

Gegevensbanken - Samenvatting [H01O9A]

2010-2011

Wouter Schaekers

2Bach Informatica - 2Bach Fysica - 2Bach Biologie - 2Bach Ingenieurswetenschappen
3Bach Wiskunde - 3Bach Biochemie en Biotechnologie - 1-2Master Sterrenkunde

February 26, 2014

Inhoudstafel

0	Inleiding en inhoudstafel	3
1	Het Relationeel Model	5
1.1	Inleiding	5
1.2	Het fysieke model	5
1.3	Begrippen in verband met het model	5
1.4	Nog wat extra begrippen	6
2	Het Entiteit Relatie Model (ER)	6
3	Het Extended Entity Relationship Model (EER)	7
3.1	Specialisatie	7
3.2	Generalisatie	7
3.3	Formele notaties	7
4	Relationele algebra en relationele calculus	7
4.1	Relationele algebra	8
4.1.1	Selectie	8
4.1.2	Projectie	8
4.1.3	Combinaties	8
4.1.4	Cartesisch product	8
4.1.5	Join operator	9
4.1.6	Deling	9
4.1.7	Aggregaatfuncties	9
4.1.8	Uitwendige join	9
4.1.9	Varianten op unie	9
4.2	Relationele calculus	9
5	SQL - Deel 1	10
5.1	Inleiding	10
5.1.1	Schema	10
5.1.2	Tabel	10
5.1.3	Definitie van een attribuut	10
5.1.4	Restricties op tabel	10
5.1.5	Drop behaviour (schema/tabel)	10
5.2	Queries	10

5.2.1	Dubbele attribuutnamen	10
5.2.2	Aliasen	11
5.2.3	DISTINCT	11
5.2.4	Verzamelingen-bewerkingen	11
5.2.5	LIKE	11
5.3	Operatoren	11
5.4	Gegevens ordenen	11
5.5	Geneste queries	11
6	SQL - Deel 2	11
6.1	CONTAINS	11
6.2	EXISTS	12
6.3	Expliciete verzamelingen	12
6.4	Join operaties	12
6.5	Aggregaatfuncties	12
6.6	HAVING	12
6.7	Toevoegen, weglaten, wijzigen	12
6.8	VIEW	12
6.9	ASSERTIONS	13
6.10	TRIGGERS	13
7	Functionele afhankelijkheden en normalisatie - deel 1	13
7.1	Afleidingsregels	13
7.2	Overdekking, equivalentie	14
7.3	Minimale verzameling	14
7.4	Normaalvormen	14
7.4.1	Enkele definities	14
7.4.2	De 0de normaalvorm	15
7.4.3	De 1ste normaalvorm	15
7.4.4	De 2de normaalvorm	15
7.4.5	De 3de normaalvorm	15
7.4.6	Boyce-codd normaalvorm (BCNF)	15
8	Normalisatie - deel 2	16
8.1	Decompositie	16
8.2	Het Chase algoritme	16
8.3	Algoritme voor het vinden van een sleutel	17
8.4	Nadelen van de decompositiemethoden	17
8.5	Meerwaardige afhankelijkheid en de 4de normaalvorm	17
8.6	Join-afhankelijkheden en de 5de normaalvorm	17
9	Geheugen en bestandsorganisatie	18
9.1	Bestandsorganisatie: blokken en records	18
9.2	Plaatsen van logisch opeenvolgende blokken	18
9.3	Toegang tot en bewerking van bestanden	19
9.4	Soorten bestanden	19
9.4.1	Ongeordende bestanden (seriële bestanden)	19
9.4.2	Geordende bestanden (sequentiële bestanden)	19
9.4.3	Directe bestanden	19
9.5	Hashing	19
9.5.1	Botsafhandeling	20
9.6	Externe hashing	20

10 Externe hashing	20
10.1 Dynamische hashing	20
10.2 Uitbreidbare hashing	20
11 Lineaire hashing	21
12 Indexstructuren	22
12.1 Soorten indexen	22
12.1.1 Primaire indexen	22
12.1.2 Clusterindex	23
12.1.3 Secundaire index	23
12.2 Meerdere niveau's	23
12.3 Boomstructuren	23
12.3.1 B-bomen	23
12.3.2 B ⁺ -bomen	24
12.3.3 B*-bomen	24
12.4 Indexen op meerdere velden	24
13 Begrippen van transactieverwerking	24
13.1 Herstel	24
13.2 Transactieroosters	25
13.3 Serialiseren van roosters	25
14 Transactiecontrole II: Technieken voor concurrentiecontrole en herstel	26
14.1 Concurrentiecontrole - Locking	26
14.1.1 Binaire grendels	26
14.1.2 Read/write locks	26
14.1.3 Twee-fasen-vergrendeling	27
14.2 Concurrentiecontrole - Timestamps	27
14.2.1 Deadlock preventie zonder timestamps	27
14.2.2 Deadlock detectie	27
14.2.3 Concurrentiecontrole zonder grendels	28
14.3 Concurrentiecontrole - Multiversietechnieken	28
14.4 Concurrentiecontrole - Optimistische concurrentiecontrole	28
14.5 Concurrentiecontrole - Granulariteit van gegevensitems	28
14.6 Herstel - Begrippen	28
14.7 Herstel - Hersteltechnieken [uitgestelde aanpassing]	29
14.8 Herstel - Hersteltechnieken [onmiddellijke aanpassing]	29
14.9 Herstel - Schaduwpaginering	29

0 Inleiding en inhoudstafel

Deze samenvatting is gemaakt voor de studenten van 2Bach Informatica, 2Bach Fysica, 2Bach Biologie, 2Bach Ingenieurswetenschappen, 3Bach Wiskunde, 3Bach Biochemie en Biotechnologie en 1-2Master Sterrenkunde.

Deze samenvatting is gebaseerd op de slides die gebruikt zijn in de les. Daarom zal de aangehouden volgorde dezelfde zijn als de lessen, dus niet de volgorde van het boek. Redundante lessen worden weggelaten.

In de samenvatting zal regelmatig verwezen worden naar afbeeldingen die in het boek staan. Hierbij is de zesde editie van het boek 'Database Systems' gebruikt.

Deze samenvatting is hoogstwaarschijnlijk niet foutloos. Eventuele aanpassingen kunnen gemaakt worden op <https://github.com/WouterSchaekers/Gegevensbanken-Samenvatting>.

De auteur is niet bereid samenvattingen te signeren.

Het sturen van spam is verboden. Het stalken van de auteur is, na toestemming, slechts in uitzonderlijke omstandigheden toegestaan.

De auteur is niet verantwoordelijk voor enige gevolgen van het gebruik van deze bundel.

Het is verboden de afgedrukte versie van de samenvatting te verbranden of op te eten.

Geen langdurig gebruik zonder wiskundig advies.

Alle lijnstukken voorbehouden.

Niet op de openbare weg gooien.


This resume is released under the beerware license. Donations on the following bitcoin address are really appreciated. Thanks.

"Alles moet zo eenvoudig mogelijk gemaakt worden, maar niet eenvoudiger dan dat."

-

Albert Einstein



 1Dh769mkyU3FGeEtxNcSQU4bSKkFZpFiPu

1 Het Relationale Model

1.1 Inleiding

Relatie	Attribuut 1	Attribuut 2	Attribuut 3
Tupel 1	Gegeven	Gegeven	Gegeven
Tupel 2	Gegeven	Gegeven	Gegeven
Tupel 3	Gegeven	Gegeven	Gegeven

Bijvoorbeeld, een anonieme vragenlijst ingevuld door informatici waarvan de gemiddelde resultaten bijgehouden worden:

Vakken	Moeilijkheid	Wekt interesse	Relevantie
Logica	6	9	8
Wiskunde I	1	6	7
Wijsbegeerte	5	1	0
IOV	7	5	3
Gegevensbanken	5	7	9
Artificiële intelligentie	4	8	9

In dit geval zijn 'Moeilijkheid', 'Wekt interesse' en 'Relevantie' de attributen. 'Logica', 'Wiskunde I', ... zijn de tupels.

Op deze manier kunnen we uit deze tabel halen dat de gemiddelde informaticus het vak Wijsbegeerte niet interessant en absoluut niet relevant vindt.

We noemen $U = \{A_1, A_2, \dots, A_n\}$ de verzameling van alle attributen en $DOM(A_i)$ de verzameling van alle mogelijke waarden voor A_i .

De formele notatie van een tupel kan op verschillende manieren gebeuren. We nemen in dit voorbeeld de eerste relatie van onze tabel:

$$\begin{aligned} &< (Moeilijkheid, Wekt\ interesse, Relevantie), (6, 9, 8) > \\ &< (Moeilijkheid, 6), (Wekt\ interesse, 9), (Relevantie, 8) > \\ &(6, 9, 8) \end{aligned}$$

Merk op dat alle waarden atomair zijn. Een waarde is atomair als deze niet meerwaardig of samengesteld is. *NULL* waarden kunnen wel voorkomen.

