Samenvatting Databanken 2017-2018

# NoSql

Relationele databanken zijn niet goed genoeg voor enorme toepassingen (bv facebook, google, amazon, …) door drie oorzaken:

1. Enorme hoeveelheden data (petabytes)
2. Gedistribueerd over duizenden servers
3. Niet of weinig gestructureerde gegevens

NoSql betekent “niet relationeel” en bestaat in meerdere vormen:

* Key/value (bv. Redis, Voldemort, Riak)
* Kolom-geörienteerde databanken (bv. BigTable, Cassandra, HBase, Hypertable)
* Document-geörienteerde databanken (bv. MongoDb, CouchDb)
* Graaf-geörienteerde databanken (bv .Neo4J, FlockDB)

# Gedimensioneerde gegevensmodellering

Bij ontwerp van databank zijn er twee keuzes:

1. De transactiedatabank (gewone databank), maakt gebruik van het relationele model
2. Datawarehouse (bijhouden van historiek), maakt gebruik van gedimensioneerde gegevensmodellering

De transactiedatabank wordt ontworpen met zo weinig mogelijk redundantie. Bij datawarehouses is zo een model te complex omdat zelfs bij eenvoudige queries, vaak vele tabellen aan elkaar gelinkt moeten worden met betrekking tot de historiek.

gedimensioneerde gegevensmodellering stelt tabellen van een databank voor in een sterschema.   
Dit schema bevat:

1. 1 feitentabel (kan petabytes aan informatie bevatten). Kenmerken van de feitentabel is
   * Meervoudige sleutels die elk verwijzen naar een dimensietabel
   * Meerdere attributen (de meetwaarden, meestal numeriek want ze moeten geaggregeerd kunnen worden)
2. Meerdere dimensietabellen. Kenmerken van dimensietabellen zijn:
   * Een enkelvoudige sleutel. (meestal artificieel gegenereerd zonder logische betekenis)
   * Kolommen die relevant zijn bij de bevraging van de tabel.
   * De tabel moet niet aan de derde normaalvorm voldoen. De reden is omdat er dan minder tabellen moeten aangesproken worden. De query verloopt dus sneller, maar er zijn dus wel meer kolommen en redundante gegevens.

De feitentabel kan enkel geraadpleegd worden via een dimensietabel. Een query verloopt op volgende manier:

1. Men zoekt eerst in de diverse dimensietabellen op welke de sleutels zijn van de rijden die aan de selectiecriteria voldoen
2. Vervolgens construeert men de sleutels in de feitentabel om de feiten effectief op te halen.

Het sequentieel aflopen van de feitentabel is geen haalbaar alternatief.

# Implementatie van Joins

Er zijn 4 implementaties van joins:

1. Block nested loop join
2. Index nested loop join
3. Sort-merge join
4. Hash join

De keuze ligt niet aan de programmeur, maar aan de database zelf. De database beslist aan de hand van bestaande indices welke techniek er gebruikt zal worden

Symbolen die gebruikt worden:

* : De uiteindelijke verwerkingstijd
* : Het aantal diskblokken voor tabel X
* : Het aantal diskblokken voor tabel Y
* : Het aantal rijen van tabel X
* : Het aantal rijen van tabel Y

## Block nested loop join

* Het eenvoudigste algoritme maar ook het slechtste
* Wordt uitgevoerd indien er geen indices zijn
* In vier geneste lussen worden elke rij van X vergeleken met elke rij van Y
  + De twee buitenste lussen lezen blokken van elke tabel in het geheugen
  + De twee inwendige lussen vergelijken een rij van de eerste blok met een rij van het tweede blok
* Het aantal blokken wordt te groot wat een impact heeft op performantie door het blokkeren van I/O
* Uitvoeringstijd: . komt overeen met de linkeroperand van de join operatie, en bevat best dus het minst rijen om de uitvoeringstijd te doen dalen.

