para cada DF en F:
        1. Dados R, F y ρ, queremos verificar si se preserva X -- > Y
        2. Z:= X;
        3. while Z cambie do for i:= 1 to k do
        4. Z:= Z U ((Z ∩ Ri)+ ∩ Ri);
        5. Si Y ⊆ Z luego X->Y, y si esto se satisface para toda DF, entonces es sin pérdida de DF.
    3. Ejemplo de aplicación para el ejemplo anterior:
        1. Queremos ver si se cumple AC->B
        2. Z = AC
            1. Z = AC U ((AC ∩ AB)+ ∩ AB) = AC.
            2. Z = AC U ((AC ∩ BC)+ ∩ BC) = AC.
        3. Tiene pérdida de DF.
39. Qué es un atributo primo?
    1. Un atributo es primo si es miembro de alguna clave candidata.
40. Qué es una DF completa, parcial y transitiva?
    1. Parcial:
        1. Se dice que, dado X -> Y, X determina funcionalmente de manera parcial a Y si existe un subconjunto Z incluído en X tal que Z->Y.
    2. Completa:
        1. Una dependencia funcional X->Y es completa si al remover cualquier atributo de X, entonces X->Y no se mantiene.
    3. Transitiva:

    1. Una dependencia funcional X->Y es transitiva si existe Z tal que X -> Z y Z -> Y.

41. Cuándo una relación está en 1FN, 2FN, 3FN, FNBC?
    1. 1FN:
        1. Un esquema de relación R está en 1FN si el dominio de sus atributos no contiene atributos multivaluados ni compuestos.
    2. 2FN:
        1. Un esquema de relación R está si todo atributo no primo A en R **no** es parcialmente dependiente de alguna clave de R.
    3. 3FN:
        1. R está en 3FN si para toda dependencia funcional no trivial X->Y sobre R, X es superclave de R o Y es primo.
        2. Es decir, para toda DF X->Y, X es superclave de R o Y pertenece a alguna clave de R.
        3. Todo esquema se puede descomponer en 3FN que sea SPI y SPDF.
        4.
    4. FNBC
        1. R está en FNBC si para TODA dependencia funcional X->Y en F+, o bien $Y \subseteq X$ o X es una superclave de R.
        2. Hay esquemas que no se pueden descomponer en FNBC y que sean SPDF.
42. Por qué todo esquema de 2 atributos tiene que estar en FNBC?
    1. Todo esquema de dos atributos R = (AB) tiene que estar en la FNBC ya que:
        1. O A es clave => A -> B (A->AB), si B->A, entonces B es clave candidata también (B->AB), por lo que la FNBC cumple.
        2. O B es clave, idem.
        3. O AB es clave, AB->AB, lo cual cumple, ambas son claves.
43. Cuál es el algoritmo utilizado para realizar una descomposición en FNBC sin pérdida de información?
    1. El algoritmo aplica la propiedad de descomposición binaria:
        1. Si hay una DF X->Y que viola FNBC, se parte R en R1 y R2 de manera que:
            1. R1 = XY
            2. R2 = R-Y
        2. La manera de hacerlo a mano es poniendo la relación R como raíz (marcar las claves, para detectar violaciones a la FNBC), y sacar ramas, a la derecha y a la izquierda, que representen R1 y R2. Continuar haciendo esto hasta que ninguna viole la FNBC.
44. Cuál es el algoritmo utilizado para realizar una descomposición 3FN sin pérdida de información y sin pérdida de dependencias funcionales?
    1. Se realiza una descomposición por **síntesis** partiendo de una cobertura minimal de F.

2. Cada dependencia funcional se convierte en un esquema (las que tienen igual lado izquierdo se juntan)
    1. Es decir, si XYZ -> U, entonces se crea un esquema R(XYZU).
3. Si ninguno de los esquemas resultantes contiene una clave se agrega uno con los atributos de alguna clave.
    1. Es decir, se agrega un esquema relacional donde sus atributos son los de una clave.
4. Eliminar esquemas redundantes: Si alguno de los esquemas resultantes esta contenido totalmente en otro, eliminarlo.
5. Ejemplo:
    1. FACULTAD (L, N, M, I, E, N1) F= {L->N, LM->I, LME->N1} CLAVE: {L,M,E}
        1. F es minimal, entonces tenemos que J = {R1(LN), R2(LMI), R3(LMEN1)}
        2. Como R3 tiene contenida la clave, entonces no se agrega R(LME).