Het is mogelijk dat je slechts een 'deeltupel' nodig hebt. In ons voorbeeld:

$$\begin{aligned} t &= < 6, 9, 8 > \\ t[Moeilijkheid, Relevantie] &= (6, 8) \end{aligned}$$

1.2 Het fysieke model

Om een aantal begrippen te kunnen uitleggen gebruiken we het 'COMPANY' diagram (figuur 7.2 of zie slides).

Vierkante elementen zijn entiteiten en duiden meestal op een voorwerp of persoon. Ovale elementen zijn attributen, de eigenschappen van deze entiteiten. Ruitvormige elementen vormen de relaties tussen de entiteiten.

1.3 Begrippen in verband met het model

Een regulier of sterk entiteitstype Hangt niet af van een andere entiteit. Met andere woorden, een entiteit dat op zichzelf staat.

Een zwak entiteitstype Hangt wel af van een andere entiteit. Deze entiteiten worden aangeduid met een dubbele rand.

Bijvoorbeeld 'Employee' is een sterke entiteit, omdat dit een bepaalde persoon voorstelt (met een unieke id).

Anderzijds is 'Team' een zwakke entiteit, omdat de identificatie van een team afhangt van het bedrijf waar het team onderdeel van is ('Team 1' kan in meerdere bedrijven voorkomen).

Sleutelattribuut Een attribuut dat een entiteit identificeert. Deze attributen worden onderlijnd.

Verwijssleutel Een attribuut dan een zwakke entiteit identificeert. Deze attributen worden onderlijnd met een stippelijnd.

In het 'COMPANY' voorbeeld vinden we dat 'Ssn' het sleutelattribuut is van EMPLOYEE en 'Name' de verwijssleutel is van DEPENDENT.

Sleutels moeten altijd uniek zijn.

Binair relatietype Een relatie tussen twee entiteiten. Deze kan $1 : 1$, $1 : N$ en N, M zijn en noemt ment de kardinaliteit. Deze wordt geschreven op de lijnen tussen de relaties. In dit voorbeeld kan een EMPLOYEE aan slechts één DEPARTMENT werken en werken er aan één DEPARTMENT meerdere EMPLOYEES. Vandaar de relatie $1 : N$.

Identificerende relatie Relatie die een zwakke entiteit identificeert. Deze wordt aangeduid met een dubbele rand. In dit voorbeeld zal DEPENDENT worden geïdentificeerd door EMPLOYEE.

Meervoudig attribuut Een attribuut dat meerdere waarden kan hebben. Bijvoorbeeld DEPARTMENT kan meerdere locaties hebben. Meervoudige attributen hebben een dubbele rand. Dit mag niet verward worden met **Samengestelde attributen**. Deze attributen bestaan uit meerdere attributen die samen voorkomen. Bijvoorbeeld 'Geboortedatum' dat uit een dag, maand en jaar bestaat.

Afgeleide attributen Deze attributen kunnen worden afgeleid uit andere attributen. Deze attributen zijn dus vrij redundant en kunnen eventueel weggelaten worden. Deze attributen worden omlijnd door een stip-pelijnd. Bijvoorbeeld het attribuut 'omtrek' van een entiteit 'cirkel' terwijl het attribuut 'straal' al gegeven is.

Totale participatie Wanneer elk element van een bepaalde entiteit meedoet in een bepaalde relatie, is de participatie totaal. Dit wordt weergegeven door een dubbele relatie-lijn. In dit voorbeeld heeft elk DEPARTMENT een EMPLOYEE als manager, maar is niet elke EMPLOYEE een manager van een DEPARTMENT. Een samenvatting van deze symbolen kan gevonden worden op figuur 7.14.

1.4 Nog wat extra begrippen

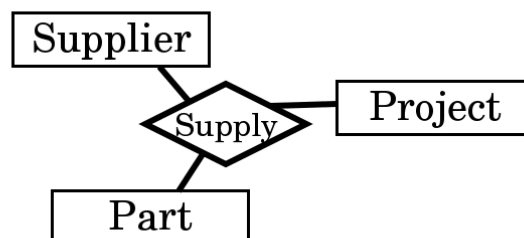
Supersleutel Een verzameling van attributen die een tuple ondubbelzinnig bepalen. Of anders gezegd: een set van attributen waarvoor geldt dat geen twee verschillende tupels dezelfde waarde hebben voor deze attributen.

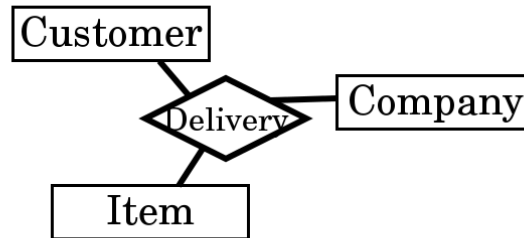
Kandidaatsleutel Een supersleutel zonder overtollige sleutel. Dus een zo klein mogelijke supersleutel.

Primaire sleutel Een sleutel gekozen uit de kandidaatsleutel. De andere sleutels uit de kandidaatsleutel zijn **Alternatieve sleutels**. In ons model komt de primaire sleutel overeen met het sleutelattribuut. Deze primaire sleutel mag uiteraard nooit *NULL* zijn.

2 Het Entiteit Relatie Model (ER)

Buiten binaire relatietypen kunnen er ook ternaire relatietypen opduiken. Dit is één enkele relatie tussen drie entiteiten.





Ternaire relaties kunnen ook worden voorgesteld door drie binaire. Zie figuur 7.17 of slides.

Een zwakke entiteit wordt geïdentificeerd via een ander entiteitstype. Het is logisch dat de deelname in deze identificerende relatie totaal moet zijn. In dit geval zijn zowel de foreign key (de sleutel via welke sterke entiteit deze entiteit wordt geïdentificeerd) en de partial key (de sleutel van deze entiteit zelf die een zekere vorm van identificatie is) een onderdeel van de primary key. Partial keys worden met een stippelijnd. onderlijnd.

3 Het Extended Entity Relationship Model (EER)

EER is een uitgebreide versie van ER. EER behandelt ook (sub/super)klassen, specialisatie/generalisatie en overerving van attributen.

3.1 Specialisatie

Specialisatie en generalisatie zijn in feite niets anders dan overerving. Bij specialisatie wordt een bepaald attribuut opgesplitst in verschillende entiteiten. Algemene attributen blijven bij de 'superentiteit'. Deze entiteit wordt dan opgesplitst in 'subentiteiten' zoals in figuur 7.19 (of slides).

3.2 Generalisatie

Bij generalisatie wordt de omgekeerde beweging gemaakt. Entiteiten worden bij elkaar gezet om tot een algemene 'superentiteit' te verkrijgen. De structuur blijft hetzelfde als bij specialisatie.

3.3 Formele notaties

Klasse = verzameling van entiteiten

$Z = \{S_1, S_2, \dots, S_n\}$ is een specialisatie van $G \Leftrightarrow \forall i : G/S_i$ is een superklasse/subklasse relatie.

Z is totaal indien $\cup S_i = G$, anders partiëel.

Z is disjunct indien $\forall i, j, i \neq j : S_i \cap S_j = \emptyset$, anders overlappend.

Een categorie T is een deelverzameling van de unie van haar superklassen: $T \subseteq D_1 \cup D_2 \cup \dots \cup D_n$

4 Relationele algebra en relationele calculus

Het opstellen van een database is niet genoeg. Je moet er uiteraard ook iets mee kunnen doen. Dat gaan we in dit en volgende hoofdstukken behandelen. Dit hoofdstuk heeft als doel een relationele taal (query language) op te zetten dat gebruikt kan worden om de database te manipuleren.

De voorbeelden die gebruikt zullen worden, zullen gaan over figuur 3.6.

4.1 Relationale algebra

4.1.1 Selectie

Selectie van tupels (= rijen uit een tabel):

$$\sigma_{\langle \text{Selectiecriteria} \rangle}(R)$$

In dit selectiecriteria mogen de volgende logische symbolen gebruikt worden: $=, \neq, <, >, \leq, \geq, \wedge, \vee, \neg$
Merk op dat een opeenvolging van selecties ook voorgesteld kan worden door een conjunctie van deze beperkingen:

$$\sigma_3(\sigma_2(\sigma_1(R))) = \sigma_{1 \wedge 2 \wedge 3}(R)$$

Voorbeeld:

$$\sigma_{\text{Salary} > 40000}(\text{EMPLOYEE})$$

Dit levert de vierde en achtste rij van de relatie EMPLOYEE.

4.1.2 Projectie

Selectie van kolommen.

$$\pi_{\langle \text{Attributenlijst} \rangle}(R)$$

Dubbele tupels worden verwijderd, waardoor de kardinaliteit kan dalen (ze kan in ieder geval niet stijgen).

