Samenvatting Databanken I

Thomas Abeel

10 januari 2004

Inhoudsopgave

Deel I

Preliminaries

Hoofdstuk 1

An overview of Database Management

1.1 Introduction

Een databasesysteem is een computersysteem om records bij te houden. Het doel hiervan is het centraal beheren en terbeschikking stellen van informatie.

De databank zelf is een elektronische opslagplaats voor een verzameling databestanden. De gebruiker kan hier operaties op uitvoeren (toevoegen, verwijderen, aanpassen, opzoeken) dmv SQL.

- Tabel: Een databestand uit een databank. (relation)
- Record: Een rij uit zo'n tabel. (tuple)
- Veld: Een kolom uit zo'n tabel. Heeft een datatype. (attribute)

1.2 What is a database system

Onderscheid tussen data en informatie: data is wat er eigenlijk is opgeslagen in de databank, terwijl informatie de betekenis is die deze data heeft.

Een databanksysteem bevat vier grote componenten: data, hardware, software en gebruikers.

1.2.1 Data

Het onderscheid tussen multi- en single-userdatabanken is dat bij een single-user er slechts 1 gebruiker tegelijk kan gebruik maken van de databank en bij multi-user zijn dit er meerdere. Dit onderscheid tussen multi-user en single-user is meestal irrelevant aangezien een multi-userdatabank er voor een gebruiker uitziet als een single-userdatabank. De nadelen/problemen zijn vnl. intern zodat de gebruiker er niets van ziet. Data zal meestal *integrated* en *shared* zijn.

- Integrated: Een databank is een samenvoeging van meerdere bestanden om redundantie te vermijden. Dit door overtollige/meervoudige data te integreren.

- Shared: Verschillende gebruikers kunnen bij de data. Er kunnen dus meerdere gebruikers dezelfde data tegelijkertijd gebruiken.

Een gevolg hiervan is dat een gebruiker meestal slechts geïnteresseerd is in een beperkt deel van de databank.

1.2.2 Hardware

Hardware bestaat uit opslagmedia, processors en geheugen. Dit valt buiten bestek van dit boek.

1.2.3 Software

Tussen de fysische opslag en de gebruikers zit het databasemanagementsystem (DBMS). Alle operaties die in 1.1 aangeven zijn, zullen door DBMS worden uitgevoerd. De belangrijkste functie van de DBMS is dus het beschermen van de gebruiker van de hardwaredetails. DBMS is de belangrijkste software, maar niet de enige, andere zijn: transaction manager, dev tools,...

1.2.4 Users

- Toepassingsprogrammeur: Programmeert in een 'hogere' programmeertaal een programma voor gebruikers. Dit programma maakt gebruik van SQL(meestal) om via de DBMS toegang te krijgen tot de databank. Applicaties kunnen online zijn zodat de gebruiker interactief data kan opvragen.
- Eindgebruiker: Gebruiker de eerder genoemde programma's of kan gebruik maken van de meegeleverde interface bij de databank. Deze ingebouwde interface's kunnen soms *menu-* of *form-driven* zijn.
- Database administrator: ook wel DBA, wordt uitgebreider besproken in 1.4

1.3 What is a database

1.3.1 Persistent data

Data in een databank is *persistent*. Dit wil zeggen dat eens de data geaccepteerd is door DBMS en dus in de databank zit, ze er pas weer uit is na een expliciete opdracht aan DBMS. De data zal niet verdwijnen als randeffect van een of ander proces.

Nieuwe definitie van een databank: Een databank is een verzameling van persistente data die gebruikt wordt door de applicatiesystemen van een gegeven onderneming.

Operational data wordt nu meestal in de context van online transaction processing (OLTP) gebruikt. Terwijl men in *decision support* spreekt van een datawarehouse i.p.v. een databank.

1.3.2 Entities en Relationships

- Entities: Objecten waarover je data wil bijhouden. Deze objecten hebben eigenschappen (properties).
- Relationships: Verbanden tussen de verschillende objecten. (binaire, ternaire,...)
- Connection trap: Drie binaire relaties zijn niet equivalent met 1 ternaire. (Jan-appels-Piet).
- Bill-of-materials: Er kunnen relaties zijn met betrekking tot 1 entiteit.

1.3.3 Properties

Entiteiten hebben properties die we naar believen kunnen invullen. De complexiteit speelt geen rol.

1.3.4 Data and Data Models

Data is een gegeven feit, dus een ware propositie. Dus een databank is een verzameling ware proposities.

SQL is gebaseerd op het *relational model of data*. Dat wil zeggen dat data voorgesteld is door rijen in tabellen. Deze rijen kunnen geïnterpreteerd worden als ware proposities. En dat er operators gegeven zijn die van de gegeven proposities nieuwe ware proposities kunnen afleiden.

