Samenvatting Databases

Robin Vanhove

Juni 2017

Inhoudsopgave

Beknopte samenvatting voor het OPO Gegevensbanken. Voornaamlijk gebaseerd op de slides, maar ook het boek.

Versie 0.0

Gecompileerd op 23 juni 2017

Deel I

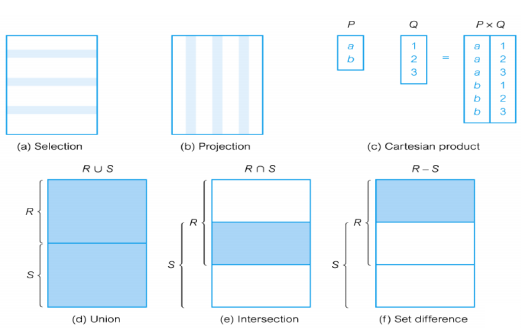
Conceptueel en Relationeel Model & Query's

1 ER & EER

Niet te kennen voor examen

2 Relationele algebra

2.1 Operatoren



Naam	Teken
Selectie	σ
Projectie	π
Hernoeming	$\rho, \leftarrow$
Unie	$\cup$
Doorsnede	$\cap$
Verschil	$-$
Caresisch product	$\bowtie$
Join Operator	$*$
Deling	$\div$

Fundamentele operatoren $\{\sigma, \pi, \cup, -, \times\}$, zijn de enige nodige operatoren. De andere kunnen er op gebaseerd worden.

2.1.1 Selectie

$$\sigma_{\text{selectiecriterium}}(R)$$

Selecteerd een aantal tupels ui een rij R meht het criterium. Het resultaat is een nieuwe relatie (tabel) met hetzelfde schema.

bv.

- $\sigma_{ID=1}(USERS)$
- $\sigma_{color='red' \vee color='green'}(BOATS)$
- $\sigma_{AGE<50}(USERS)$

Selectie is cummulatief, dus $\sigma_a(\sigma_b(T)) = \sigma_{a \wedge b}(T)$

2.1.2 Projectie

$$\pi_{\text{attributenlijst}}(R)$$

Een aantal kolommen uit een tabel halen.

bv.

- $\pi_{fist\_name,last\_name}(USERS)$
- $\pi_{color}(\sigma_{ID=1}(BOATS))$

2.1.3 Hernoeming

$$RESULT \leftarrow \sigma_{Dno=1}(EMPLOYEE)$$
$$\rho_{RESULT}(\sigma_{Dno=1}(EMPLOYEE))$$

2.1.4 Unie doorsnede en verschil

Unie	$\cup$
Doorsnede	$\cap$
Verschil	$-$

Enkel op vergelijkbare relaties.

2.1.5 Cartesisch product

$$Q = R \times S$$

Geeft als resultaat een nieuwe relatie die elke mogelijke combinatie van de twee tupels bevat.

2.1.6 Join Operator

$$R \bowtie_F S$$

Is hetzelfde als een cartesisch product gevolgd door een selectie.

Er zijn meerdere soorten joins

- **Theta join** een join waarbij de voorwaarde in de vorm is van $A\theta B$
  - Met $\theta = \{=, <, >, \leq, \geq, \neq\}$
- **Equi-join**, $R \bowtie_{a=b} S$
- **Natuurlijke join**, $R * S$. Een join waarbij de sleutel gebruikt wodt ip een conditie.

Een uitwendige join (links, rechts of volledig) is een speciale join die sowieso ieder element uit de (linker, rechter of beide) relatie, met en null als er geen paar gevonden is. Het teken hiervoor is een strikje met twee lijntjes in de richting van de join.

2.1.7 Deling

$$T = R \div S$$

Tegengestelde van het cartesisch product.

Bijvoorbeeld, voor welke zeilers bestaan reserveringen voor alle boten in een verzameling

2.2 Aggregaatfuncties

$$_{groepering}\Im_{functies}(R)$$

Functies die op een verzameling waarden uitgevoerd worden. SUM, AVERAGE, MAX, MIN, COUNT.

- Groepering is de verzameling van attributen waarop de groepering gebeurt
- Functies is de lijst van koppels (functie, attributt)

bv. $_{Dno}\Im_{AVERAGESalary}(EMPLOYEE)$

3 SQL

Structured Query Language

3.1 Operatoren

Formularium van SQL beschikbaar op examen.

3.2 Aggregaatfuncties

AVG, SUM, MIN, MAX, COUNT

Eerst groeperen met GROUG BY

3.2.1 Aanpassingen

INSERT, UPDATE, DELETE

3.3 Views

Een **view** is een afgeleide relatie, dit wil zeggen dat de tupels niet expliciet worden opgeslagen. Implementatie op twee manieren

- **Query modification**, de query wordt aangepast voor hij wordt uitgevoerd op de onderliggende tabellen.
- **View materialization**, de afgeleide tabel (view) wordt aangemaakt en daarna wordt de query er op uitgevoerd.

Aanpssingen zijn mogelijk als de view maar uit 1 tabel (basisrelatie) bestaat en een pk bevat.

3.4 Geneste Queries

Query in en Query.