Een opeenvolging van projecties $\pi_Y(\pi_X(R))$ is idempotent (enkel de laatste, buitenste projectie moet namelijk uitgevoerd worden, de binnenste projecties zijn dus redundant) en kan enkel indien:

$Y \subseteq X$ (anders worden er attributen gebruikt die niet meer bestaan)

Voorbeeld:

$$\pi_{\text{Fname}, \text{Lname}}(\text{EMPLOYEE})$$

Dit levert de eerste en derde kolom.

4.1.3 Combinaties

Uitdrukkingen kunnen gecombineerd worden:

$$\pi_{\text{Fname}, \text{Lname}}(\sigma_{\text{Salary} > 40000}(\text{EMPLOYEE})) \text{ (Geef de voor- en achternaam van de werknemers die meer dan 40000 verdienen.)}$$

Dit levert volgende tabel:

Fname	Lname
Jennifer	Wallace
James	Borg

Er kan ook gebruik gemaakt worden van variabelen:

$$\text{SELECTION} = \sigma_{\text{Salary} > 40000}(\text{EMPLOYEE})$$

$$\text{RESULT} = \pi_{\text{Fname}, \text{Lname}}(\text{SELECTION})$$

Hernoeming is ook mogelijk:

$$\text{SELECTION}(\text{Firstname}, \text{Lastname}) = \sigma_{\text{Salary} > 40000}(\text{EMPLOYEE})$$

Uiteraard moet er dan met deze nieuwe namen voortgewerkt worden.

Het gebruik van unie \cup , doorsnede \cap en verschil $/$ is toegestaan indien mogelijk (Beide relaties moeten uiteraard dezelfde graad en domein hebben.). Dubbele tupels worden weggelaten.

4.1.4 Carthesisch product

$$Q = R * S$$

Dit geeft alle mogelijk combinaties van tupels. (Indien R 3 tupels heeft en S 7, dan heeft Q er 21.) Dubbele attributen blijven bestaan.

4.1.5 Join operator

$$Q = R \bowtie_F S$$

Hetzelfde als het carthesisch product, enkel met een selectiecriterium.

Voorbeeld:

$DEPARTMENT \bowtie_{DEPARTMENT.Dnumber=DEPT_LOCATION.Dnumber} DEPT_LOCATION$

Dit geeft vijf tupels.

Equi-join: Join waarbij enkel '=' wordt gebruikt in het criterium (zoals het bovenstaand voorbeeld).

Natuurlijke join: Equi-join, waarbij overtollige attributen worden weggelaten. $Q = R *_F S$

F kan genoteerd worden als een conditie, maar ook als een attribuut, aangezien enkel gelijkheid mag worden gebruikt. Indien er geen attributen gegeven worden, wil dat zeggen dat alle overeenkomstige attributen gelijk moeten zijn.

4.1.6 Deling

$$Q = R \div S$$

Het omgekeerde van het carthesisch product.

Q is de maximale relatie waarvoor geldt dat: $Q * S \subseteq R$

Q bevat enkel de attributen van R die niet in S zitten.

4.1.7 Aggregaatfuncties

Functies zoals $SUM, AVERAGE, MAX, MIN, COUNT$.

$attributen \mathcal{S}_{functie}(R)$

Voorbeeld:

$Dno \mathcal{S}_{MAXSalary}(EMPLOYEE)$

Dit geeft de tabel van werknemers, gesorteerd op Dno, waarbij voor elke Dno het maximum salaris is weergegeven. (Er zijn dus drie tupels.)

4.1.8 Uitwendige join

De gewone join levert enkel informatie indien tupels aan een bepaalde voorwaarde voldoen. De uitwendige join wil deze tupels wel weergeven (dit geeft NULL waarden).

Er zijn twee uitwendige joins: de linkse $Q = R] \bowtie S$ en de rechtse $Q = R \bowtie [S$.

Voorbeeld:

$EMPLOYEE] \bowtie_{Ssn=Mgr_ssn} DEPARTMENT$

zal evenveel tupels leveren als EMPLOYEE er heeft (8), maar zal NULL waarden geven bij sommige attributen uit DEPARTMENT.

De rechtse join zal 3 tupels leveren (en in dit geval geen NULL waarden bevatten).

4.1.9 Varianten op unie

U^+ : De uitwendige unie (NULL waarden voor attributen die niet van toepassing zijn)

U^- : Enkel gemeenschappelijke attributen blijven behouden.

4.2 Relationale calculus

Relationele calculus maakt gebruik van eerste orde predikatenlogica.

Basisvorm van een query:

$\{t.A_1, t.A_2, \dots, t.A_n \mid formule(t)\}$ (A_i = attribuut)

Voorbeeld:

$\{t.Fname | EMPLOYEE(t) \text{ and } t.Sex = 'M' \text{ and } t.Salary < 30000\}$

De betekenis van deze query is vrij triviaal en levert 'Ahmad' op.

In de relationele calculus kan gebruik worden gemaakt van quantoren (\forall, \exists), ontkenning (\neg), implicatie (\Rightarrow) en equivalentie (\leftrightarrow).

5 SQL - Deel 1

5.1 Inleiding

5.1.1 Schema

CREATE SCHEMA *name*

AUTHORIZATION *name*

5.1.2 Tabel

CREATE TABLE *name*

(*column name column type attribute constraint*

...

table constraint)

column type: Soort van variabelen (INT, FLOAT, DEC(*i* (precisie), *j* (schaal)), CHAR(*n*), VARCHAR(*n*), BIT(*n*), BIT VARYING(*n*), BOOLEAN, DATE, TIME, TIMESTAMP)

attribute constraint:

NOT NULL

table constraint:

PRIMARY KEY(*attribute*),

FOREIGN KEY(*attribute*) REFERENCES *TABLE*(*key*)

5.1.3 Definitie van een attribuut

CREATE DOMAIN *attribute name* AS *type*

5.1.4 Restricties op tabel

CONSTRAINT *name* PRIMARY KEY(*attribute*) ON DELETE SET NULL ON UPDATE CASCADE (voer verandering door, om integriteit te bewaren)

5.1.5 Drop behaviour (schema/tabel)

CASCADE: Alles wordt weggelaten (inclusief eventuele views)

RESTRICT: Enkel weglaten indien geen verwijzingen ernaar

ALTER TABLE *TABLE NAME*

DROP COLUMN *column name* CASCADE;

5.2 Queries

Algemene structuur:

SELECT *attributen* (projectie)

FROM *tabel(len)* (cartesisch product)

WHERE *condities* (selectie)

5.2.1 Dubbele attribuutnamen

In het geval van dubbele attribuutnamen: *tabel.attribuut*

5.2.2 Aliasen

```
SELECT E.Ssn, S.Ssn
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE ...
```

5.2.3 DISTINCT

SQL elimineert niet automatisch alle dubbele tupels.

Vandaar het gebruik van DISTINCT.

```
SELECT DISTINCT attribute
```

...

5.2.4 Verzamelingen-bewerkingen

UNION, INTERSECT, EXCEPT

(SELECT ...) *BEWERKING* (SELECT ...)

Gebruik het sleutelwoord ALL na een bewerking indien gewerkt wordt met multisets (members zijn niet uniek).

5.2.5 LIKE

Patroonherkenning.

...

```
WHERE Date LIKE '2011____'
```

5.3 Operatoren

+, -, *, / kunnen ook gebruikt worden.

Net zoals concatenatie (voor strings): ||

BETWEEN: BETWEEN 1 and 3

ALL en ANY: a \neq ANY b (c

5.4 Gegevens ordenen

...

```
WHERE ...
```

```
ORDER BY attribute [ASC/DESC]
```

5.5 Geneste queries

De IN operator: duidt aan dat een tuple in een bepaalde verzameling zit.

```
SELECT ...
```

```
FROM ...
```

```
WHERE ... IN (SELECT ...)
```

6 SQL - Deel 2

Deel 2 gaat over geavanceerdere concepten dan in het vorige hoofdstuk werden behandeld.

6.1 CONTAINS

'Heeft als deelverzameling.' (niet meer geïmplementeerd in de meeste versies van SQL)

6.2 EXISTS

EXISTS(*verzameling*) geeft 1 terug als de verzameling niet leeg is, 0 in het andere geval.

6.3 Expliciete verzamelingen

...
WHERE *attribute* IN (1, 2, 3)

6.4 Join operaties

'FROM' is een join operatie. Deze operaties kunnen ook expliciet opgeschreven worden.
FROM ... NATURAL JOIN ...

6.5 Aggregaatfuncties

COUNT, SUM, MAX, MIN, AVG
SELECT *function(attribute)*

6.6 HAVING

Conditie die achteraf gesteld wordt.
SELECT ..., COUNT(*)
...
HAVING COUNT(*) > 2;

Let hier mee op!

