Data management

# 15/02/2024

Bij transactie: als een persoon aanpassingen maakt na het starten van een transactie kan iemand anders die de database gebruikt deze aanpassingen nog niet zien. De aanpassingen zijn pas zichtbaar voor andere gebruikers wanneer de aanpassing ‘gecommit’ zijn. Bij het starten van een transactie staat namelijk nog niet vast of deze aanpassingen vastgelegd of ongedaan gaan worden.

Problemen bij transacties:

1. Dirty read: een andere gebruiker kan de aanpassingen van de transactie al zien, ookal zijn deze aanpassingen nog niet gecommit
2. Nonrepeatable read: A non-repeatable read occurs, when during a transaction, a row is retrieved twice and the values within the row differ between reads.
3. Phantom read: occurs when, during a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first.
4. Lost update: a scenario in which two or more transactions update the same row, but neither transaction is aware of the modification made by the other transaction, and the second change overwrites the first modification.
5. Serialization anomaly: The result of successfully committing a group of transactions is inconsistent with all possible orderings of running those transactions one at a time.

* Oplossing: LOCK TABLE

Probleem bij lock table = deadlock: twee of meerdere gebruikers wachten op elkaar

Oplossing: ISOLATION LEVEL: mate van isolatie van gebruikers. Dit zijn de verschillende niveaus van meest gescheiden (veiligst) tot minder gescheiden:

* Serializable: concurrentie is het laagst, snelheid is het laagst
* Repeatable read
* Read commited
* Read uncommited: concurrentie is het hoogst, moeten weinig op elkaar wachten, kunnen gegevens lezen die enkele momenten later niet meer bestaan

zoek op welke problemen kunnen voorkomen bij welk niveau!

Examples:

* User A runs the same query twice. In between, User B runs a transaction and commits. **Non-repeatable read**: The A row that user A has queried has a different value the second time.
* **Phantom read**: All the rows in the query have the same value before and after, but different rows are being selected (because B has deleted or inserted some). Example: select sum(x) from table; will return a different result even if none of the affected rows themselves have been updated if rows have been added or deleted.
* **Serialization anomaly**: user A adds an extra single value during a transaction, which User B cannot see yet. User B adds a row where they added up the single values to show the total. Both users commit. The result is a table with a wrong total value because the extra value of user A hasn’t been added up to the total value.

­­­coinstraints uitstellen naar het einde van de transactie (geen standaardgedrag, maar kan wel handig zijn). Dit kan enkel wanneer constraint deferrable is

# 22/02/2024

## Pooling

Zonder pooling: iedereen connecteert rechtstreeks op de databank. Met pooling server: iedereen verbindt met de pool en de pool verbindt met de databank.

## Geschiedenis

1. Bestanden: alles werd opgeslagen in bestanden
2. Er zijn beperkingen bij bestanden dus overgeschakeld naar Hiërarchisch model
3. Netwerkmodel
4. Relationeel model
5. Object-geörienteerd model
6. Andere paradigma

### Hiërarchisch model

Eén wortelsegment kiezen en vandaaruit verder bouwen, meerdere wortels kan niet.

Voorbeeld met dokter, patiënt en consultatie is niet hiërarchisch want je hebt twee wortels dus we moeten aanpassingen maken: is een beperking van hiërarchisch model

### Netwerkmodel

Recordtype

Settype

Recordkeys

meerdere relaties tussen recordtypes

## Vergelijking relationeel model

Hiërarchische structuur kunnen we opslaan in moderne, relationele databanken.

Netwerk structuur kunnen we ook opslaan in moderne, relationele databanken

Hierarchisch < netwerk < Relationeel 🡪 deelverzameling: eigenlijk hebben we het oorspronkelijke idee altijd uitgebreid.

## Subqueries: niet gecorreleerd

output van select statement: tabel

Subquery: tabelexpressie binnen een tabelexpressie. Wanneer gebruiken? “ik kan het antwoord niet weten want hier bestaat geen tabel voor…” Dan zelf tabel maken = subquery

* Scalaire subquery: één waarde (1r, 1k)
* rij-subquery: 1rij
* kolom: 1 kolom
* tabel: meerdere rijen, meerdere kolommen

## Gecorreleerde subqueries

De subquery ziet alles wat de hoofdquery ziet.