45. Qué es una entidad débil?
    1. Una entidad débil es una entidad de la cual su existencia depende de otra entidad.
46. Qué es una interrelación? Definir los tipos de interrelaciones
    1. Una interrelación es un vínculo (o una asociación) entre entidades.
    2. Interrelaciones unarias: una entidad se vincula consigo misma.
    3. Interrelaciones binarias: una entidad se vincula con otro tipo de entidad.
    4. Interrelaciones ternatias: 3 entidades se relacionan entre sí.
47. Qué es la participación total en las interrelaciones? Y la parcial?
    1. Una interrelación es total si todas las instancias (?) de la entidad participan en la interrelación.
    2. Una interrelación es parcial si puede haber instancias (?) de la entidad que no participen en la interrelación. En las relaciones no ternarias, puede ocurrir que no haya ningún participante.
48. Para qué sirven los atributos identificatorios en las interrelaciones?
    1. Sirven para poder identificar las interrelaciones, y así poder tener varias instancias de las mismas. Por ejemplo, un alumno está interrelacionado con materia. Si no agrego un atributo identificatorio a materia, entonces esa interrelación sólo puedo existir a lo sumo una vez, imposibilitando múltiples cursadas.
49. Cuándo un índice es clustered?
    1. Un índice es clustered cuando los registros (filas) de una tabla están almacenados en el mismo orden que el índice.
        1. Una tabla puede tener sólo un índice clustered.
        2. Tiene la ventaja de que los valores parecidos se almacenan cerca.
        3. Un ejemplo son los árboles B+ clustered.

    2. Un índice es no-clustered cuando las filas de las tablas no se ordenan necesariamente en el mismo orden que el índice.
50. Cuándo un índice es denso?
    1. Un índice denso es archivo de pares <keys, punteros> a registros de filas en un archivo de datos, donde cada clave en archivo de índices está asociado a un registro particular en un archivo de datos.
    2. Un índice no denso es un archivo que contiene pares <key, punteros> por cada bloque en los archivos de datos, conservando así espacio.
51. Cuándo un índice es primario?
    1. Se llaman índices primarios (!= claves primarias) a aquellos índices que contienen todos los registros completos de los archivos. En caso de tener sólo rids se los llaman índices secundarios.
    2. Un índice primario no tiene duplicados y contiene los registros de los archivos junto al índice (los datos de la tabla están con el índice).
    3. Un índice secundario es un índice que no es primario (el archivo de datos está separado del índice) y que puede tener duplicados.
52. Qué es la selectividad de un predicado?
    1. La selectividad de un predicado p es la proporción de tuplas de una relación que satisfacen el predicado.
53. Mencione dos propiedades de la selectividad en el álgebra relacional
    1. Cascada:
        1. Selección de selección puede resumirse por una selección con ands.
    2. Conmutatividad:
        1. Pueden intercambiarse los órdenes de la aplicación.
54. Mencione dos propiedades de la proyección en el álgebra relacional
    1. Cascada:
        1. La proyección en cascada puede resumirse como la proyección de la intersección de los atributos en las proyecciones.
    2. Conmutatividad de la proyección sobre la selección
        1. Pueden intercambiarse las operaciones siempre y cuando la proyección incluya los atributos dentro de la condición de la selección.
55. Mencione dos propiedades del join en el álgebra relacional
    1. Conmutatividad
        1. El join (producto cartesiano) de dos relaciones es “conmutativo” (R x S equivalente S x R), el orden de los atributos cambia.
        2. El join (producto cartesiano) es conmutativo respecto de la selección
56. Mencione tres heurísticas aplicables
    1. Realizar primero los joins más selectivos
        1. Para poder reducir la cantidad de tuplas que se propagan hacia otras operaciones.
    2. Mover las proyecciones y las selecciones lo más cerca de las hojas posibles
        1. Para reducir el tamaño de las tuplas que se propagan hacia otras operaciones.

    2. Siempre y cuando no interrumpa otras operaciones y no quite índices que podrían mejorar el costo.
3. Reemplazar los productos cartesianos seguidos de selecciones por joins.
    1. Para reducir la cantidad de tuplas en la salida.