Stel dat de vraag 'Geef voor elk departement met minstens 5 werknemers ...', dan moeten eerst de Dno's gevonden worden van de departementen die meer dan 5 werknemers in dienst hebben, voor de rest gevonden kan worden.

6.7 Toevoegen, weglaten, wijzigen

INSERT INTO *TABLE*
VALUES *row*

DELETE FROM *TABLE*
WHERE *condition*

UPDATE *TABLE*
SET *attribute* = *value*
WHERE *condition*

6.8 VIEW

Een view legt een relatie vast. Niet door middel van tuppels, maar door deze te berekenen uit andere relaties.
CREATE VIEW *view name (attributes)*
AS *query*

Views kunnen queries sterk vereenvoudigen en het is nog steeds beter dan een nieuwe tabel (een view is steed up-to-date omdat deze on-the-fly wordt berekend).

Views kunnen op twee manieren geïmplementeerd worden.

Query modification: De view wordt letterlijk in de gebruikte query ingeplaatst.

Het wijzigen van views is niet altijd op een eenduidige manier bepaald. Dit wel eenduidig bepaald indien de view een primaire sleutel of kandidaatsleutel van een tabel bevat.

6.9 ASSERTIONS

Eerder hebben we al restricties op sleutels en referenties gedefiniëerd.

Als een andere vorm van restricties nodig is, dan kan gebruik worden gemaakt van ASSERTIONS.

CREATE ASSERTION *name*

CHECK *condition*

De CHECK-clausule kan ook gebruikt worden bij het creëren van een domein:

CREATE DOMAIN *name*

CHECK *condition*

6.10 TRIGGERS

Wanneer niet aan een assertion wordt voldaan, dan wordt er een trigger geactiveerd.

CREATE TRIGGER *name*

BEFORE/AFTER *event* (INSERT, UPDATE, ...) ON *table*

FOR EACH ROW

WHEN *condition*

action

7 Functionele afhankelijkheden en normalisatie - deel 1

Bij het ontwerpen van een gegevensbank is het belangrijk dat redundantie zo veel mogelijk vermeden wordt en dat de performantie maximaal is.

Bijvoorbeeld de database van COMPANY zou redundanter zijn indien in de EMPLOYEE relatie ook de departementslocaties zouden zijn opgenomen.

Bij het ontwerpen van een relatieschema moet met het volgende rekening gehouden worden:

- de betekenis van het schema moet makkelijk verklaarbaar zijn;
- redundantie moet vermeden worden, alsook anomalieën;
- vermijd attributen die NULL kunnen zijn;
- een natuurlijke join tussen twee relaties op primaire of verwijssleutels mogen geen onechte tupels opleveren (de keuze van een attribuut dat niet uniek is, kan op deze manier onechte tupels opleveren).

Om tot een database te komen die rekening houdt met bovenstaande zaken, gaan we een database 'normaliseren'.

Om te kunnen normaliseren is het noodzakelijk om te weten wat de afhankelijkheden zijn in het model.

Een afhankelijkheid wordt formeel gedefiniëerd als:

Y is functioneel afhankelijk van X ($X \rightarrow Y$) als en slechts als

$X \neq \emptyset$ en

\forall mogelijke extensies van R : $t_1, t_2 \in r$ en $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$

De sluiting van F (F^+ genoteerd) is de verzameling functionele afhankelijkheden die door F geïmpliceerd worden.

7.1 Afleidingsregels

$Y \subseteq X \Rightarrow X \rightarrow Y$

$X \rightarrow Y \Rightarrow XZ \rightarrow YZ$

$X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

$X \rightarrow YZ \Rightarrow X \rightarrow Y$

$X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$

$X \rightarrow Y, WY \rightarrow Z \Rightarrow WX \rightarrow Z$

Hoe kunnen we nu alle functionele afhankelijkheden vinden?

Stel de sluiting van X t.o.v. F op. ($= X_F^+$)

Algoritme: Voor elke $X \rightarrow Y \in F$, bepaal X_F^+

Voorbeeld:

$F = \{A \rightarrow B, \{A, C\} \rightarrow D\}$

$\{A\}^+ = \{A, B\}$

$\{A, C\}^+ \rightarrow \{A, B, C, D\}$

7.2 Overdekking, equivalentie

E wordt overdekt door $F \Leftrightarrow E \subseteq F^+$

E en F zijn equivalent $\Leftrightarrow E^+ = F^+$

Overdekking nagaan: $\forall X \rightarrow Y$ in E , bepaal X_F^+ en controleer of $Y \subseteq X_F^+$

(Het is belangrijk dat je deze zin volledig begrijpt.)

7.3 Minimale verzameling

F is een minimale verzameling \Leftrightarrow het weglaten van een afhankelijkheid of een attribuut aan de rechter- of linkerkant geen equivalente verzameling oplevert.

Minimale overdekking: Overdekking die minimaal is.

Hoe de minimale overdekking vinden?

Stap 1:

Splits alle rechterleden op volgens het volgende principe:

$\{A \rightarrow BC\} \Rightarrow \{A \rightarrow B, A \rightarrow C\}$

Stap 2:

Iets gelijkaardig doen we voor het linkerlid. Als het linkerlid meervoudig is, probeer er dan een element uit te schrappen. Als dat element deel uitmaakt van de transitieve sluiting van de andere elementen, dan mag dit element definitief geschrapt worden. Anders niet.

$\{A \rightarrow B, BC \rightarrow A\} \Rightarrow \{A \rightarrow B, C \rightarrow A\}$

In bovenstaand voorbeeld is er gekozen B te schrappen. Dit mag omdat $B \in C_F^+$ (het overblijvende element).

Stap 3:

In deze stap wordt nagegaan of sommige afhankelijkheden overbodig zijn. Stel $X \rightarrow Y$ zich in de overdekking bevindt. Als Y nog steeds in de transitieve sluiting van X bevindt als deze afhankelijkheid geschrapt wordt, mag deze definitief geschrapt worden.

$\{A \rightarrow B, A \rightarrow C, C \rightarrow B\} \Rightarrow \{A \rightarrow C, C \rightarrow B\}$

7.4 Normaalvormen

In het begin van het hoofdstuk werd er gesproken over normaalvormen. We hebben nu genoeg achtergrond om dit begrip te bestuderen.

Normaalvormen zijn opgedeeld in graden. Hoe hoger de graad van de normaalvorm, hoe minder afhankelijkheden zich kunnen voordoen. Het is dus de bedoeling om een schema om te vormen tot een zo hoog mogelijke normaalvorm.

Eén van de voornaamste manieren om een schema te 'normaliseren' is decompositie. Dit wil zeggen dat schema's die niet voldoen aan een bepaalde normaalvorm opgedeeld zullen worden in kleinere relatieschema's.

7.4.1 Enkele definities

Merk op dat een deel van deze definities al uitgelegd zijn in vorige hoofdstukken. Ze worden deze keer wat formeler uitgelegd.

Supersleutel: K is een supersleutel voor een relatie R met schema $\langle U, F \rangle \Leftrightarrow K \subseteq U$ en $K \rightarrow U \in F^+$.

Kandidaatsleutel: K is een kandidaatsleutel voor een relatie R met een schema $\langle U, F \rangle \Leftrightarrow K$ is een zo

klein mogelijke supersleutel voor R .

Sleutelattribuut: A is een sleutelattribuut voor de relatie $R \Leftrightarrow$ Er bestaat een kandidaatsleutel K voor R waarvoor geldt $A \in K$.

Zowel de primaire als de alternatieve sleutel zijn sleutelattributen.

7.4.2 De 0de normaalvorm

Een relatieschema $S_R = \langle U, F \rangle$ is in de nulde normaalvorm. Er zijn geen voorwaarden opgelegd aan de attributen.

7.4.3 De 1ste normaalvorm

1NF: Het domein van U van het relatieschema $S_R = \langle U, F \rangle$ is enkelvoudig.

In 1NF brengen: Samengestelde attributen opsplitsen en voor een meerwaardig attribuut meerdere tupels voorzien of een nieuwe relatie creëren.

7.4.4 De 2de normaalvorm

Als $X \rightarrow Y$, dan is Y partieel functioneel afhankelijk indien $Z \subset X \Rightarrow Z \rightarrow Y$.

2NF: Voor geen enkel niet-sleutelattribuut A van U van het relatieschema $S_R = \langle U, F \rangle$ geldt dat A partieel functioneel afhankelijk is van een kandidaatsleutel R .

Of: Voor elk niet-sleutel attribuut is de hele primaire sleutel nodig om het te determineren.

In 2NF brengen:

Voor elk attribuut A dat partieel functioneel afhankelijk is van een kandidaatsleutel K :

Zoek een subset K' van K waarvan A volledig functioneel afhankelijk is.

Elimineer A uit U .

Maak een nieuw relatieschema met attributenverzameling $K' \cup A$.