IN, ALL, ANY, EXISTS

```
SELECT xxxx
FROM xxxx
WHERE xxxx IN (
    SELECT yyyy
    FROM yyyy
    WHERE yyyy
);
```

3.5 Transactie

Een atomaire eenheid. Standaard is iedere query een transactie. Een transactie wordt als permanente aanpssing gezien.

START ... COMMIT

ROLLBACK kan gebruikt worden om een huidige transactie ongedaan te maken.

```
START;
UPDATE xxx SET xxxx WHERE xxxx;
UPDATE xxx SET yyyy WHERE yyyy;
COMMIT;
```

3.6 Permissies

Verschillende database gebruikers kunnen ander rechten hebben.

```
GRANT right ON table TO user;
REVOKE right ON table TO user;
```

3.7 Restricties

Een atribuut kan een primaire sleutel zijn (id). Gewoon uniek. Of een sleutel die naar een andere tabel wijst.

```
PRIMARY KEY <attr>
UNIQUE <attr>

FOREIGN KEY <attr> REFERENCES <table><attr>
```

Een atribuut kan een standaard waarde hebben. Zou niet 'null' mogen zijn. Of kan aan andere voorwaarde onderheven worden.

```
NOT NULL <attr>
DEFAULT <value>
CHECK <condition>
```

Algemene beperking opleggen met ASSERTION

```
CREATE ASSERTION <name> CHECK <cond>
```

3.8 Triggers

Event - voorwaarde - actie

```
CREATE TIGGER <name>
{BEFORE | AFTER} <event> ON <table>
FOR EACH ROW
WHEN <cond>
    <action>
```

4 Relationele Calculus

Relationele Algebra: **Hoe** Relationele Calculus: **WAT**

- Tupelcalculus
- Domeincalculus

Gebruik van predikatenlogica.

4.1 Queries

Een query is in de vorm van {t | formule(t)} of {t.A, t.B, ... T.z | formule(t)}

```
{ t.Bdate, t.Address | EMPLOYEE(t) and t.Fname = 'John' and t.Lname = 'Smith' }
{t | BOATS(t) and (t.color='red' or t.color='green')}
```

De kwantoren $\exists$ en $\forall$ zijn ook mogelijk.

5 Programma's verbinden met een Database

Niet te kennen voor examen

6 Ontwerp van een database

1. Hoogniveau moddellering (top-down)
   - (E)ER schema
2. Meteen een relationeel gegevnesschema (bottom-up)

Informatie bewaren, minimale redundantie ($\rightarrow$ maximale peformantie)

$\rightarrow$ **normaliseren**

6.1 Informele richtlijnen

1. De betekenis van een relatie mot gemakkelijk verklaard kunnen worden.
   - Betekenis van een relatie moet duidleiljk zijn.
   - Betekenis van een attribuut moet duidleiljk zijn.
2. Redunantie en anomalieën vermijden.
3. Vermijd de waarde 'null'.
   - Niet van toepassing, ongekend.
4. Vermijd dat na equi-joins onechte tupels kunnen ontstaan. Vermijd attributen met dezelfde naam in verschillende relaties (als ze geen foreing keys zijn).

6.2 Functionele Afhankelijkheden

Een **functionele afhankelijkheid** ($X \rightarrow Y$) tussen twee (verzamelingen van) attributen X en Y is een beperking op de mogelijke tupels die gevormd kunnen worden. Voor twee tupels t_1 en t_2 is het namelijk zo dat als $t_1[x] = t_2[x]$ geldt dan ook $t_1[y] = t_2[y]$ geldt.

In andere woorden de waarden van de Y component van de tupel wordt bepaalt door de waarde van X. Of de waarde van de X component bepalen uniek (functioneel) de waarde van de Y component.

bv.

- Ssn $\rightarrow$ Ename
- Pnumber $\rightarrow$ {Pname, Plocation}
- {Ssn, Pnumber} $\rightarrow$ Hours

6.2.1 Afleidingsregels

1. Reflexiviteitsregel $Y \subseteq X \Rightarrow X \rightarrow Y$
2. Uitbreidingsregel $\{X \rightarrow Y\} \models XZ \rightarrow YZ$
3. Transitiviteitsregel $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$
4. Decompositieregel $\{X \rightarrow YZ\} \models X \rightarrow Y$
5. Verenigingsregel $\{X \rightarrow Y, Y \rightarrow Z\} \models \{X \rightarrow YZ\}$
6. Pseudo-transitviteitsregel $\{X \rightarrow Y, WY \rightarrow Z\} \models \{WX \rightarrow Z\}$

F^+ is de verzameling die alle afleidingen die uit F volgen bevat.

bv. $F = \{SSN \rightarrow ENAME, PNUMBER \rightarrow \{PNAME, PLOCATION\}, \{SSN, PNUMBER\} \rightarrow HOURS\}$

- { SSN }$^+$ = { SSN, ENAME }
- { PNUMBER }$^+$ = { PNUMBER, PNAME, PLOCATION }
- { SSN, PNUMBER }$^+$ = { SSN, PNUMBER, ENAME, PNAME, PLOCATION, HOURS}

6.3 Normalisatie

Een **normaalvorm** legt bepaalde eisen op aan een relatie. Normalisatie is een relatie in een bepaalde normaalvorm brengen.

Elke volgende normaalvorm is een speciaal geval van de vorige.

Een **super sleutel** of super key is een aantal attributen van een tupel die uniek zijn voor de tupel.

