TRANSACCIONS - MOTIVACIÓ NOSQL

Aquesta sessió té dues parts, tal com indica el títol que hi hem posat. La primera fa un repàs als mecanismes de què disposen els SGBD relacionals per gestionar les transaccions i garantir les propietats que els són inherents, així com l'impacte que aquests mecanismes poden tenir en el rendiment de la BD. A la segona posarem de manifest que hi ha situacions en què es pot prescindir d'alguns d'aquests mecanismes, reflexió que va donar lloc a l'aparició del moviment NOSQL.

Transaccions. La diapositiva 4 recorda les propietats de les transaccions en l'entorn relacional. Recordeu que una transacció és una seqüència de sentències de consulta, actualització o manipulació de la BD que implementen una operació lògica des del punt de vista de l'aplicació o l'usuari.

- Per això necessitem que siguin **A**tòmiques: o bé s'executen completament o bé, si per alguna raó no poden acabar bé, s'ha de desfer tot el que calgui per tal que la BD quedi en el mateix estat que estava abans d'iniciar la transacció.
- D'altra banda, la transacció ha de començar en un estat **C**onsistent i també ha de deixar la BD en un estat consistent.
- Per a un millor aprofitament dels recursos i una millor experiència d'usuari, volem poder executar diverses transaccions en paral.lel però de manera que no s'interfereixin les unes a les altres, preservant l'**I**solament necessari entre elles.
- Finalment, volem que els efectes de les transaccions que acaben **D**urin per sempre, independentment de possibles accidents o avaries que afectin els sistemes sobre els quals tenim funcionant la BD.

No tractarem en aquesta unitat sobre la consistència, la responsabilitat de la qual recau en les transaccions però que ha de ser controlada en la mesura del possible per l'SGBD, a base de restriccions i assercions.

Comencem parlant de **l'aïllament** i, més endavant, parlarem d'atomicitat i durabilitat. Quan dues transaccions s'executen en paral.lel, poden produir-se interferències entre elles que impliquin que, depenent si unes operacions d'una transacció s'executen abans o després d'unes operacions de l'altra transacció, el resultat de l'execució sigui diferent. Les interferències que es poden produir són (mireu, si us plau, el fitxer Excel "ExemplesAïllament" que teniu en el LearnSQL):

- Actualització perduda. Sota el títol "Lost Update" podeu observar que la T1 llegirà una informació diferent a la que ha escrit.

- Lectura no confirmada. Sota el títol "Read Uncommitted" podeu observar que T2 llegeix una informació que després és revocada per T1.

- Lectura no repetible. Sota el títol "Unrepeatable Read" podeu observar que T1 obté dues informacions diferents fent la mateixa lectura.

- Valors fantasma. En els tres casos anteriors, la transacció afectada experimenta efectes no volguts en informacions que ella mateixa crea o que ja existien en començar-se a executar. També pot ser un problema que, durant l'execució d'una transacció T1, apareguin nous valors creats per una altra T2. Per exemple T1 recorre una taula i T2 fa una inserció a la mateixa taula.

L'estàndard defineix diferents nivells d'aïllament que impedeixen que es produeixin més o menys interferències. Per garantir l'aïllament, la solució tradicional és fer esperar una transacció fins que es pugui executar sense interferir. Per decidir quina i quan s'ha d'esperar, es defineixen bloquejos sobre les dades, que poden ser en exclusiva (X) o amb possibilitat de compartir (S). Una transacció fer un bloqueig X sobre una dada si la dada està lliure i pot fer un bloqueig S si la dada està lliure o té altres bloquejos S. Cada nivell d'aïllament requereix que les transaccions facin o no bloquejos abans de poder efectuar operacions; si la transacció no pot fer el bloqueig requerit, ha d'esperar. Els nivells i bloquejos són aquests:

- Read Uncommitted. Aquest nivell, que impedeix que es produeixin actualitzacions perdudes, requereix que la transacció bloquegi X una dada abans de modificar-la. Per llegir no requereix cap bloqueig. Fixeu-vos que, si estem en aquest nivell, en el primer exemple T2 no pot modificar la dada x perquè T1 ja la té bloquejada prèviament en exclusiva i T2 s'ha d'esperar que T1 acabi.

- Read Committed. Aquest nivell, que impedeix, a més, que es produeixin lectures no confirmades, requereix, a més, que la transacció bloquegi S una dada abans de llegir-la i la desbloquegi després de la lectura. Fixeu-vos que, si estem en aquest nivell, en el segon exemple T2 no pot llegir la dada x perquè T1 la té bloquejada en exclusiva i T2 s'ha d'esperar que T1 acabi (avorti, en aquest cas). Si estiguéssim en el nivell Read Uncommitted T2 faria la lectura sense bloquejar i es produiria la interferència.

- Repeatable Read. Aquest nivell, que impedeix, a més, que es produeixin lectures no repetibles, requereix un protocol de bloqueig en dues fases: la transacció va bloquejant dades X per modificar o S per llegir i, quan comença a desbloquejar, ja no pot tornar a bloquejar una altra vegada. Fixeu-vos que en el tercer exemple, si no estem en aquest nivell, quan T2 vol modificar la dada x ho pot fer perquè T1 no la té bloquejada. En aquest nivell, en canvi, T2 mantindrà el bloqueig S sobre la dada x fins després de la darrera lectura, cosa que impedirà que T2 la pugui modificar pel mig.

- Serialitzable. Aquest nivell impedeix totes les interferències. Requereix que els bloquejos es facin sobre les taules/índexs sencers.