57. Mencione tres componentes del RDBMS
    1. El procesador de consultas
    2. El módulo de recuperación
    3. El módulo de logging
58. Describir el método de recuperación Undo
    1. Deshace las transacciones incompletas. Para cada transacción incompleta, escribe un <ABORT T> al final del log y lo flushea.
    2. En cada log, se guarda <T, X, ValorAnterior>.
    3. Reglas:
        1. Los registros <T,X,V> se escriben antes de que se escriban los valores en disco.
        2. El <COMMIT T> se escribe en el log después de que todas las operaciones a disco se hayan realizado.
    4. Procedimiento:
        1. Comienzo a leer el log desde la última entrada hacia la primera, poniendo en disco el valor anterior para cada ítem de la base de datos de las transacciones incompletas.
59. Describir el método de recuperación Redo
    1. Rehace las transacciones que hicieron commit. Para cada transacción incompleta escribe un <ABORT T>.
    2. En cada log se guarda <T, X, ValorNuevo>
    3. Reglas:
        1. Los registros <T,X,V> se escriben en el log antes de que el nuevo valor se escriba a disco.
        2. El registro <COMMIT T> se escribe en el log antes de que de cualquier elemento modificado por T se haya escrito en disco.
    4. Procedimiento:
        1. Se comienza a leer el log desde el registro más antiguo hasta el más reciente, re-haciendo las transacciones commiteadas.
60. Describir el método de recuperación Undo/Redo
    1. Aplica Undo y luego Redo, es decir, se deshacen las transacciones incompletas, y se rehacen las transacciones completas. Para cada transacción incompleta, se escribe un <ABORT T>.
    2. En cada log se guarda, <T, X, ValorAnterior, ValorNuevo>
    3. Procedimiento:
        1. Aplicar una política y luego la otra.
61. Qué diferencia hay entre el sistema de logging con checkpoint quiescente y no quiescente?

1. En el sistema de logging quiescente no se toman nuevas transacciones hasta que no se hayan completado las transacciones activas al momento del checkpoint, mientras que en un sistema no quiescente, se toman transacciones nuevas.

62. Cómo se realiza Undo con checkpoint quiescente? y no quiescente?
    1. Checkpoint quiescente:
        1. Cuando se comienza el checkpoint, se dejan de aceptar transacciones nuevas. Se espera a que todas las transacciones hayan sido completadas y se agrega un <CKPT>. Se vuelve luego a aceptar nuevas transacciones. Se realiza Undo desde los logs más recientes hasta encontrar un <CKPT>
    2. Checkpoint no quiescente:
        1. Se escribe <Start CKPT(T1,T2,…,Tk)> en el log, con las transacciones activas hasta el momento y se continúan aceptando transacciones. Cuando las transacciones terminan, se agrega un <End CKPT>. Cuando se aplica Undo, no se debe leer más allá del <START CKPT> si se encuentra un <END CKPT>, caso contrario, leer hasta <START CKPT>.
63. Cómo se realiza un Redo con checkpoint no quiescente?
    1. Inicio:
        1. Escribir <Start CKPT(T1,T2,...,Tk)> en el log, y efectuar un flush.
        2. Esperar a que todas las modificaciones realizadas en los buffers por transacciones ya commiteadas al momento del Start CKPT sean escritas a disco.
        3. Escribir <End CKPT> en el log y efectuar un flush.
    2. Pasos:
        1. Encuentro un <END CKPT>. Indica que las transacciones que hicieron COMMIT antes del START CKPT tienen sus cambios en disco, luego las ignoro. Rehacer las transacciones que hicieron COMMIT, y que comenzaron luego del START CKPT, o que estuvieron activas al momento del START CKPT, desde el START más antiguo de dichas transacciones.
        2. Encuentro un <START CKPT...> (no hay un <END CKPT> debido a un crash). Ubicar el <END CKPT> anterior y su correspondiente <Start CKPT(S1,S2,...,Sk)> y rehacer todas las transacciones comiteadas que comenzaron a partir de ese START CKPT o están entre las Si , desde el START más antiguo de dichas transacciones.
64. Cómo se realiza Undo/Redo con checkpoint no quiescente?
    1. Inicio:
        1. Se escribe <Start CKPT(T1,T2,…,Tk)> con las transacciones activas hasta el momento.
        2. Escribir en disco todos los valores en los buffers.
        3. Escribir <End CKPT> en el log y efectuar un flush.
    2. Pasos:

    1. Para las transacciones incompletas, se aplica Undo hasta el start más antiguo de ellas.
    2. Para las transacciones completas, si leyendo desde el último registro del log, encontramos un registro <End CKPT>, sólo será necesario rehacer las acciones efectuadas desde el correspondiente registro Start CKPT en adelante.
65. Para qué queremos tipos de datos complejos (objetos)?
    1. Para modelar de manera más intuitiva las aplicaciones.
    2. Permite dominios no atómicos como los conjuntos.
66. Mencionar dos extensiones del modelo de objetos en SQL. Ejemplifique.
    1. Tipos estructurados (user defined type)
        1. Se crean tipos, como si fueran tablas, con su nombre y atributos.
        2. Los tipos estructurados pueden utilizarse como tipos en la creación de tablas.
        3. Tienen sus constructor functions, que deben definirse y tienen como objetivo funcionar como constructores del user defined type que construyen.
        4. Los tipos tienen herencia, simple o múltiples, es decir, heredan los atributos de sus padres.
        5. Ej:
            1. create type Name as
                1. (firstname varchar(20),
                2. lastname varchar(20))
                3. final
    2. Object identity and reference types
        1. Las tablas pueden incluir referencias entre sus atributos a otras tablas, utilizando identificadores de las tablas a referenciar, que pueden ser generados por sistema o por usuario.
        2. Permiten evitar joins.
67. Describa las BD no relacionales de clave-valor. Ejemplifique.
    1. Son bases de datos que funcionan como un diccionario.
    2. Se opera a través de una clave.
    3. Cada valor asociado a una clave puede tener diferente tipo, y puede cambiar libremente.
        1. Local[‘Pepe:1234’] = {domicilio: “9 de julio 1919”}
68. Describa las BD no relaciones column-family. Ejemplifique.
    1. Los datos se guardan en column families como filas que pueden tener varias columnas asociadas con una key para cada fila.
    2. Las column families agrupan filas con distintas columnas.
    3. Las column families representan datos que generalmente se acceden al mismo tiempo.
    4. Se puede pensar a las column families como un contenedor de filas, las cuales tienen diferentes columnas. Las filas pueden agregarse en cualquier momento.

69. Describa las BD no relacionales column store. Ejemplifique.
    1. Almacenan los datos en columnas (a diferencia de las otras BD que lo hacen por filas).
    2. Fila: (nombre1, teléfono1, domicilio1), (nombre1, teléfono2, domicilio2)
    3. Columna:
        1. nombre1, nombre2
        2. teléfono1, teléfono2
        3. domicilio1, domicilio2
70. Describa las BD no relacionales de documentos. Ejemplificar.
    1. Cada entrada se llama “documento”. Son datos semi-estructurados, definidos en JSON o XML. Los documentos pueden ser diferentes. Las estructuras de los documentos pueden cambiar en cualquier momento.
    2. Se pueden realizar consultas sobre los valores de los documentos.
        1. Documento empleado1 = {nombre: “Juan”}
        2. Documento empleado2 = {nombre: “Pepe”, domicilio: “9 de julio”}
71. Describa las BD no relacionales de grafos. Ejemplifique.
    1. Se guardan relaciones que conforman grafos.
    2. Nodo -> relación -> Nodo.
    3. Libro -> autor -> Juan Perez
72. Qué son las propiedades BASE?
    1. Basic Availability
        1. Cada solicitud garantiza una respuesta: ejecución exitosa o fallida.
    2. Soft-sate
        1. El estado del sistema puede cambiar con el tiempo, a veces sin ninguna entrada (por consistencia eventual).
    3. Eventual consistency
        1. La BD puede ser momentáneamente inconsistente pero será consistente con el tiempo.
73. Cuáles son los niveles de aislamiento?
    1. Nivel 0: T no sobrescribe los dirty reads de las transacciones de mayor nivel.
    2. Nivel 1: No hay lost updates
    3. Nivel 2: no hay lost updates ni dirty reads.
    4. Nivel 3: Nivel 2 + repeatable reads.
74. En qué consiste la operación map-reduce?
    1. Map aplica una función a cada par (clave,valor), generando una nueva colección
    2. Reduce aplica una función de agregación (ej: suma) a la colección generada por map, para devolver el resultado final.
75. En qué consiste realizar sharding?
    1. Sharding es la operación de fragmentar la BD en fragmentos denominados shards y distribuirlos a través de los servidores disponibles. Conceptualmente, los shards comparten esquemas y colectivamente representan el total del dataset.