Een **sleutel** is een supersleutel waarvan geen attribuut verwijderd kan worden. Dus een sleutel is minimaal. Alle mogelijke sleutels zijn **kanidaat sleutels** maar er is maar 1 **primaire sleutel**.

6.3.1 Eerste Normaalvorm

Een relatieschema is in de **eerste normaalvorm** als het domein van elk atribuut is enkelvoudig (atomair).

Zo kan een attribuut geen lijst zijn. Maar moet er voor ieder element van de lijst een nieuwe tupel zijn.

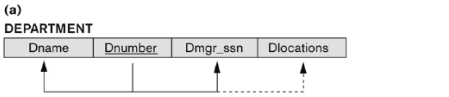


Figure 10.8
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

(b)
DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)
DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

6.3.2 Tweede Normaalvorm

We spreken over een **partiele functionele afhankelijkheid** of een **partial dependency** als voor een afhankelijkheid $X \to Y$ geldt dat we een X door Z kunnen vervangen waarbij $Z \subset X$ en de afhankelijkheid $Z \to Y$ nog altijd geldig is. Dus als we een attribuut kunnen weglaten is het partieel functioneel afhankelijk.

Anders spreken we over een **volledige functionele afhankelijkheid**.

Een relatieschema is in de **tweede normaalvorm** asa ieder niet-sleutel attribuut volledig functioneel afhankelijk is.

In andere woorden voor elk niet-sleutel-attribuut moet de hele primaire sleutel nodig zijn om het te determineren.

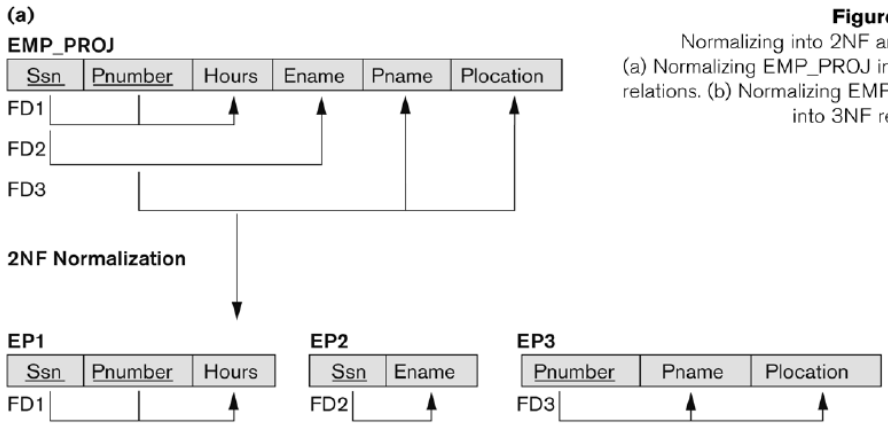


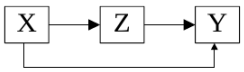
Figure 10.10
Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

6.3.3 Derde Normaalvorm

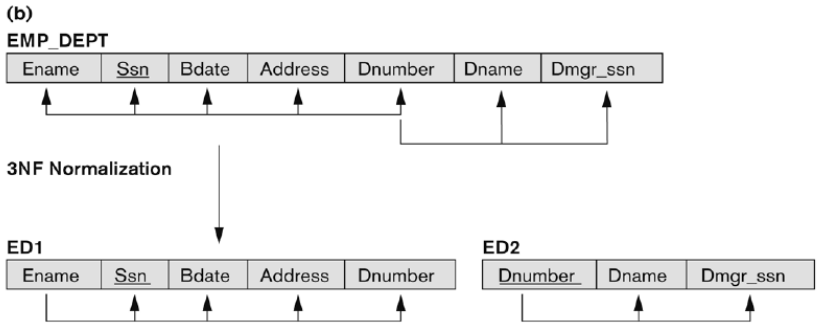
Y is **trivaal functioneel afhankelijk** van X asa $Y \subseteq X$.

Een functionele afhenkelijkheid $X \to Y$ is een **transitieve functionele afhankelijkheid** asa er geen Z bestaat waarbij

1. Z is volledig en niet-triviaal functioneel afhankelijk van X.
2. Z is geen (echte of onechte) deelverzameling van een kandidaatsleutel.
3. Y is niet-triviaal functioneel afhankeklijk van Z.



Een 2NF-reltieschema is in de **derde normaalvorm** asa voor geen enkel niet-sleutelattribuut A geldt dat A *transitief* functioneel afhankelijk is van een kandidaatsleutel. Dus alles is *direct* afhankelijk van een sleutel.



6.3.4 Boyce-Codd Normaalvorm

De **boyce-Codd Normaalvorm** (BCNF) geldt als er voor iedere niet-triviale functionele afhankelijkheid $X \to Y$ geldt dat X een supersleutel is.

De BCNF haalt alle ongewenste functionele afhankelijkheden weg. Maar dit is niet steeds bereikbaar zonder andere problemen te creëren.