Voorbeeld: $\{\underline{AB} \rightarrow C, \underline{AB} \rightarrow D, \underline{B} \rightarrow E\} \Rightarrow \{\underline{AB} \rightarrow C, \underline{AB} \rightarrow D\}, \{\underline{B} \rightarrow E\}$

7.4.5 De 3de normaalvorm

$X \rightarrow Y$ is een triviale functionele afhankelijkheid $\Leftrightarrow Y \subseteq X$

$X \rightarrow Y$ is een transitieve functionele afhankelijkheid \Leftrightarrow

Er bestaat een Z :

Z is geen deelverzameling van de kandidaatsleutel

Z is niet triviaal afhankelijk van X

Y is niet triviaal afhankelijk van Z

$X \rightarrow Z \rightarrow Y$

3NF: 2NF, waarbij voor geen enkel niet-sleutelattribuut A van U geldt dat A transitief functioneel afhankelijk is van één of andere kandidaatsleutel.

In 3NF brengen:

Transitieve relaties opsplitsen.

Voorbeeld:

$\{\underline{A} \rightarrow B, B \rightarrow C\} \Rightarrow \{\underline{A} \rightarrow B\}, \{\underline{B} \rightarrow C\}$

7.4.6 Boyce-codd normaalvorm (BCNF)

Bij 3NF zijn binnen sleutels nog functionele afhankelijkheden toegestaan.

BCNF: 3NF, waarbij voor geen enkel sleutelattribuut A van U geldt dat A transitief functioneel afhankelijk

is van één of andere kandidaatsleutel.
De werkwijze is exact hetzelfde als deze bij 3NF.

8 Normalisatie - deel 2

8.1 Decompositie

Wanneer decompositie wordt toegepast, is het belangrijk dat er geen informatie of afhankelijkheden verloren gaan.

Afhankelijkheden-bewardende decompositie:

Zoek de minimale overdekking.

Voor elke X die ergens aan de linkerkant van een afhankelijkheid voorkomt, maak een schema $\{X \cup A_1 \cup A_2 \cup \dots \cup A_n\}$ (unie van X met alle attributen die ervan afhankelijk zijn)

Plaats alle attributen die over blijven in een apart relatieschema.

Merk op dat dit algoritme afhankelijkheden bewaart, maar informatie kan verliezen.

Voorbeeld:

$\{A \rightarrow B\}, \{A \rightarrow C\}, \{AB \rightarrow C\} \Rightarrow \{A, B\}, \{A, C\}$

8.2 Het Chase algoritme

Het chase algoritme gaat na of dat een decompositie verliesloos is of niet.

De formele notatie van het algoritme is een beetje overkill. Vandaar dat het algoritme hier uitgelegd zal worden via een voorbeeld:

$\{A \rightarrow B\}, \{C \rightarrow DE\}, \{AC \rightarrow F\}$ met bijbehorende decompositie: $\{A, B\}, \{C, D, E\}, \{A, C, F\}$

De bedoeling is dat er eerst een tabel wordt opgesteld aan de hand van de decompositie. In de cursus en in de slides wordt er gewerkt met a's en b's en een lawine aan indices. Dat is niet nodig. 1'tjes en 0'tjes zijn veel overzichtelijker.

Elke rij van de tabel stelt een relatie van de decompositie voor. Zet een 1 waar dat een attribuut in de huidige relatie voorkomt. Dit geeft voor het gebruikte voorbeeld:

A	B	C	D	E	F
1	1	0	0	0	0
0	0	1	1	1	0
1	0	1	0	0	1

Hierna is het de bedoeling om de Y attributen gelijk te stellen aan elkaar indien de X attributen dat ook zijn. Met andere woorden: zet een 1 op de plaatsen van de Y attributen indien de X attributen allemaal 1 zijn voor elke afhankelijkheid. De decompositie is verliesloos indien er in de onderste rij enkel 1'tjes staan.

Dit is nog vrij abstract. Vandaar de toepassing op de tabel:

De eerste afhankelijkheid is: $A \rightarrow B$. In de A kolom zijn 2 1'tjes. In de rijen waar de 1'tjes staan moet dus ook een 1 staan in de B kolom. Dit geeft:

A	B	C	D	E	F
1	1	0	0	0	0
0	0	1	1	1	0
1	1	1	0	0	1

De tweede afhankelijkheid $C \rightarrow DE$ toepassen geeft:

A	B	C	D	E	F
1	1	0	0	0	0
0	0	1	1	1	0
1	1	1	1	1	1

De derde afhankelijkheid $AC \rightarrow F$ zal niets meer toevoegen omdat er geen enkele andere rij is dan de derde die een 1 heeft staan bij zowel A als C . Maar dat hoeft ook niet. De onderste rij staat vol al 1'tjes. We kunnen besluiten dat de decompositie verliesloos is.

8.3 Algoritme voor het vinden van een sleutel

$K = R$

Voor elk attribuut A in K , als $(K - A)^+$ alle attributen van R bevat, dan $K = K - \{A\}$.

8.4 Nadelen van de decompositiemethoden

- functionele afhankelijkheden moeten op voorhand bekend zijn (moeilijk bij een grote gegevensbank);
- algoritmen zijn niet deterministisch (meerdere oplossingen);
- (dure) joins zijn nodig om de originele informatie te reconstrueren

8.5 Meerwaardige afhankelijkheid en de 4de normaalvorm

Weeral overkill in de slides. We houden het op de woordelijke uitleg.

X bepaalt eenduidig een verzameling Y 's. Welke Y 's hierbij horen hangt niet af van andere attributen.

Notatie: $X \twoheadrightarrow Y$

Voorbeeld:

$\{A \rightarrow B, A \rightarrow C\}$

A	B	C
Good	0	Excellent
Good	0	Splendid
Good	1	Excellent
Good	1	Splendid

Het is duidelijk dat indien er meerdere attributen gemultidetermineerd worden, er dan een explosie aan combinaties plaatsvindt (van exponentiële orde).

Daarom worden meerwaardige afhankelijkheden geweerd uit 4NF.

Triviale meervoudige afhankelijkheid: $X \twoheadrightarrow Y$ is triviaal $\Leftrightarrow Y \subseteq X$ of $X \cup Y = U_R$

4NF: Voor elke niet-triviale meerwaardige afhankelijkheid $X \twoheadrightarrow Y$ van F^+ geldt dat X een supersleutel is.

In 4NF brengen:

Voor elke $X \rightarrow Y_i$ een aparte tabel maken, zodat X niet meer meerwaardig is.

Merk wel op dat het mogelijk is dat er afhankelijkheden verdwijnen.

8.6 Join-afhankelijkheden en de 5de normaalvorm

Join-afhankelijkheid: Indien een join uitgevoerd wordt op alle attributen afzonderlijk (dus $\pi_{U_1}(r) * \pi_{U_2}(r) * \dots * \pi_{U_n}(r)$) kan de oorspronkelijke relatie altijd teruggevonden worden.

Voorbeeld:

Als volgende tupels voorkomen:

A	B	C
x	y	
x		z
	y	z

dan komt ook deze tupel voor:

A	B	C
x	y	z

Een join afhankelijkheid is triviaal indien één van de tabellen in de join alle attributen bevat van de tabel.

5NF: Voor elke niet-triviale join-afhankelijkheid $JD(U_1, \dots, U_n)$ van F^+ geldt dat U_1, \dots, U_n allemaal supersleutels zijn van R .

In 5NF brengen:

Voor elke $JD(\delta)$ in F_R^+ , splits op volgens δ .

Voorbeeld:

A levert B aan C :

A levert aan B

C gebruikt B

A levert aan C

Dit wordt genoteerd als: $JD(\{A, B\}, \{B, C\}, \{A, C\})$.

Maak dus drie tabellen met deze afhankelijkheden.

9 Geheugen en bestandsorganisatie

De tijd die nodig is voor het zoeken, verplaatsen en opslaan van gegevens is veel hoger dan de tijd die nodig is om gegevens te manipuleren. Het is dus belangrijk om bestanden zo te structureren dat deze tijd geminimaliseerd wordt.

9.1 Bestandsorganisatie: blokken en records

Record: Bij elkaar horende groep gegevens.

Recordlengte kan zowel vast als variabel zijn.

Blok: De entiteit voor plaatsgebruik op de schijf en de bijbehorende gegevenstransport.

Records kunnen spanned (record over meerdere blokken uitgespreid) of unspanned (record in 1 blok (wel meerdere records per blok toegestaan) \rightarrow verloren ruimte) in een blok geplaatst worden.

