Transacties en multi-user gebruik

Probleem:

* Onderscheid tussen single-user en multi-user omgevingen
* Confilct ontstaan wanneer meerdere gebruikers dezelfde gegevens willen lezen/schrijven
  + Concurrency control is vereist om gelijktijdigheid te regelen
* Raadplegen van data uit meerdere tabellen

Wanneer zinvol?:

* Als bepaald gegeven uit meerdere tabellen geschrapt moet worden
* Als gebruiker zich vergist heeft bij aanpassingen

Single-user vs multi-user

* Toegang tot dezelfde data:
  + single-user: 1 gebruiker heft toegang tot het systeem
  + multi-user: meerdere gebruikers hebben gelijktijdig toegang tot het systeem en kunnen dezelfde gegevens willen lezen of wijzigen
* conflicten:
  + single-user: conflicten zijn zeldzaam
  + multi-user: conflicten kunnen ontstaan bij gelijktijdige pogingen om dezelfde gegevens te wijzigen
* concurrency control (gelijktijdigheid):
  + single-user: geen complexe maatregelen nodig.
  + Multi-user: cruciaal om conflicten te voorkomen; techniek zoals vergrendeling en transacties-isolatie zijn noodzakelijk
* Toegang tot meerdere tabellen:
  + Single-user: minder complexe databasestructuur mogelijk.
  + Multi-user: goed ontworpen databasestructuur en geoptimaliseerde query’s zijn nodig voor het raadplegen en bewerken van gegevens uit verschillende tabellen.

Transacties

* Transactie: een reeks SQL-instructies die als één eenheid worden behandeld. Ze worden volledig uitgevoerd of volledig geannuleerd bij een fout.
* Autocommit: Elke SQL-instructie wordt automatisch als een transactie behandeld en onmiddelijk toegepast.
* Commit: bevestigt alle wijzigingen binnen een transactie permanent in de database
* Rollback: maakt alle wijzigingen binnen een transactie ongedaan bij een fout of op verozke van de gebruiker.
* Eenmaal gecommit, geen rollback mogelijk: Na een commit zijn de wijzigingen permanent en onomkeerbaar.
* Transactiecyclus: begint met SQL-instructies en eindigt met een commit of rollback. Commit maakt wijzigingen permanent; rollback maakt ze ongedaan
* Laatste actie: Elke transactie eindigt met een commit of een rollback

Wanneer commit of rollback gebruiken?

* Commit: gebruik wanneer wijzigingen correct en permanent moeten worden vastgelegd.
* Rollback: gebruik bij fouten of onjuiste wijzigingen om de database terug te brengen naar de vorige staat.

Specifieke voorbeelden voor rollback

* Meerdere tabellen: als een bewerking op meerdere tabellen mislukt, zorgt een rollback ervoor dat alle wijzigingen ongedaan worden gemaakt
* Gebruikersfouten: als de gebruiker fouten maakt bij het invoeren of bewerken van gegevens, kan een rollback de database herstellen naar de staat voor de transactie

Mogelijke uitzonderingen

* Instructies die de catalogus wijzigen: sommige database-instructies die de catalogus beinvloeden, kunnen beperkingen hebben en van invloed zijn op wanneer commit of rollback wordt uitgevoerd.

Hoe transacties

Impliciete start:

* Na rollback of commit: Elke SQL-instructie wordt automatisch als een transactie beschouwd, die direct wordt gecommitteerd of gerollbackt zonder expliciete commando’s van de gebruiker.

Autocommit:

* Direct effect: Elke SQL-instructie wordt automatisch gecommitteerd na uitvoering, waardoor wijzigingen meteen permanent worden.

Expliciete start:

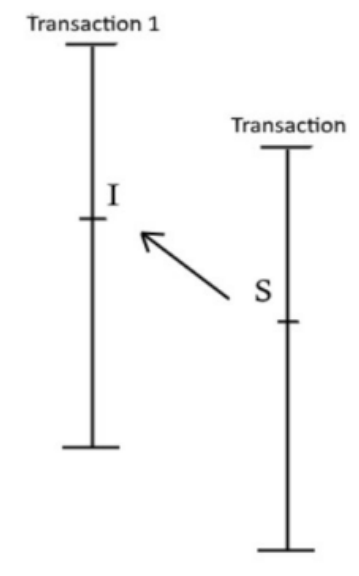
* BEGIN TRANSACTION: gebruiker start en eindigt transactie expliciet met COMMIT of ROLLBACK.

Savepoints

* Tussentijdse momentopnames: deel van een transactie kan ongedaan worden gemaakt zonder de hele transactie terug te draaien.