6.3.5 Vierde Normaalvorm

Een **meerwaardige afhankelijkheid** genoteerd met $X \twoheadrightarrow Y$ als er vier tupels zijn zodat

- $t_1[X] = t_2[X] = t_3[X] = t_4[X]$
- $t_3[Y] = t_1[Y]$ en $t_4[Y] = t_2[Y]$
- $t_3[Z] = t_2[Z]$ en $t_4[Z] = t_1[Z]$

Tupel	X	Y	X
t_1	a	b_1	c_1
t_2	a	b_1	c_2
t_3	a	b_2	c_1
t_4	a	b_2	c_2

bv. T-shirts hebben een *model*, *maat* en *kleur*. Model $\twoheadrightarrow$ maat betkent dat:

- Voor elk *model* wordt in welbepaalde *maten* geleverd, onafhankelijk van de *kleur*.
- Dus elke combinatie van kleu en maat is mogelijk.

Een **meerwaardige afhankelijkheid** $X \twoheadrightarrow Y$ is **triviaal** asa $Y \subseteq X$.

Een relatieschema is in de **vierde normaalvorm** asa voer iedere niet-triviale meerwaardige afhankelijkehid van de vorm $X \twoheadrightarrow Y$ van F^+ geldt dat X een supersleutel is.

6.3.6 Vijfde Normaalvorm

Een **join-afhankelijkheid** JD($U_1, ...U_n$) in een relatie is en restrictie die aangeeft dat voor elke r van R geldt dat er een verliesloze decompositie in relaties $R_1, ..., R_n (n > 2)$ is met attributen $U_1, ...U_n$.

Een relatieschema is in de **vijfde normaalvorm** asa voor elke niet-triviale join-afhankelijkeheid JD($U_1, ...U_n$) van F^+ geldt dat $U_1, ...U_n$ supersleutels zijn.

Deel II

Het Fysiek Model

7 Indexeren

Je leest of schrijft nooit 1 record van de opslaghardware, altijd in blokken.

Een blok die in het geheugen gelden wordt kan snel doorzocht worden.

7.1 Zoeken in bestanden

7.1.1 Zoeken in ongeordende bestanden

In **ongeordende bestanden** moet er **lineair** gezocht worden. Gemiddeld wordt de helft van de records bekeken.

7.1.2 Zoeken in geordende bestanden

Als de bestanden **geordend** zijn volgens het attribuut waarop we zoeken kunnen we **binair zoeken**.Dit kan gemiddeld in $log_2(N)$ tijd.

7.2 Indexen: Definitie en soorten

Een **index** is een lijst van onderwerpen met wijzers naar het bestand. Of in andere woorden, een index is een datastructuur die toegang tot een bestand via een veld efficient maakt.

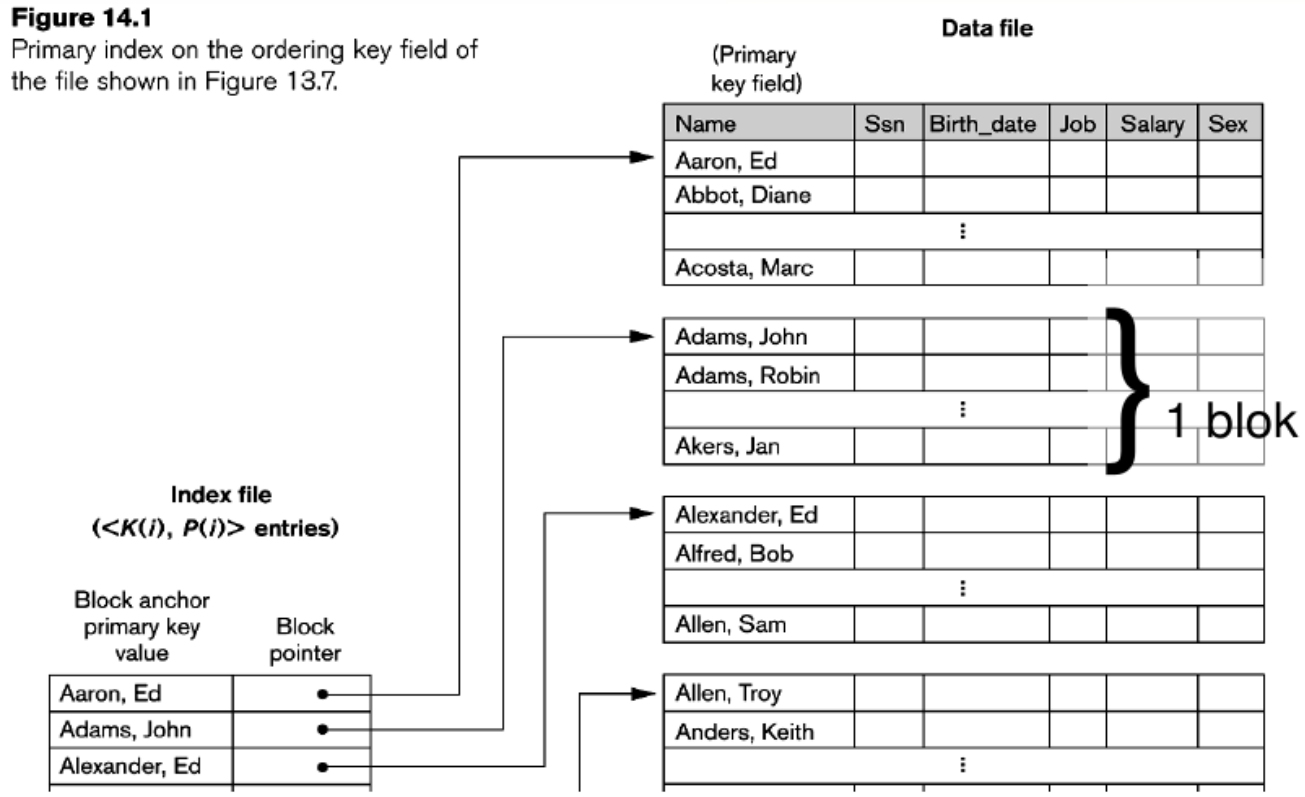
Een index is een bestand zoals de data een bestand is.