    2. El Sharding se realiza cuando la cantidad de datos no es posible de ser almacenada en un sistema.
76. Qué dos tipos de sistemas de réplicas se puede tener?
    1. Master-Slave
        1. Se tiene un nodo master, donde se realizan lecturas y escrituras. Las escrituras son luego propagadas hacia los slave. Los slaves sólo responden a lecturas.
    2. Peer to Peer
        1. Todos los nodos poseen el mismo nivel de jerarquía y pueden recibir escrituras y lecturas.
77. Qué dice el teorema de CAP?
    1. Dice que no se pueden tener Consistency, Availability y Partition tolerance al mismo tiempo.
78. Qué es data mining?
    1. Es la extracción de patrones o información interesante de grandes bases de datos.
79. Qué es el clustering?
    1. Clustering es el análisis de clusters (extracción) de un conjunto de datos.
    2. El clustering busca maximizar la diferencia entre clusters y minimizar la diferencia entre elementos del cluster.
80. Ejemplos de algoritmos de clustering
    1. Clustering Jerárquico (genera clusters con jerarquía)
        1. Agrupamiento aglomerativo
            1. Comienza con observaciones singulares y las une para formar elementos superiores en la jerarquía.
        2. Agrupamiento divisivo
            1. Comienza con un grupo con todas las observaciones, y se divide en subgrupos
    2. Clustering no jerárquico
        1. Método de particionamiento: construir una partición de la base de datos D de n objetos en k clusters.
81. Ejemplos de algoritmos no supervisados
    1. Clustering, que consiste en la extracción de clases o grupos de los datos. Pueden ser jerárquicos o no jerárquicos.
    2. Reglas de asociación
        1. La idea es generar reglas basada en patrones o correlaciones entre bases de datos o ítems de las mismas.
82. Describir el modelo dimensional
    1. El modelo dimensional esta basado en hechos, dimensiones y medidas.
        1. Los hechos son colecciones de ítems de datos y datos de contexto. Cada punto de entrada en la tabla de hechos se conecta a una dimensión.
        2. Una dimensión es una colección de miembros o unidades del mismo tipo.
            1. Ejemplo: tiempo, geografía, cliente, vendedor.

        2. Miembros posibles: meses, trimestres, años (en la dimensión tiempo).
    3. Medidas: es un atributo numérico que representa la performance o comportamiento del negocio relativo a la dimensión. (Creo que son los valores de las dimensiones)

83. Dar dos tipos comunes del modelo dimensional
    1. Star
    2. Snowflake
84. Cuáles son los distintos tipos de fragmentación que existen?
    1. Fragmentación vertical
        1. En la fragmentación vertical, los fragmentos son partes más pequeñas del esquema original, junto a una superclave.
        2. El conjunto de tuplas es fragmentado según una condición sobre los atributos (se aplica una selección).
    2. Fragmentación horizontal
        1. En la fragmentación horizontal, los fragmentos creados contienen tuplas originales del esquema a fragmentar (se aplica una proyección).
    3. Fragmentación mixta
        1. Es una combinación entre la fragmentación vertical y la horizontal (coincide a la aplicación de un select seguido de una proyección).
85. Cómo es el proceso de creación de una BD?
    1. Análisis de requerimientos
    2. MER (modelo de entidad relación), que nos permite realizar un modelo o una abstracción de alguna situación de interés de nuestro problema. Nos permite modelar los objetos o conceptos y sus características y relaciones.
        1. Está compuesto por:
            1. Entidades: modelan conceptos u objetos los cuales queremos perdurar.
            2. Atributos: son las propiedades que describen las entidades. Es la información que queremos preservar de las mismas.
            3. Interrelaciones: modelan vínculos entre las entidades.
            4. Reglas del dominio: restricciones.
    3. MR (Modelo Relacional), donde las entidades son representadas como esquemas relacionales, los atributos como atributos dentro de las relaciones, y las interrelaciones como claves foráneas o relaciones.
    4. Normalización: Las relaciones pasan por el proceso de normalización para evitar problemas de redundancia.
    5. Diseño físico: se establecen los índices, los tipos de datos, etc.
86. Explicar el costo de acceder por igualdad a registros de una BD utilizando un índice hash.
    1. Para acceder a un conjunto de valores utilizando un índice hash, es necesario aplicar la función hash dado el valor a buscar (costo 0) para conseguir el bucket relacionado a ese valor. Como el índice hash está compuesto por un conjunto de