Bloklengte = B , Recordlengte = L

- Unspanned
Blocking factor ($\#$ records/blok) $bfr = \lfloor B/L \rfloor$
- Spanned
Gemiddelde nr. van records/blok

r = totaal aantal records

Aantal blokken dat nodig is: $b = \lceil r/bfr \rceil$

9.2 Plaatsen van logisch opeenvolgende blokken

- Aaneensluitend: opeenvolgende blokken fysisch aaneensluitend ("contiguous allocation")
- Geketend: elk blok bevat een wijzer naar het volgend blok ("linked allocation")
- Een combinatie: clusters van aaneensluitende blokken, elke cluster bevat een wijzer naar de volgende cluster
- Indexering: Één of meer indexblokken, bevatten enkel wijzers naar de echt gegevens ("indexed allocation")

9.3 Toegang tot en bewerking van bestanden

Het criterium voor de selectie van records dat aangeboden wordt door een besturingssysteem is meestal zeer eenvoudig, een DBMS moet dus ingewikkelde criteria omzetten naar meerdere eenvoudige criteria.

Enkele belangrijke criteria:

file activity: # records dat gebruikt wordt door een toepassing / totaal # records van het bestand

file volatility: # records dat in een bepaalde periode een verandering ondergaat / totaal # records van het bestand

file turnover: # records dat in een bepaalde periode vervangen wordt door nieuwe records / totaal # records van het bestand

file growth: toename van het # records gedurende een bepaalde periode / het oorspronkelijk # records

9.4 Soorten bestanden

9.4.1 Ongeordende bestanden (seriële bestanden)

Toevoegen: achteraan in het laatste blok

Opzoeken: Lineair

Weglaten: veel verloren ruimte

9.4.2 Geordende bestanden (sequentiële bestanden)

Geordend volgens sleutelveld

Lezen van records in sleutelvolgorde → zeer efficiënt

Binair zoeken mogelijk

Aanpassen in sleutelvolgorde gaat snel

Zoeken op ander veld dan sleutel blijft tijdrovend

Toevoegen en verwijderen vereist het verplaatsen van records

Aanpassen van records gaat traag indien de aanpassingen niet in sleutelvolgorde worden aangeboden

9.4.3 Directe bestanden

Het principe van hashing: hashfunctie beeldt waarde k af op adres $f(k)$, op een zodanige manier dat deze adressen zo goed mogelijk gespreid zijn.

Interne hashing: Gegevens en hashfunctie zijn in een centraal geheugen voorgesteld.

Externe hashing: Gegevens (+ eventueel ook hashfunctie) zijn opgeslagen in een bestand.

9.5 Hashing

Sleutelruimte: Verzameling van mogelijke sleutels

Adresruimte: Verzameling van mogelijke adressen

Technieken:

- module, enkel numerieke sleutels (grote priemgetallen!)
- vouwen, enkel numerieke sleutels (in stukken splitsen en optellen)
- kwadrateren en middelste cijfers nemen, enkel numerieke sleutels
- sleutels numeriek coderen, niet-numeriek sleutels

9.5.1 Botsafhandeling

Meerdere records in dezelfde cel.

Open adressering (= gesloten hashing): Neem gewoon dezelfde cel

zeer eenvoudig

lineair zoeken

weglaatalgoritme ingewikkeld

Ketening (= gesloten adressering, open hashing): Verwijzing naar nieuwe cel

Varianten:

Coalesced/separate chaining: Maken gebruik van een linked list als overloopgebied. Het verschil is dat bij coalesced chaining het overloopgebied is geïntegreerd in de huidige adresseringsruimte, terwijl separate chaining gebruik maakt van een gescheiden linked list.

Direct chaining: Alle gegevens staan meteen in het overloopgebied (sequentiële toegang is heel langzaam).

record snel bereikbaar via sleutel

toevoegen en weglaten redelijk eenvoudig

opzoeken via ander sleutelveld heel moeilijk

doorlopen van bestand in sleutelvolgorde kan traag zijn

Meervoudige hashing Na botsing of overloop: andere hashfunctie gebruiken tot alle hashfuncties opgebruikt zijn.

E = gemiddelde zoeklengte

α = vullingsgraad

Knuth's formule:

$E = \frac{1}{2}(1 + 1/(1 - \alpha))$ (bij succes)

$E = \frac{1}{2}(1 + 1/(1 - \alpha)^2)$ (bij falen)

9.6 Externe hashing

Cellen = "buckets"

Elke bucket, of cel, bevat meerdere records.

Indien er toch nog overloop plaatsvindt, dan gebeurt dat in extra overloopcellen.

record snel bereikbaar via sleutel

toevoegen en weglaten redelijk eenvoudig

opzoeken via ander sleutelveld moeilijk

doorlopen van bestand in sleutelvolgorde kan traag zijn

vaste ruimte gereserveerd voor bestand (static hashing)

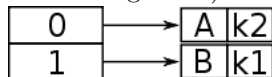
10 Externe hashing

10.1 Dynamische hashing

- Begin met 1 cel
- Bij overloop → cel splitst in twee (volgens de eerstvolgende bit van de hashwaarde)

10.2 Uitbreidbare hashing

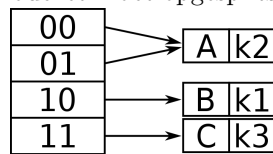
Ook hier werkt men met een tabel. Deze tabel bevat verwijzingen naar buckets (deze buckets hebben een arbitraire grootte).



Laat de hashwaarde van $k1 = 010110$ en de hashwaarde van $k2 = 010110$. Deze twee hashwaarden zijn

gesorteerd volgens hun eerste bit. Beide buckets worden dus geïdentificeerd volgens hun eerste bit. Hun lokale index d' is 1.

Wanneer $k3 = 110110$ moet worden toegevoegd, zal er overloop zijn in de tweede bucket. Vandaar dat deze bucket moet opgesplitst worden:



De index van de eerste bucket blijft 1, deze van de tweede en derde bucket worden 2. De globale index d (de index van de hashtable), is steeds de grootste lokale index. Logischerwijs bevat de hashtable 2^d elementen.

hashtabel blijft klein

uitbreiden van bestand is goedkoop (cel in twee splitsen)

verdubbeling van tabel vraagt veel werk

1 extra indirectie (te verwaarlozen)

11 Lineaire hashing

Lineaire hashing begint met M cellen (traditioneel een macht van 2).

De eerste hashfunctie die gebruikt wordt is $h_0 = k \bmod M$. Het aantal cellen kan variëren. Dat wil dus zeggen dat de hashfunctie ook varieert.

In geval van overloop, gebruik ketening.

Begin met index $i = 0$ en variabele $n = 0$.

Indien een bepaalde load factor wordt overschreden (het aantal gebruikte cellen/het totaal aantal cellen), voeg een cel toe. Maak een nieuwe hashwaarde $h_{i+1} = k \bmod 2^{i+1}M$ en hoog n op. Alle elementen die voor de cel n zitten, daarvan zullen de hashwaarden opnieuw moeten berekend worden via de nieuwe hashfunctie $i + 1$. Indien n groter wordt dan $2^i * M$, hoog dan i op. Zet $n = 0$. Bereken h_0 en h_1 opnieuw.

Voorbeeld:

Aantal elementen per bucket = 2

Split als load factor > 0.7

Initialiseer $M = 4$

Toe te voegen: 5, 9, 3, 6, 22, 4, 7

Initialiseer, $M = 4$, $i = 0$, $n = 0$, load factor = $0 < 0.7$

Voeg toe: 5, $M = 4$, $i = 0$, $n = 0$, load factor = $1/8 < 0.7$

Eerste hashfunctie: $h_0 = k \bmod 4$, $h_0(5) = 1$

	5		

Voeg toe: 9, $M = 4$, $i = 0$, $n = 0$, load factor = $1/4 < 0.7$

$h_0(9) = 1$

	5		
	9		

Voeg toe: 3, $M = 4$, $i = 0$, $n = 0$, load factor = $3/8 < 0.7$

$h_0(3) = 3$

	5		3
	9		

Voeg toe: 6, $M = 4$, $i = 0$, $n = 0$, $load\ factor = 1/2 < 0.7$

$h_0(6) = 2$

	5	6	3
	9		

Voeg toe: 22, $M = 4$, $i = 0$, $n = 0$, $load\ factor = 5/8 < 0.7$

$h_0(22) = 2$

	5	6	3
	9	22	

Voeg toe: 4, $M = 4$, $i = 0$, $n = 0$, $load\ factor = 3/4 > 0.7 \rightarrow split!$

$h_0(4) = 0$

4	5	6	3
	9	22	

 \rightarrow

	5	6	3	4
	9	22		

$M = 4$, $i = 0$, $n = 1$, $load\ factor = 3/5 < 0.7$

$h_1 = k \bmod 2M$

$h_1(4) = 4$

Voeg toe: 7, $M = 4$, $i = 0$, $n = 1$, $load\ factor = 7/10 = 0.7$

$h_0(7) = 3$

	5	6	3	4
	9	22	7	

12 Indexstructuren

Index: Een gegevensstructuur die de toegang op dat bestand via een bepaald veld efficiënter maakt. Deze index kan opgeslagen zijn in een centraal geheugen of in een bestand in een extern geheugen.