Er zijn drie soorten indexen, de **primaire index** is de index op het veld dat de ordening van de bestanden bpaald en die ieder bestand uniek identificeerd (pk). De **clusterindex** is een index op het veld dat de ordening bepaalt maar niet noodzakelijk uniek is. En een **secundaire index** is een index op een ander verld dan dat wat der ordening bepaalt (ook niet uniek).

Een index is gemakkelijk omdat het veel kleiner is om in het geheugen te laden, het moet namelijk enkel enkele records en pointers bijhouden.

7.2.1 Primaire indexen

De primaire index bevat een fysish geordende lijst volgens de sleutel. De index bevat 1 record per blok. Dit is het *ankerrecord* en is meestal het eerste of het laatste record van het blok.



7.2.2 Cluster indexen

De index is fysish geordend volgens het veld dat niet uniek is. De bestanden zijn wel geordend volgens dit veld.

De index bevat per waarde van het veld 1 wijzer naar het blok waar de eerste record met die waarde in voorkomt.

7.2.3 Secundaire indexen

Een secundaire index is een index op en veld dat niet de ordening bepaalt. De index zelf wordt wel geordend volgens dat veld.

Als dit veld een secundair sleutel veld is kan er per waarde van de sleutel (en dus per record) een wijzer in de index staan.

We hebben dichte index die alle records bevat. Dit is nog steeds nutig omdat het kleiner is dan de data zelf. (minder kollomen)

7.2.4 Hash indexen

Een hashing functie wordt gebruikt om de key meteen om te zetten naar een adres.

Collisions, botsingen mogelijk.

Linear probing: open adressering of Sepereate chaining: Gesloten adressering.

7.3 Indexen in MySQL

Clustered. reorders the way records in the table are physically sotred.

Nonclustered index. Order in index is different than physically.

8 Querryverwerking en Optimalisatie

SQL Query wordt omgezet naar relationele algebra. Er wordt een boom opgesteld voor de query deze wordt geoptimaliseerd.

Eerst wordt de queryboom opgebouwd in canonieke vorm (dus letterlijk hoe de query is). Vervolgens wordt de boom geherstructureer zonder equivalentie te verliezen.

8.1 Transformatie regels

- σ cascade
  - Selectie op conjuncties van condities omzetten in opeenvolgende eenvoudige selecties.
  - $\sigma_{c_1 \text{ AND } c_2 \text{ AND ... AND } C_n}(R) = \sigma_{c_1}(...(\sigma_{c_n}(R)))$
- σ is commutatief
  - $\sigma_a(\sigma_b(R)) = \sigma_b(\sigma_a(R))$
- π cascade
  - Enkel laatste projectie overhouden
  - $\pi_a(\pi_b(R)) = \pi_a(R)$
- Omisselen van σ en π
  - Enkel als de voorwaarde c toepasbaar is op de atrributen
  - $\pi_A(\sigma_c(R)) = \sigma_c(\pi_A(R))$
- Commutativiteit van $\bowtie$ en $\times$
  - $R \times S \equiv S \times R$
  - $R \bowtie S \equiv S \bowtie R$
- Omwisselen van σ en $\bowtie$ of $\times$
  - Als de voorwaarde c toepasbaar is op de atrributen van R
  - $\sigma_c(R \bowtie S) \equiv \sigma_c(R) \bowtie S$
  - Als de voorwaarde c toepasbaar is op de atrributen van R en S
  - $\sigma_c(R \bowtie S) \equiv \sigma_{c_1}(R) \bowtie \sigma_{c_2}(S)$
- Omwisselen van π en $\times$
  - $\pi_L(R \times S) \equiv \pi_{L(R)}(R) \times \pi_{L(S)}(S)$
- Omwisselen van π en $\bowtie$
  - Als de join conditie allen attributen in L gebruikt
  - $\pi_L(R \bowtie_c S) \equiv \pi_{L(R)}(R) \bowtie_c \pi_{L(S)}(S)$
  - Anders R en S projecteren op join attributen + attributen projectie-lijst daarna joinen en op het einde projecteren op L.
- $\cup$ en $\cap$ zijn commutatief
- $\bowtie, \times, \cup$ en $\cap$ zijn associatief
- Commutativitiet van σ met verzamelings-operaties
  - $\sigma_c(R \cap S) \equiv \sigma_c(R) \cap \sigma_c(S)$
  - $\sigma_c(R \cup S) \equiv \sigma_c(R) \cup \sigma_c(S)$
  - $\sigma_c(R/S) \equiv \sigma_c(R)/\sigma_c(S)$
- Commutativiteti van π met verzamelings operaties.
  - $\pi_L(R \cup S) \equiv \pi_L(R) \cup \pi_L(S)$
- Samenvatten van $\sigma(\times)$ in $\bowtie$
  - $\sigma_c(R \times S) \equiv R \bowtie_c S$
- ...