buckets, con bucket siendo un conjunto de bloques, debemos leer todos los bloques del bucket en búsqueda de los rid que coinciden con el valor buscado (esto tiene costo MBxBi, es decir, máxima cantidad de bloques por bucket del índice i). Luego, debemos acceder a un bloque por cada rid para recuperar el registro (T' tuplas coincidentes). El costo total es: MBxBi + T' .

87. Explicar el costo de acceder por igualdad a registros de una BD utilizando un sorted file.
    1. Los sorted files contienen los registros ordenados según un atributo. Se busca el primer registro que cumpla con la condición (log2(Br)) y luego se procede a leer todos los bloques de los registros que cumplan con la misma (techo(T'/FBr)). El costo total es: log2(Br) + techo(T'/FBr).
88. Explicar el costo de acceder por igualdad a registros de una BD utilizando un heap file.
    1. El heap file es un archivo que contiene los registros de manera desordenada. La búsqueda por igualdad es: Br. Es decir, todos los bloques de la relación.
89. Explicar el costo de acceder por igualdad a registros de una BD utilizando un índice con forma de árbol b+ unclustered.
    1. Para acceder a los registros que cumplan con la condición de igualdad, es necesario leer los bloques de cada nodo del árbol b+ de índices hasta llegar a la hoja que corresponda con el primer índice que coincide con la búsqueda. Esto se hace con costo Xi.
    2. Como los índices unclustered no determinan el orden de las tuplas en los archivos, hay que leer los bloques de índices necesarios (hojas del árbol b+) para tomar todos los rid relacionados con las tuplas que coinciden con la condición. Esto se hace en costo techo(T'/FBi).
    3. Para cada rid leido de las hojas, es necesario leer un bloque. Esto se hace en costo T'.
    4. El costo final es: Xi - 1 + techo(T'/FBi) + T'
90. Explicar el costo de acceder por igualdad a registros de una BD utilizando un índice con forma de árbol b+ clustered.
    1. Para acceder a los registros que cumplan con la condición de igualdad, es necesario leer los bloques de cada nodo del árbol b+ (buscando el primer índice que coincida con la condición) hasta llegar a una hoja (que contiene el rid del primer coincidente), esta operación se hace con costo Xi (altura del árbol). Una vez obtenido el rid del primer registro coincidente, se leen los bloques correspondientes a todas las tuplas que coinciden con la condición, esto se hace con costo techo(T'/FBr). Costo total: Xi + techo(T'/FBr).
91. Dar dos optimizaciones algebráicas para la selección. Ejemplificar.
    1. Conmutatividad de selecciones: σc2(σc1(R1)) equivalente σc1(σc2(R2))
    2. Conmutatividad de la selección y el producto cartesiano o junta: σc1(R x S) equivalente σc1(R) x σc1(S)
    3. Dada una sucesión de selecciones (posiblemente obtenidas de la descomposición por la propiedad de cascading de la sucesión del árbol canónico), podemos bajar o subir las selecciones en el árbol según convenga. Considerando además la conmutatividad de las selecciones en el producto

cartesiano o junta, podemos utilizar la conmutatividad para mover las selecciones por juntas, hasta las raices, permitiendo reducir considerablemente la cantidad de tuplas antes de realizar juntas.