Problemen multi-user gebruik

Dirty read (uncommitted read):



Gegevens lezen die door een andere transactie zijn gewijzigd maar nog niet gecommitteerd. Als deze transactie wordt teruggedraaid, zijn de gelezen gegevens ongeldig

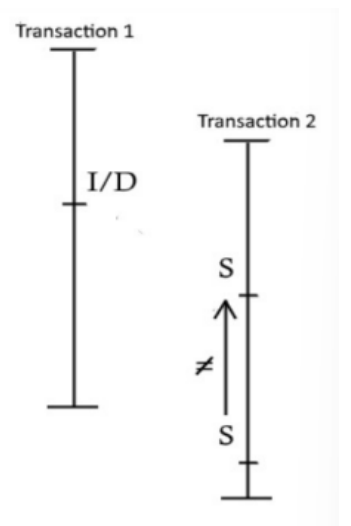
Non repeatable read:

A diagram of a line with a line and a line with a line and a line with a line and a line with a line and a line with a line and a line with a line and

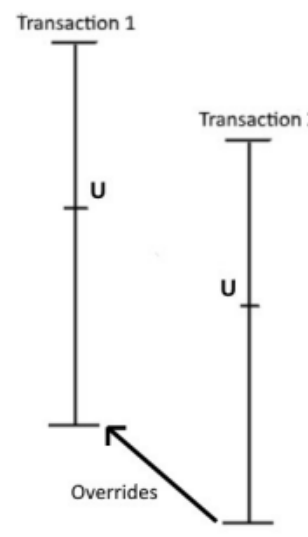
Description automatically generated

Gegevens lezen die tussen twee leesbewerkingen (bv commit) door een andere transactie zijn gewijzigd, wat inconsistenties veroorzaakt.

Phantom read:

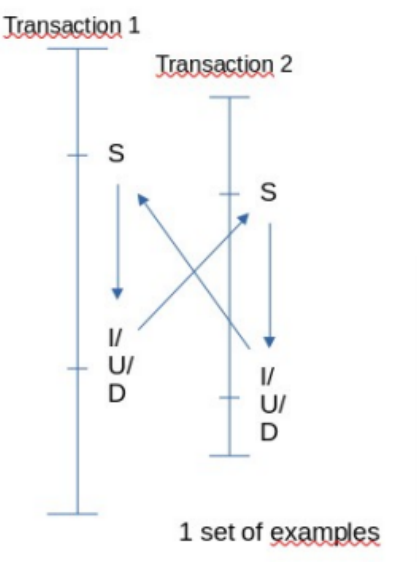
  
een query tweemaal uitvoeren waarbij nieuwe rijen tussen de twee leesbwerkingen zijn toegevoegd door een andere transactie.

Lost update:



Wijzigingen van een gebruiker worden overschreven door een andere gebruiker zonder dat de eerste gebruiker hiervan op de hoogte is.

Serialization anomaly:



De resultaten van parallelle transacties komen niet overeen met de resultaten van een sequentiele uitvoering van dezelfde transacties.

Oplossingen:

* Transacties serieel verwerken: voor systemen met lage gebruikers concurrentie, minimaliseert de noodzaak voor complexe concurrency control.
* Transacties parallel verwerken: voor systemen met hoge gebruikers concurrentie, vereist geavanceerde concurrency control-mechanismen zoals isolatieniveaus en vergrendelingen om problemen te voorkomen.
* Locking van tabellen om conflicten te voorkomen
* Deadlock: indien twee of meerdere gebruikers op elkaar wachten.

Locking mechanisme

* Doel: waarborgt consistentie en integriteit van gegevens in multi-user omgevingen.
* Gelockte rijen: voorkomt gelijktijdige wijzigingen door andere gebruikers tijdens een transactie.
* Lock release: na voltooiing van een transactie wordt de lock opgeheven
* Buffer: locks worden vaak beheerd in RAM voor snelle toegang.
* Granulariteit: verschillende niveaus zoals rij-, pagina en tabelniveau beinvloeden de mate van detail en overhead.

Lock rechten

* SHARE-lock: toegestaan voor leesoperaties door meerdere gebruikers.
* EXCLUSIVE-lock: verhindert andere gebruikers om dzelfde gegevens te lezen of te schrijven.

Deadlock

* Oorzaken: ontstaan door transacties die wederzijds exclusieve bronnen nodig hebben en elkaar blokkeren.
* Oplossingen: deadlock-detectie en herstel door transacties af te breken, of time-outs waarna acties opnieuw worden geprobeerd.

Transacties: isolation level

* Serializable: hoogste isolatieniveau, voorkomt alle gelijktijdige toegang problemen, maar verlaagt de prestaties
  + Maximale isolatie van gebruikers
* Repeatable read: voorkomt wijzigingen tijdens leesbwerkingen, maar kan nonrepeatable reads en phantom reads veroorzaken.
  + Herhaalbaar lezen met exclusieve blokkades voor mutaties
  + Lezen: share blokkades (stopt bij einde transactie)
  + Muteren: exclusieve blokkades
* Read committed: laat leesbwerkingen toe zolang er geen wijzigingen zijn, maar kan phantom reads toestaan
  + Lezen van gecommitteerde gegevens met exclusive blokkades voor mutaties
  + Lezen: share blokkades(stopt bij einde select)
  + Muteren: exclusieve blokkades
* Read uncommitted: Laagste isolatieniveau, staat dirty reads toe, maar verhoogt de gelijktijdige toegang.
  + Lezen van ongecommitteerde gegevens met exclusieve blokkades voor mutaties
  + Lezen: share blokkades (stopt bij einde select)
  + Muteren: exclusieve blokkades (stopt bij einde mutatie)

Gevolgen isolatieniveaus

* vermijd langdurige transacties: minimaliseer duur van transacties om blokkades te verminderen
* isolatieniveaus hebben invloed op concurrency en snelheid:
  + serializable: laagste gelijktijdigheid, lagere snelheid
  + read uncommitted: hoge gelijktijdigheid, risico op inconsistente gegevens door dirty reads