8.2 Heuristische optimalisatie

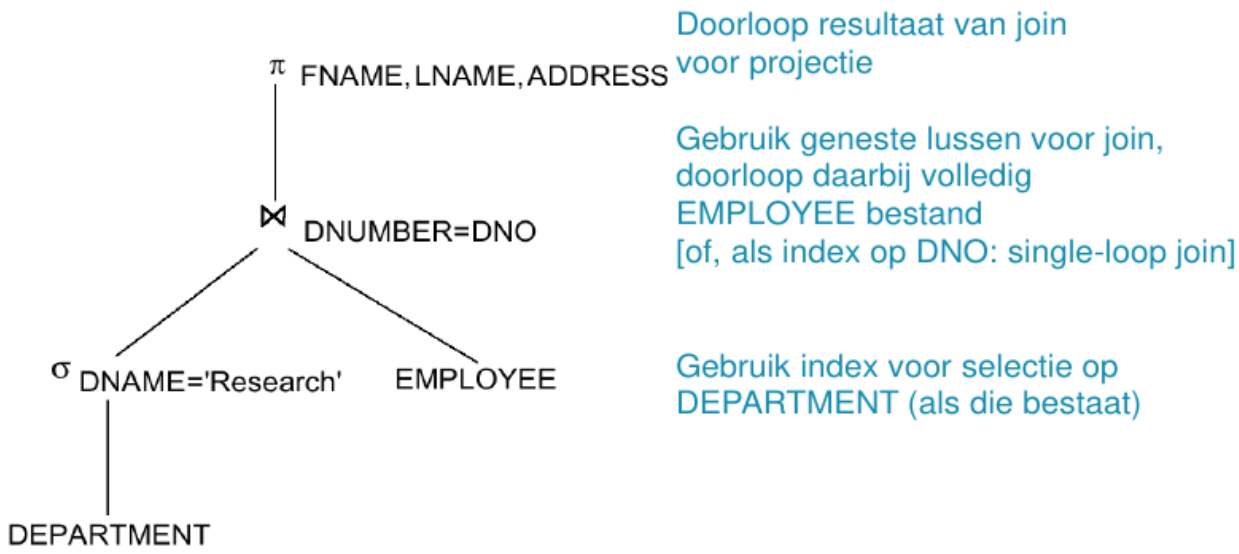
1. Splits conjunctie van selecties
2. Schuif selecties zo ver mogelijk naar beneden
   - selectie over 1 relatie: net boven de relatie
   - selectie over 2 relaties: zo dicht mogelijk boven hun cartesisch product
3. Schuif kleine relaties zo ver mogelijk naar links
   - maar houd join condities bij cartesische producten
4. Combineer cartesisch product gevold door selectie met join conditie tot join
5. Splits projecties op, en projecteer zo vroeg mogelijk
   - houd alleen attributen die verder boven nodig zijn
6. Identificeer deelbomen die door één algoritme kunnen uitgevoerd worden (zonder tijdelijke bestanden)

Kleine relaties zijn letterlijk hoeveel tupels er in de relatie zijn. Dus extra informatie, de gorete van de tupels en het aantal waardes bijhouden.

8.3 Uitvoeringsplan

Nadat de boom is geoptimaliseerd weten we de volgorde van de operaties en hebben we een indicatie van deelbomen waar mogleijk een algoritme voor bestaat.

Maar nog geen exacte implementatie dus welke indexen, en welk soort evaluatie.



8.4 Extern Sorteren

Vaak niet genoeg geheugen voor intern te sorteren bv. quicksort. Dus extern sorteren (merge sort). Dan kunnen aparte blokken om de beurt in het geheugen geladen worden.

8.5 Selectie implementeren

1. Linwair zoeken
   - kan altijd
   - duur
2. Binair zoeken
   - kan als, gesorterd op veld waarop gezocht wordt

- cheaper

3. Gebruik index of hash-functie
   - Kan als er een index of hash is
   - Bv. $\sigma_{ID=5}(USER)$
   - Primaire index: koste het aatal blokken om de tupel in de index te vinden
   - Hashing: meteen naar ongeveer de juiste blok (dus 1 of 2)
4. Gebruik de Primaire index om meerder records op te halen
   - Bv. $\sigma_{ID<5}(USER)$
   - Kost is afhankelijk van het aantal blokken waarin de tupels zich bevinden
5. Cluster index om meerdere records op te halen
   - Een '=' op een attributt dat geen key is
   - Kan als er een cluster index is op dat attribuut
   - Kost is afhankelijk van het aantal blokken waarin de tupels zich bevinden
6. Gebruik van een secundaire index (B^+ boom)
   - voor '=' en ongelijkheden
   - als index voor dat atribuut bestaat
   - Kost is afhankelijk van vergelijkingsoperatie, aantal tupels, uniekheid van waarde
7. Conjuctieve selectie c1 AND C2
   - Als voor een constrain een van de methodes S2-S6 bruikbaar is: slecteer volgens ci en test andere condities voor elk gevonden record.
   - bv $\sigma_{ID=5 \text{ AND } SEX=F}(EMPLOYEE)$ met index op ID
   - kost, afhankelijk van methode
8. (Slides direct naar 9)
9. Conjuctieve selectie door intersectie van recordpointers
   - Kan als secundaire indexen met recordpointers bestaan voor aantal subcondities met '='

9 Concurrentie

Als er meerdere applicaties querys teglijk uitvoeren moet meer er voor zorgen dat de database nog wel aan een aantal constraints voldoet.