# 29/02/2024

Als men spreekt over ‘relatie’ dan bedoelt men ‘inhoud van de tabel’. Selectie is de ‘where‘ conditie.

ACID:

* Atomair: het gaat door of het gaat niet door, de ultieme basis
* Consistent: ten alle tijden garandeerd uw databank dat de data voldoet aan de regels voor de databank
* Isolatie: heeft te maken met meerdere gebruikers, je kan op verschillende isolation levels werken
* Duurzaamheid

OLTP: Online transaction processes

OLAP: Online analytical processing

# Bewust productkeuzes maken

Keuzes maken op basis van:

* efficiëntie,
* GDPR,
* prijs/geld/winst,
* performantie
* Hoeveel gebruikers zijn er?
* Duuzaamheid
* Ondersteuning
* Systeemvereisten, besturingssysteem… Draait onze server wel op een windows systeem?
* Licentie: staat er in de licentie of we het wel of niet mogen verkopen

Gratis gesloten software is echt gevaarlijk! Het wordt gratis aangeboden, vraag je dan af waarom dat het gratis is? Meestal ben jij als gebruiker dan het product.

Open en vrij is niet hetzelfde: open heeft betrekking op de code van het programma, en vrij heeft te maken met het gebruik van het programma

Wat zijn de fouten die gebeuren bij het kiezen van de databank:

* Niet genoeg nagedacht op voorhand over wat er belangrijk is en bijvoorbeeld alleen maar kiezen op basis van één element
* Geen rekening houden met GDPR
* Te veel databanken: applicatie X op db1, app y op db2… terwijl die heel gelijkaardig zijn en beter op dezelfde databank kunnen staan
* Datanormalisatie: soms gaan mensen hier te ver in. Moelijke balans met gezond verstand. Denk na over “waarom houd je dit bij? Waarom zou je het weglaten?”
* Replicatie is niet hetzelfde als backup. Sommigen vergeten backups te maken
* Met de hamer de loodgieterij doen: houd je niet vast aan wat je altijd gebruikt, of hoe je het normaal doet, maar zoek bij elke situatie wat je nodig hebt

Softwareapplicaties gaan maar 5-10 jaar mee. Dus vraag je af hoe lang de data nodig is, hoelang die moet meegaan en op basis daarvan kiezen welke software je gebruikt, of je eigen databank ontwikkelen.

## Joins

Bij full outer join: je kan ‘using’ gebruiken als in beide tabellen de kolommen die gelijk moeten zijn exact dezelfde naam hebben. In dat geval krijg je maar één kolom met deze data (bijv 1 kolom met alle spelersnummers.) Wanneer je ‘on’ gebruikt moet je twee kolommen opgeven en krijg je ook twee kolommen terug (twee kolommen met in elks de spelersnummers)

Natural join: op basis van de titel van de kolommen gaat er een join gemaakt worden (moet je dus wel een beetje mee oppassen)

## Lateral joins

Om snel data te genereren: select generate\_series(n,m)

## Set operatoren

Monsterquery: voor elk team het aantal gespeelde sets

Intersect is basically een inner join

Except is een left outer join met een on-conditie OF een “not in” OF idk

## Views

Bekijk de powerpoint i guess

‘with check option’

Je moet je afvragen: “wat heeft zin” als in welke beperkingen bij je view, niet in het leven

Vkflqdjlfjiqsdjfdjfslkdqklfljmuh’qzeflkljkqsdfhjjfie

Mmaterialized view: extra data komt in je oorspronkelijk geselcteerde tabel ofzoiets idk ik ben niet meer mee 0 focus

Materialized view is performanter ennnnnn de data erin verandert niet is bijvoorbeeld handigbij het maken van een voetbalteam dat een heel seizoen niet verandert