Si només tenim en compte les interferències, pensarem que és una ximpleria definir quatre nivells d'aïllament i que el que convindria seria que les transaccions s'executessin sempre amb nivell d'aïllament serialitzable. Hi ha, però, un altre factor molt important a tenir en compte: com més alt és el nivell d'aïllament, més bloquejos es fan, més esperes s'han d'imposar a les transaccions i, per tant, menys paral.lelisme i, en conseqüència, menys rendiment. És per això que es defineixen els nivells per deixar que cadascú decideixi en funció de:

- quines interferències es poden produir segons les operacions de les transaccions que van en paral.lel
- quina importància té que es produeixin o no
- quina combinació de rendiment i precisió/correctesa és millor

Fins i tot podem desactivar completament els bloquejos si les circumstàncies no els fan necessaris. Per exemple, si totes les transaccions són read-only o si no hi ha transaccions en paral.lel.

Per acabar, hem d'esmentar un altre efecte secundari dels bloquejos: les abraçades mortals. Si hi ha diverses transaccions en paral.lel bloquejant i fent-se esperar les unes a les altres, pot produir-se una situació de deadlock, em què totes les transaccions estan esperant que una altra desbloquegi alguna cosa. És costós preveure si un bloqueig produirà una situació d'aquest estil o si una transacció en espera s'acabarà reactivant o no. Els SGBD resolen aquesta situació forçant que avorti una transacció que porta més d'un determinat temps esperant, per si de cas estava en un deadlock. Cal no definir aquest temps massa llarg (per no esperar massa a desfer una abraçada mortal) ni massa curt (per no fer avortar una transacció que, amb una mica més d'espera, hauria continuat).

Pel que fa a **l'atomicitat** i la **durabilitat**, el SGBD dóna suport a aquestes propietats bàsicament amb dos mecanismes: el dietari (log) i les còpies de seguretat (backup). El dietari conté el registre de totes les operacions de modificació que s'han fet a la BD des de la darrera còpia de seguretat, a més de les indicacions d'inici/final de transacció i encadenaments entre les operacions de la mateixa transacció, ja que les operacions es van anotant a mesura que es van executant i les transaccions s'executen en paral.lel i, per tant, les operacions de les diferents transaccions queden barrejades en el dietari. El dietari serveix per:

- desfer (undo) les operacions que ha fet una transacció que avorta (rollback).
- refer (redo) les operacions de les transaccions confirmades que s'ha executat després de la darrera còpia de seguretat, en cas de necessitar reconstruir la BD

després d'una avaria Hw. Aquesta reconstrucció començarà a partir de la còpia i es completarà reexecutant les transaccions confirmades.

- desfer/refer (undo/redo) transaccions segons si havien quedat a mitges o si havien acabat després d'una caiguda del sistema per error de la BD, el SO, problema elèctric, ... i així restaurar la BD. La raó per la qual pot ser necessari refer transaccions és perquè el SGBD fa servir buffers a memòria per minimitzar escriptures a disc i la informació dels buffers que encara no s'ha enregistrat al disc es perd en cas de caiguda. El que sí que s'enregistra immediatament és l'entrada corresponent al dietari quan s'executa una operació.

A la diapositiva 12 hi teniu l'esquema genèric de l'arquitectura d'un SGBD relacional des del punt de vista de la gestió de transaccions. El gestor de transaccions rep les peticions de les transaccions i controla deadlocks, permisos, ... El planificador gestiona l'aïllament i bloquejos i li passa les peticions de lectura i escriptura de les transaccions al gestor de recuperació. Aquest llegeix i escriu als buffers per dur a terme les lectures i escriptures que demanen les transaccions. Si necessita accedir a un bloc que no està als buffers, li demana al gestor de cache que el vagi a buscar a disc (fetch) i quan decideix que una pàgina de memòria s'ha d'enregistrar al disc demana que es faci flush. També va enregistrant les operacions al dietari, també a través dels buffers. Per tal que aquesta arquitectura mantingui la consistència i sigui capaç de garantir l'atomicitat i la durabilitat, s'han de fer les escriptures seguint una sèrie de normes, les més importants les teniu a la diapositiva 15, on hi faltaria que quan una transacció confirma tot allò que ha fet ha de quedar en algun dispositiu de memòria estable. Com a mínim s'ha d'enregistrar la part de dietari que contés les operacions d'aquesta transacció.

Motivació NOSQL. Com acabem de veure, la gestió de transaccions garantint les propietats ACID implica una complexitat i recursos que poden afectar el rendiment de la BD i la capacitat de suportar segons quin nivell de velocitat en la demanda de modificació d'informació. Però aquestes limitacions no es van presentar durant algunes dècades, durant les quals qualsevol sistema que necessitava una BD, incorporava una BD relacional, tot i que això significa incorporar tots els components d'un SGBD d'aquestes característiques amb tota la complexitat inherent.

Podem imaginar, però, situacions en què es podria simplificar l'arquitectura del sistema. En teniu alguns exemples a les diapositives 21, 22 i 23.

Aquest mateix raonament està en l'inici de NOSQL, que planteja situacions en què no cal garantir (totes) les propietats ACID (estrictament) però en canvi requereixen un temps de resposta que una BD relacional no pot assolir degut a la complexitat que té. A partir d'aquesta reflexió, es posa en qüestió l'adopció de BD relacionals universalment com s'havia fet fins al moment i comencen a sortir models que s'adapten a les noves situacions plantejades.