Zo kan 1 plaats op een vliegtuig maar door 1 persoon geboekt worden. Als er meerde mensen tegelijk de laatste plaats boeken mag maar 1 van hen die krijgen.

Om dit te bereiken groeperen we onze queries in transacties. Deze kunnen **interleaved** uitgevoerd worden, dit zil weggen afzisselend door 1 processor. Of **simultaan** dan werken meerdere processoren in parallel. We bekijken enkel het interleaved model.

9.1 Mogelijke problemen

9.1.1 Tijdelijke aanpassing (dirty read)

Een programma maakt een aanpassing maar wordt door een fout afgebroken. De gewijzigde waarde wordt herstelt maar een ander pogramma heeft deze tijdleijke waarde al gelezen.

9.1.2 Foutieve sommering

Een aggregaatfunctie kan bv. een som berekenen. Als de waarde tegelijk wordt aangepast kan dit fouten opleveren.

9.1.3 Niet herhaalbare lezing

Een relatie wordt twee keer kort na elkaar gelezen maar geeft een ander resultaat.

bv. Check of er plaats is, ja. Maak een reservatie, plaats al bezet.

9.1.4 Oorzaaken

1. computer-crash
   - inhoud van geheugen kan verloren zijn
2. transactie- of systeemfout
   - verkeerde parameter, overflow, deling door 0, logische programmeerfout,..
3. uitzonderingscondities
   - bv. bestand kan niet gelezen worden, ...
4. opgelegd door concurrentiecontrole
   - bv. transactie afgebroken wegens deadlock
5. schijf-fout
   - bv. beschadigd spoor
6. fysieke problemen, catastrofes
   - brand, stroomonderbreking, ...

Bij falingen van de types 1 tot 4 moet de oorspronkelijke toestand hersteld kunnen worden (bij 5-6 ook, maar dit werkt met backups)

9.2 Transacties

Een **transactie** is een uitvoering van een programma dat de gegevensbank raadpleegt of wijzigt. Een transactie wordt ofwel helemaal uitgevoerd ofwel helemaal niet, dit om fouten te voorkomen.

Na een fout waarbij de transactie niet uitgevoerd kan worden moet de oorsprongkelijke toestand herstald worden.

Een **read-only** transactie leest enkel uit de database. Terwijl een **update** transactie gegevens aanpast.

Als en transactie correct is gebeurt wordt de transactie gecommit men spreekt dan over een **commit point**.

9.2.1 Operaties

Een transactie bestaat uit enkele operaties.

- BEGIN_TRANSACTION
- READ / WRITE
- END_TRANSACTION
- COMMIT_TRANSACTION

In het geval dat er een fout gebeurt zijn er extra bewerkingen.

- ROLLBACK (ABORT)
- UNDO
  - 1 bewerking wordt ongedaan gemaakt
- REDO
  - 1 bewerking wordt opneieuw uitgevoerd.

9.2.2 Systeemlog

In de systeemlog worden alle transacties (die de database wijzigen) bijgehouden. Dit is nodig om de database te kunnen herstellen. De volgende gegevens worden bijgehouden (T is een transactie ID)

- [start_transaction, T]
- [write_item, T, X, oude waarde, nieuwe waarde]
- [read_item, T, X]
- [commit, T]
- [abort, T]

9.2.3 Gewenste eigenschappen van transacties

ACID properties

- Atomicity (ondeelbaarheid)
- Consistency preservation
- Isolation
  - Effect moet zijn alsof het de enige uitgevoerd transactie is.
- Durability

9.3 Transactierooster (schedules)

- Conflict als 2 operaties
  - bij verschillende transaties horen
  - en hetzelfde gegevenselement gebruiken
  - en minsten 1 van hen een write is

Een rooster is **volledig** als alle operaties van de transacties er in voorkomen, in de juiste volgorden. En voor elk paar conflicterende operaties geldt dat de volgorde eenduidig vatligt.

Een rooster is **herstelbaar** asa elke transactie die gecommit is nooit meer ongedaan gemaakt moet worden. Herstelbaar betekend niet eenvoudig **cascading rollback** is mogelijk. Hierbij moet na het terugrollen van een transactie een andere ook teruggerolt worden (omdat deze iets las dat door de andere werd aangepast).

Cascadeloze rooster zijn roosters die garanderen dat cascading rollback niet nodig zijn. Een manier om dit te doen is er voor zogen dat een transactie past waarden leest die al gecommit zijn. Maar dit zorgt ervoor dat minder transacties gelijktijdig uitgevoerd kunnen worden.

In een **strikt rooster** wordt een waarde pas gelezen of geschreven nadat die gecommit is.

Een cascadeloos rooster impliceert een herstelbaar rooster en een strikt rooster impliceert een cascadeloos rooster.

9.3.1 Serialiseren van roosters

In een **serieel rooster** worden de operaties van iedere transactie na elkaar uitgevoerd, er worden dus geen transacties door elkaar uitgevoerd.