12.1 Soorten indexen

Primaire index: Index op het veld dat de ordening van het bestand bepaalt.

De records zijn uniek.

Clusterindex: Index op het veld dat de ordening van het bestand bepaalt.

De records zijn niet noodzakelijk uniek.

Secundaire index: Index op een ander veld dan het veld dat de ordening van het bestand bepaalt.

12.1.1 Primaire indexen

- Geordend volgens sleutelwaarden
- Index bevat 1 record per blok: "ankerrecord" (= eerste of laatste record in een blok) en het adres

Blok heeft vaste lengte \rightarrow overloop

Ankerrecords kunnen veranderen!

Oplossing: overloopgebieden en markering

Performantie:

records = r

blokken = b

recordlengte = R

bloklengte = B

blocking factor $bfr = \lfloor B/R \rfloor$

De performantie wordt weergegeven door het aantal schijftoegangen bij binair zoeken. $= \lceil \log_2 b \rceil = \lceil \log_2 \lfloor r/bfr \rfloor \rceil$.

12.1.2 Clusterindex

Geordend volgens niet-uniek veld.

Aangezien er meerdere velden dezelfde waarden hebben, wijst de clusterindex naar de eerste blok dat deze waarde bevat.

Bij toevoeging zouden de indexen kunnen veranderen.

Oplossing: Voor elke waarde een aparte blok gebruiken.

12.1.3 Secundaire index

Sleutelveld

De index verwijst naar een sleutelveld dat niet geordend is. Maar de index zelf is wel geordend.

De index kan heel groot worden, maar aangezien deze geordend is, is binair zoeken mogelijk.

De performantie wordt op dezelfde manier berekend als bij de primaire index.

Niet-sleutelveld

De index verwijst onrechtstreeks naar een niet-sleutelveld dat niet geordend is. De index zelf is wel geordend.

De index verwijst naar een blok recordwijzers.

Deze blok recordwijzers verwijzen naar dezelfde sleutels in het bestand (niet-sleutelvelden zijn niet meer uniek). Dit zorgt voor 1 indirectie meer.

12.2 Meerdere niveau's

Principe: Index bouwen bovenop index.

De blocking factor bfr_i is even groot voor alle indexen (= "fan-out" fo).

Aantal blokken op niveau $k = \lceil r_1 / fo^k \rceil$ (r_1 = aantal records op niveau 1 = totaal aantal records)

snelle toegang tot bestand, toevoeging en weglating efficiënt

overloop is vertragend en verkrist ruimte + verplichte reorganisatie vraagt tijd

12.3 Boomstructuren

Gewone binaire zoekboom is efficiënt, maar is ongebalanceerd.

12.3.1 B-bomen

B-boom van orde p , is een zoekboom waarvoor geldt dat:

- de wortel heeft minstens 2 knopen;
- elke andere knoop heeft minstens $\lceil p/2 \rceil$ en maximum p kinderen;
- alle bladeren zitten even diep.

Maximale hoogte $h = \log_{\lceil p/2 \rceil} (n + 1) / 2$

Berekening van de orde van een B-boom:

orde = p

grootte blokadres = P

grootte veld = V

grootte recordadres = R

grootte blok = B

Elke knoop moet binnen 1 blok passen. Dat wil zeggen: $\text{orde} * \text{blokadres} + (\text{orde} - 1) * (\text{veld} + \text{record}) = \text{blokgrootte}$

Of: $pP + (p - 1)(V + R) = B$

Typisch voor een B-boom is dat de f_o klein is. Hierdoor is deze boom enkel bruikbaar wanneer er met een klein aantal records, of een kleine recordgrootte wordt gewerkt.

12.3.2 B⁺-bomen

Recordadressen enkel in knopen, niet meer in interne knopen. Dit zorgt ervoor dat de orde van de interne knopen veel groter is.

Aan het einde van elk blad: een wijzer naar het volgende blad (voor sequentiële doorloop).

Orde van de interne knopen:

$$pP + (p - 1)V = B$$

Orde van de bladeren:

$$p(V + R) + P = B$$

Toevoegen van een sleutel:

Indien blad vol: Splits blad in de helft. De eerste helft (afgerond naar boven) blijft, de tweede helft wordt een nieuw blad. De laatste waarde van de eerste helft moet toegevoegd worden aan ouderknoop. (Indien ouderknoop vol, doe recursief hetzelfde).

Verwijderen van een sleutel:

Indien de sleutel ergens in de interne knopen voorkomt: vervang door de waarde net links ervan.

Indien er onderloop optreedt: steel enkele waarden van een naburig blad, voeg eventueel 2 bladeren samen en verwijder 1 wegwijzer uit bovenliggende knoop.

Indien onderloop in interne knoop: herverdeel

12.3.3 B*-bomen

Elke knoop is minstens 2/3 gevuld.

Splits wanneer 2 naburige knopen vol zijn.

12.4 Indexen op meerdere velden

Velden combineren door meerdere velden als 1 veld te beschouwen of door "partitioned hashing" (= resultaat is combinatie van aparte hashfuncties).

In plaats van rijen met adressen, nu matrices met adressen.

13 Begrippen van transactieverwerking

Transactie: De uitvoering van een programma dat een gegevensbank raadpleegt of wijzigt.

Interleaved uitvoering: serieel

Simultane uitvoering: parallel

We zullen ons verder baseren op interleaved uitvoering.

Concurrentie: De gelijktijdige uitvoering van transacties.

In geval van concurrentie kunnen er aanpassingen verloren gaan waardoor bepaalde gegevens niet meer correct zijn.

Dirty read: Een transactie schrijft iets weg en faalt. De gewijzigde waarden worden opnieuw hersteld, maar ondertussen heeft een andere transactie deze foute waarden al gebruikt.

Foutieve somming: Sommige waarden van voor en sommige van na een wijziging worden opgeteld.

Niet herhaalbare lezing: Twee maal dezelfde waarde lezen levert een ander resultaat op.

13.1 Herstel

Herstel: Een transactie moet ofwel volledig uitgevoerd worden ofwel helemaal niet.

Systeemlog: Noteert alle transacties die waarden in een gegevensbank wijzigen.

Herstellen na falen:

- Transactie volledig ongedaan maken (UNDO).
- Transactie goed afwerken (REDO).

Wat te kiezen? Commit point: Punt waarop beslist wordt dat de transactie goed afgewerkt moet worden in plaats van ongedaan gemaakt.

Commit point wordt bereikt indien alle bewerkingen van de transactie met succes zijn uitgevoerd en alle bewerkingen ook op de log zijn geregistreerd.

Transacties moeten 'ACID' zijn.

Atomicity (ondeelbaarheid)

Consistency preservation (databank moet consistent blijven)

Isolation (geen interferentie met andere transacties)

Durability (effect mag niet verloren gaan)

13.2 Transactieroosters

2 operaties **conflicteren** indien er ten minste één write wordt uitgevoerd door 2 verschillende transacties op hetzelfde gegevenselement.

Een rooster S voor n transacties T_i is **volledig** als dit rooster alle operaties van de transacties T_i bevat, de volgorde van de operaties binnen T_i behoudt en de volgorde vastligt voor elk paar conflicterende operaties.

Een rooster is **herstelbaar** \Leftrightarrow een transactie die gecommit is, nooit meer ongedaan moet worden gemaakt.

Cascading rollback: Een transactie terugrollen kan ervoor zorgen dat er nog meer transacties moeten teruggerolled worden.

Cascadeloze roosters: Garanderen dat er geen cascading rollbacks nodig zijn.

Voldoende voorwaarde: Elke transactie leest enkel waarden geschreven door transacties die al gecommit hebben.

Strikte roosters: Elke transactie leest en schrijft enkel waarden die geschreven zijn door transacties die al gecommit hebben.

13.3 Serialiseren van roosters

Serieel rooster: De transacties worden na elkaar uitgevoerd.

Beperking op concurrentie.

Een rooster is **serialiseerbaar** indien het rooster equivalent is met een serieel rooster (dat dezelfde transacties bevat).

Verschillende soorten equivalentie:

- resultaat-equivalent: gegeven beginvoorwaarden, zelfde resultaat (te zwak, twee verschillende functies kunnen toevallig hetzelfde resultaat geven)
- conflict-equivalent (conflict-seraliseerbaar): volgorde van twee conflicterende operaties steeds hetzelfde in beide roosters.
- view-equivalent (view-seraliseerbaar): Elke overeenkomstige leesopdracht in S_1 leest dezelfde waarden als in S_2 en de laatst geschreven waarde voor een item is in beide roosters hetzelfde.

Conflict-seraliseerbaarheid is te testen met "**precedence-graph**". Dit is een graaf die de volgorde van de transacties aanduidt.

Precedence-graph:

- Maak voor elke transactie een knoop.
- Maak een boog van T_i naar T_j . Als T_j een $read(X)$ na een $write(X)$, een $write(X)$ na een $read(X)$ of een $write(X)$ na een $write(X)$ van T_i uitvoert.
- Het rooster is serialiseerbaar \Leftrightarrow er zijn geen cycli.