92. Dar dos optimizaciones algebráicas para la proyección. Ejemplificar
    1. Conmutatividad respecto al producto cartesiano o junta: πL(R x S) es equivalente a πL1(R) x πL2(S), siendo L = L1 U L2.
    2. Conmutatividad respecto a la selección con la proyección: σ1(π1(R)) es equivalente π1(σ1(R)). Siempre y cuando los atributos proyectados sean parte de las selecciones.
    3. Podemos utilizar estas propiedades para mover proyecciones lo más bajo posible en el árbol, pasando por selecciones hasta raíces o por productos cartesianos, reduciendo el tamaño de las tuplas.
93. Dar dos optimizaciones algebráicas para el join o producto cartesiano. Ejemplificar.
    1. Conmutatividad del producto cartesiano o junta: R x S equivalente S x R
    2. Asociatividad del producto cartesiano o junta: (R x S) x T es equivalente a R x (S x T).
    3. Junto a estadas dos propiedades, es posible, partiendo del árbol canónico, juntar productos cartesianos que compartan atributos, para poder convertirlos en joins, utilizando alguna selección posible.
94. Describa el algoritmo BNLJ.
    1. Por cada segmento de B - 2 bloques de Br
        1. Por cada bloque de S
            1. Para cada tupla del segmento de bloque de r y del bloque s
                1. Agrego <r, s>
    2. El costo es Br + Bs * techo(Br/B-2)
95. Describa el algoritmo INLJ
    1. Necesita índices
    2. Para cada tupla r de R
        1. Para cada tupla s de S | ri = si (buscada por índice)
            1. Agregar <r,s> al resultado
    3. Costo: Br + Tr * (buscar para la tupla tr index en el índice de S + buscar valores apuntados por las entradas de índice)
    4. Recorre una de las dos relaciones y busca por índice en la otra.
96. Describa el algoritmo SMJ
    1. Se ordenan las dos relaciones participantes por el atributo de junta.
    2. Luego se realiza un emerge Br + Bs
    3. Sólo podría ser bueno si estuvieran ordenadas.
97. Sea phi un enunciado de primer orden sobre un vocabulario tao. Es siempre posible decidir si phi vale en todas las tao-estructuras finitas (bases de datos)? Enunciar el teorema general que responde esta pregunta.
    1. No es verdadero, si existiera un proceso de decisión para decidir si un enunciado de primer orden Phi sobre un vocabulario Tau vale en todas las Tau-estructuras finitas, se podría usar el mismo sobre NOT(Phi) para decidir si Phi vale en al

menos una Tau-estructura finita (si es finitamente satisfacible). Pero esto contradice al teorema de Trakhtenbrot.

98. En términos de estructuras finitas, qué es un query?. En particular, qué es un first order query?
    1. Una query m-aria sobre σ-estructuras es un mapeo Q que asocia con cada estructura U un subconjunto de A^m, tal que Q es cerrado bajo isomorfismo: si U (igual, equi) B vía el isomorfismo h: A->B, entonces Q(B) = h(Q(U)).
    2. Decimos que Q es definible en una lógica L si existe una fórmula ϕ(x1, …, xm) de L en el vocabulario σ tal que para todo U:
        1. Q(U) = {(a1, .., am)} e A^m | U |= ϕ(x1, …, xm)}
    3. Una query es una first order query cuando L es una lógica de primer orden.
99. Exhibir un ejemplo de un query que no es first order, en qué nociones y construcciones de la lógica se basan las pruebas conocidas que un query no es first order?
    1. El poder expresivo de la lógica de primer orden no puede expresar “counting properties”. Un ejemplo de esto es la cláusula sería, si tenemos un grafo, es par la cantidad de nodos?.
    2. La herramienta principal para demostrar expresabilidad de queries en primer orden es la teoría de juegos de Ehrenfeucht-Fraïssé.
100. Cuáles lógicas se invocan para tratar con queries que no son first order, y qué relación tienen estas lógicas con clases de complejidad?
    1. Para tratar queries que no son first orders, la bibliografía utiliza variaciones de la lógica de segundo orden (y de primer orden). Se especifica un subconjunto de la lógica de segundo orden, basado en fórmulas, específicamente las *existential second-order logic* ($\exists$SO) y universal *second-order logic* ($\forall$SO la negación de las existencial). Según el teorema de Fagin, los problemas NP son aquellos que pueden ser expresados en lógicas existenciales de segundo orden. Es decir, Fagin provee una caracterización lógica pura de la clase de complejidad NP.