Een rooster is serializeerbaar asa het equivalent is met een serieel rooster met dezelfde transacties. Er zijn twee soorten equivalentie **resultaat-equivalentie** is dat twee roosters met dezelfde beginwaarden hetzelfde resultaat geven. Twee rooosters zijn **conflict-equivalent** asa de volgorde van de conflicten dezelfde is.

Twee roosters zijn **view equivalent** als

- de laatste write voor een read moet in beide roosters dezelfde zijn
- en de laatste write moet hetzelfde is

9.4 Concurrentiecontrole

Het is niet altijd mogelijk om een rooster te serializeren of dit te testen. Daarom worden er enkele regels (protocols) gebruikt om serializeerbaarheid te verzekeren.

9.4.1 Vergrendeling (locking)

Een slot beperkt toegang tot een element.

Het eenvoudigste soort slot is een **binair slot** dat meer twee toestanden heeft (wel/geen toegang). Bij een binair slot moet iedere transactie wachten tot het slot vrij is en het slot vrij geven als de transactie klaar is.

Een **read/write lock** of **gedeeld/exclusief slot** heeft drie toestanden en kan lezen en schrijven apart beperken. Een transactie mag altijd lezen als er het de staat van het slot gedeeld is. Om de schrijven moet de transactie exclusieve toegang hebben.

Met deze regels alleen wordt geen serialiseerbaarheid gegarandeerd.

Twee fasen vergrendeling is een protocol waarbij een transactie eerst al zijn locks op vraagt voor een lock vrij te geven. Er zijn dus twee fases de expanding phase en de shrinking phase.

Een **deadlock** is een situatie waarbij twee locks op elkaar wachten waardoor niets kan gebeuren.

Een variant van twee fasen vergrendeling neemt locks alvorens iets te schrijven. Dit vermijd deadlocks maar is niet altijd mogelijk. Omdat het volgende doel van een lock kan afhangen van een andere waarde.

9.4.2 Tijdstempels en Multiversie-technieken

Ook tijdstempels (timestaps) zijn een techniek om deadlocks te voorkomen.

Een mogelijkheid is om enkel de oudste transactie te laten wachten op en lock.

Men kan ook gewoon wachten niet toelaten maar de transactie afbreken als die geen lock krijgt en later herstarten.

Een transactie zou ook alleen maar mogen wachten als die transactie waar op gewacht wordt niet aan het wachten is. Dit is **voorzichtig wachten**.

Er kan ook een **timeout** zijn die een transactie afbreekt als die te lang moet wachten.

Een andere optie is om aan **deadlock detectie** te doen. Hierbij worden deadlock niet voorkomen maar een van de processen wordt afgebroken als er een deadlock voordoet. Maar **starvation** is hierbij een probleem als steeds hetzelfde proces afgebroken wordt.

Men kan ook deadlocks voorkomen door geen gebruik te maken van locks en enkel van tijdstempels.

Bij **multiversie** concurrentiecontrole worden meerdere versies van een waarde bijgehouden, zo kan een oude versie nog gelezen worden.

9.4.3 Optimistische concurrentiecontrole

Bij optimistische concurrentiecontrole wordt de transactie op een kopie van de data uitgevoerd en later uitgevoerd als het serialiseerbaar is. Zoniet wordt de transactie gewoon herstart.

9.4.4 Granulariteit van items

Als we spreken over het lezen, schrijven of het nemen van een lock op een item kunnen we ons afvragen over wat we juist spreken.

Bij het kiezen van dit item hebben we dus een keuze we spreken over **granulaiteit**

- Database (grove granulariteit)
- bestand
- blok
- record
- veld (fijne granulariteit)

Hoe groter het item hoe minder mogelijk, hoe fijner het item hoe meer overhead.

Intention locks zijn locks die een lock op lager nivewu toelaat.

9.5 Herstel

Ieder DBMS maakt gebruik van een **cache** om de queries te versnellen.

Bij **write ahead logging** wordt eerst naar de log geschreven (op de schijf) voor de aanpassing doorgevoerd wordt.

Een **checkpoint** in een systeemlog is een punt waarop alle wijzigingen naar de schijf geschreven zijn.

Bij **onmiddelijke aanpssingen** worden de aanpassingen aangebracht voor het bereiken van een commit point. Bij het falen is dan een rollback nodig.

Uitgestelde aanpassingen gebeuren pas als een commit point bereikt is.

9.5.1 Technieken voor onmiddellijke aanpassing

Bij onmiddelijke aanpassing is het naast de log ook nog nodig om twee lijsten bij te houden. Een met transacties die gecommit zijn na het laatste checkpoint en een met transacties die nu bezig zijn.

9.5.2 Technieken voor uitgestelde aanpassing

Een transactie kan de gegevensbank niet wijzigen vóór het bereiken van haar commit point. En een transactie bereikt haar commit point niet vooraleer alle aanpassingsopdrachten geregistreerd zijn in de log (en de log geschreven is naar schijf) Dus en undo is nooit nodig.

Enkel redo's.

9.5.3 Schaduwpaginering

Bij **shaduwpaginering** wordt er voor iedere transactie een kopie gemaakt van de oorsprongkelijke blok. Na eem commit kan deze shaduwpagina verwijdert worden.