Problemen met testen van serialiseerbaarheid:

- Interleaving van operaties wordt bepaald door het besturingssysteem.
- Transacties worden continu aangeboden: Het begin en einde van de roosters is moeilijk te voorspellen.

Het is dus beter om bij het opstellen van de transacties serialiseerbaarheid te garanderen.

14 Transactiecontrole II: Technieken voor concurrentiecontrole en herstel

14.1 Concurrentiecontrole - Locking

Een **grendel (lock)** is een variable dat bij de gegevensbank hoort en de status beschrijft van een element.

14.1.1 Binaire grendels

2 toestanden:

$lock(X) = 1$: X is niet toegankelijk

$lock(X) = 0$: X is wel toegankelijk

2 bewerkingen:

$lock_item(X)$: een transactie vraagt toegang tot X

$unlock_item(X)$: een transactie geeft X weer vrij

Regels voor vergrendeling:

T moet:

- $lock_item(X)$ uitvoeren voor $read_item(X)$ of $write_item(X)$
- $unlock_item(X)$ uitvoeren nadat alle $read_item(X)$ of $write_item(X)$ uitgevoerd zijn

T mag geen:

- $lock_item(X)$ uitvoeren als het al een grendel heeft op X
- $unlock_item(X)$ uitvoeren als het geen grendel heeft op X

14.1.2 Read/write locks

Ook gedeelde/exclusieve grendels genoemd.

3 toestanden: niet vergrendeld, lees-grendel, schrijf-grendel

3 bewerkingen:

$read_lock(X)$, $write_lock(X)$ en $unlock_item(X)$

Regels voor vergrendeling:

T moet:

- $read_lock(X)$ of $write_lock(X)$ uitvoeren voor een $read_item(X)$
- $write_lock(X)$ uitvoeren voor een $write_item(X)$
- $unlock(X)$ uitvoeren nadat alle $read_item(X)$ of $write_item(X)$ uitgevoerd zijn

T mag geen:

- $read_lock(X)$ of $write_lock(X)$ uitvoeren als het al een grendel heeft op X
- $unlock_item(X)$ uitvoeren als het geen grendel heeft op X

Conversie van grendels:

Als T als enige een leesgrendel heeft op X , dan mag deze geupgraded worden naar een schrijfgrendel.

Een schrijfgrendel mag altijd gedowngraded worden naar een leesgrendel.

Lees-/schrijf-vergrendeling vermijdt bepaalde problemen, maar levert geen serialiseerbaarheid. Daarom moet er een protocol ingevoerd worden.

14.1.3 Twee-fasen-vergrendeling

Alle vergrendelingen van een transactie gebeuren voor de eerste ontgrendeling.

Garandeert serialiseerbaarheid.

Vermijdt geen deadlock/starvation.

Varianten:

- Basisversie: zoals hierboven gezien
- Conservatieve versie: Alle grendels plaatsen voor de transactie begin (vermijdt deadlocks, maar grendels niet altijd op voorhand gekend).
- Strikte/rigoureuze versie: Geen enkele grendel vrijgeven voor commit of abort (garandeert strikt rooster, maar is niet deadlockvrij).

14.2 Concurrentiecontrole - Timestamps

Deadlock prevention: Conservatieve 2FV (grote beperking op concurrentie) of het gebruik van tijdstempels.

Tijdstempels: T_1 start voor T_2 : $TS(T_1) < TS(T_2)$

Mogelijke schema's:

- wait-die schema: als $TS(T_i) < TS(T_j)$, dan moet T_i wachten tot er een grendel vrijkomt, anders wordt T_j afgebroken en later terug herstart met dezelfde tijdstempel.
- wound-wait schema: als $TS(T_i) < TS(T_j)$, dan wordt T_j afgebroken, anders moet deze wachten tot er een grendel bijkomt.

	Wait/Die	Wound/Wait
Y_i needs a resource held by Y_j	Y_i waits	Y_j dies
Y_j needs a resource held by Y_i	Y_j dies	Y_j waits

De jongste transactie is degene die altijd afgebroken wordt → minder werk gaat verloren.

Deadlock -en starvation-vrij

Sommige transacties worden onnodig afgebroken.

14.2.1 Deadlock preventie zonder timestamps

- "No waiting" schema: Transactie onmiddellijk afbreken indien deze een grendel niet kan krijgen.
- "Cautious waiting" schema: Transactie mag enkel wachten op een grendel indien de transactie die deze grendel heeft zelf niet aan het wachten is. (Is deadlockvrij)
- Timeouts: Transactie te lang aan het wachten: afbreken en herstarten.

14.2.2 Deadlock detectie

Detectie op basis van de "wacht op" graaf. Lus in graaf = deadlock.

Indien deadlock: Kies een slachtoffer om af te breken. Maar: Starvation.

Oplossingen voor starvation:

- Eerlijke toekenning van grendels (first come, first serve; verschillende prioriteiten)
- Eerlijke slachtoffersselectie.

14.2.3 Concurrentiecontrole zonder grendels

Serialiseerbaarheid garanderen d.m.v. tijdstempels.

Twee tijdstempels:

$read_TS(X)$: Tijdstempel van de jongste transactie die X heeft gelezen.

$write_TS(X)$: Hetzelfde voor schrijven.

Voorbeeld: T mag X niet lezen of schrijven als X door een transactie met latere tijdstempel geschreven is.

Indien tijdstempels niet kloppen: Transactie ongedaan maken en opnieuw aanbieden (met latere tijdstempel)
→ Cascading rollback.

Vermijdt deadlock, maar geen starvation. Het tijdstempelordeningsalgoritme is bovendien niet herstelbaar.

Strikte tijdstempelordering:

Er wordt gewacht met $read(X)$ of $write(X)$ totdat de transactie met timestamp $write_item(X)$ gecommit of afgebroken is.

14.3 Concurrentiecontrole - Multiversietechnieken

Als T een $write(X)$ mag uitvoeren: creatie van een nieuwe versie X_{k+1} met $read_TS(X_{k+1}), write_TS(X_{k+1}) = TS(T)$

Als T de waarde van versie X_i mag lezen:

$read_TS(X_i) = \max(read_TS(X_i), TS(T))$

14.4 Concurrentiecontrole - Optimistische concurrentiecontrole

Alle aanpassingen gebeuren in lokale kopies.

Valideringsfase: Indien aan serialiseerbaarheid voldaan: commit, anders herstart later opnieuw.

14.5 Concurrentiecontrole - Granulariteit van gegevensitems

Granulariteit = grootte van de gegevens

Databank = grove granulariteit, veld in een record = fijne granulariteit

Hoe fijner de granulariteit, hoe meer concurrentie mogelijk is.

14.6 Herstel - Begrippen

Immediate update: Wijzigingen kunnen aangebracht worden voor het bereiken van een commit point.

Deferred update: Wijzigingen pas aanbrengen na het bereiken van een commit point.

DBMS systemen gebruiken cache als buffer.

Dirty bit: Geeft aan of een pagina gewijzigd is.

Pin/unpin bit: Geeft aan of pagina teruggeschreven mag worden naar disk (pin zolang een transactie nog werk heeft).

Write-ahead logging: Eerst naar log schrijven, dan pas aanpassing doorvoeren.

Steal: Pagina naar schijf schrijven, ook al is deze vastgepind.

Force: Alle gewijzigde pagina's onmiddellijk naar schijf schrijven.

Checkpoint: Alle gewijzigde buffers worden naar schijf geschreven. Checkpoint wordt op de log geschreven.

Fuzzy checkpoint: Wanneer alles geschreven is op de log, transacties hervatten (vorige checkpoint blijft nog even bestaan).

14.7 Herstel - Hersteltechnieken [uitgestelde aanpassing]

Een transactie kan de gegevensbank niet wijzigen voor het bereiken van haar commit point.

Een transactie bereikt haar commit point vooraleer aanpassingsopdrachten zijn geregistreerd op de log.

NO-UNDO/REDO algoritme. (RDU_M)

Efficiëntere versie:

Enkel de laatste write van een item doorvoeren. De log achterstevoren doorlopen en REDO *write(X)* enkel indien eerder (verder op de log) nog geen *write(X)* tegengekomen is.

14.8 Herstel - Hersteltechnieken [onmiddellijke aanpassing]

Wijzigingen gebeuren in de database zelf.

Write-ahead log protocol.

REDO/UNDO. (RIU_S)

14.9 Herstel - Schaduwpaginering

Construeer een index voor de relevante delen van de database. Kopiëer deze paginatable naar de schijf (= schaduwindex).

Gedurende de transactie wordt de paginatable aangepast, maar de originele schaduwindex blijft ongewijzigd.

Bij een commit verdwijnt deze schaduwindex.