## Index nested loop join

* Wordt uitgevoerd indien alle optredende attributen van één van de tabellen geïndexeerd zijn.
* Drie geneste lussen.
  + De meeste uitwendige lus leest blokken in van de niet-geïndexeerde tabel X.
  + De middenste lus loopt doorheen elke rij van dergelijke blokken
  + De binnenste lus zoekt via de indices naar de corresponderen rijen van Y
* Uitvoeringstijd: . fy is een functie van de gebruikte indexmethode en van de aantallen rijen die aan het join-predicaat voldoen. De index nested loop is b ij een equijoin( een join waarbij het *on* predicaat een gelijkheid heeft: bv. on x.id = y.id)

## Sort Merge Join

* Wordt uitgevoerd indien beide tabellen gesorteerd zijn volgens de attributen van het equi-join-predicaat
* Geen lussen
* De tabellen worden gemerged.
* Er wordt van elke tabel 1 blok in het geheugen genomen en wordt er telkens één rij beschouwd
  + Afhankelijk van de relatieve volgorde van deze twee rijen, wordt er een rij verder geschoven in één van de blokken.
* Uitvoeringstijd: indien beide tabellen al op voorhand gesorteerd zijn.

## Hash Join

Idk

# DDL Aspecten

= Data Definition Language

## Tabel-en kolomdefinities

## Constraints

Constraints zijn beperkingen die opgelegd worden aan kolommen

Belangrijke constraints zijn:

* Not null
  + De waarde is verplicht
  + Enkel inline definitie van deze constraint mogelijk
* Primaire sleutel
  + De waarde is uniek voor elke rij
  + Wordt geïmplementeerd door “primaire index”
  + Zonder primary key kunnen er duplicate rijen optreden
* Verwijzende sleutel
  + Een sleutel die verwijst naar een item in andere tabel
  + Als een item waarnaar verwezen wordt verwijdert wordt, kunnen er 3 acties optreden:
    - No Action: Het verwijderen van de ouderrij wordt onmogelijk indien er nog verwijzingen naar zijn. Het kindrij moet dus eerst aangepast worden
    - Cascade: Bij een wijziging van een ouderrij worden alle kindrijen trapsgewijs aangepast. Indien er andere regels zijn die deze aanpassingen tegenhouden, wordt de actie in het geheel afgebroken
    - Set null / set default: Als een ouderrij aangepast of verwijdert wordt, kan de waarde van de verwijzende sleutel op de null/default waarde gezet worden. Indien er andere regels zijn die deze aanpassingen tegenhouden, wordt de actie in het geheel afgebroken
* Unique
  + Om meerdere kolomcombinaties uniek te maken. Wordt vaak gebruikt bij kandidaatsleutels die niet primaire sleutel zijn. (bv primaire sleutel is “id”, de combinatie van “id” en “naam” wordt unique)
* Check
  + Wordt gebruikt om bv. intervallen op te leggen (enkel waarden tussen 10 en 20 toegelaten) , of enkel woorden met de letters ‘a’, ‘b’ of ‘c’.

## Views

Een view is een aparte tabel dat uitsluitend bedoelt is om te lezen en te gebruiken in queries (analoog aan Common Table Expressions).

* Materialisation: De view wordt op voorhand uitgevoerd en de resulterende tabel wordt opgeslagen in het geheugen of de schijf
* Resolution: De view wordt selectief met de hoofdquery verwerkt.

Een view kan in meerdere queries gebruikt worden. Views kan in de meeste gevallen enkel door de database administrator aangemaakt worden. CTE heeft het voordeel dat iedereen ze kan maken.

De inhoud van een view kan gewijzigd worden, voor zover de view aan dezelfde restricties voldoet als wijzigbare CTE’s. De with check option zal de view controleren indien deze view gewijzigd wordt. Indien er rijen worden toegevoegd of kolommen worden aangepast, zal er gekeken worden of de view nog altijd aan de constraints voldoet. Indien dit niet het geval is wordt de operatie in het geheel afgebroken.

## Indexering

### Bestandsorganisatie

1. Heap Bestanden
   * Rijen worden toegevoegd in volgorde van creatie
   * Beste oplossing voor kleine tabellen waarop niet selectief geslecteerd wordt
2. Sequentiële bestanden
   * Rijen worden gesorteerd volgens sorteerdeorde (niet in blok zelf, maar bloken ten opzichte van elkaar zijn gesorteerd)
   * Niet goed voor updates/inserts
   * Binaire zoekmethode
3. Hash bestanden
   * I