Algemene betekenissen van een data model:

- Abstracte logische definitie van de objecten en operatoren die de abstracte machine vormen waarmee de gebruiker werkt. De objecten bepalen de structuur, de operatoren het gedrag. De implementatie (fysische realisatie) moet niet bekend zijn door gebruiker.
- Een toepassing van de eerste betekenis op een specifiek probleem.

1.4 Why database?

Voordelen van een databank over papier:

- Compact
- Snelheid
- Minder slavenarbeid
- Altijd up-to-date
- Bescherming

Een bijkomend voordeel voor een multi-userdatabank is de centrale controle van data die zo'n systeem biedt en toch een verspreide toegang.

1.4.1 Data Administration and Database Administration

Data administrator (DA) is verantwoordelijk voor de databank. D.w.z. welke data opgeslagen wordt, opstellen van onderhouds- en beveiligingspolicies. De databaseadministrator (DBA) is de technieker (of team) die de beslissingen van DA gaat implementeren.

1.4.2 Benefits of the Database Approach

- Minder redundantie. Door databestanden te integreren wordt redundantie geëlimineerd.
- Inconsistentie vermijden. Door redundantie te verwijderen, zullen er geen duplicate records zijn die elkaar zouden kunnen tegenspreken. Evt. kan er wel gecontroleerde redundantie zijn, maar zodanig dat alle duplicaten steeds samen geüpdate worden (propagating updates).
- Transactionele verwerking. Sommige logische werkeenheden bestaan uit meerdere databankbewerkingen. Deze bewerkingen moeten ofwel allemaal ofwel geen van allen uitgevoerd worden.
- Bewaar integriteit door middel van *integrity constraints*. Dit is het beperken van de mogelijke waarden die een property kan krijgen. Dit is zeer belangrijk aangezien de data shared is en een fout van 1 gebruiker aanleiding kan geven tot veel gebruikers die foute data gebruiken.
- Beveiliging door middel van *security constraints*. Men krijgt enkel toegang tot databank via geijkte kanalen. Toegang tot gevoelige informatie kan gelogd worden. Bepaalde gebruikers mogen maar een bepaald gedeelte van de databank gebruiken.
- Algemene best-performance. De DA kan beslissen wat de algemene noden zijn van de onderneming en de databank daar voor laten optimaliseren. Het kan zijn dat dit niet optimaal is voor individuele programma's.
- Standaarden kunnen opgelegd worden. Dit zorgt dat data gemakkelijker uitgewisseld kan worden.

1.5 Data Independence

We bedoelen hier alleen fysische data *independence*. We leggen data *independence* uit a.d.h.v. programmatuur die data*dependent* is. Dit wil zeggen dat het programma kennis heeft van de fysische representatie van de data en daar ook gebruik van maakt. Het is duidelijk dat er zo problemen ontstaan, omdat de fysieke representatie niet gewijzigd kan worden zonder ook de programmatuur aan te passen.

- Verschillende applicaties maken gebruik van dezelfde, data maar in een ander view.
- De DBMS moet de mogelijkheid hebben om fysieke representatie te veranderen en om evt. nieuwe data toe te voegen zonder dat alle software, die gebruik maakt van de databank, moet herschreven worden.

Data*independence* kan dus gedefinieerd worden als: de immuniteit van programma's voor een verandering in opslagstructuren of toegangstechniek.

- Stored field: kleinste data-eenheid. Een databank zal veel instanties van een type stored field bevatten.
- Stored record: verzameling van stored fields die in verband staan met elkaar.
- Stored file: is de verzameling van alle bestaande stored records van een bepaald type.

Mogelijke wijzigingen waar een DBMS immuun voor moet zijn:

- Voorstelling van numerieke gegevens. Men moet de interne arithmetische notatie kunnen wijzigen om bv. prestaties te verbeteren.
- Voorstelling van karaktergegevens. Men moet codering die men gebruikt om karakters op te slaan, kunnen wijzigen. (ASCII,Unicode,...)
- Eenheden voor numerieke data wijzigen.
- Data codering. Men zal niet altijd de volledige waarde opslaan, maar slechts een code.
- Data materialisatie. Ten eerste is er directe materialisatie. Een logisch veld wordt geconstrueerd a.d.h.v. een stored field. Ten tweede is er indirecte materialisatie, waarbij een virtueel veld geconstrueerd wordt a.d.h.v. enige berekening, evt. op meerdere stored fields. Deze data kan dus niet rechtstreeks aangepast worden door de gebruiker.
- Structuur van stored records kan wijzigen. Bestaande records kunnen samegevoegd of opgesplitst worden.
- Structuur van stored files. Indexen, opslagdevices kunnen wijzigen.

Het gevolg van de lijst is dat een databank moet kunnen groeien zonder dat bestaande applicaties aangepast dienen te worden.

1.6 Relational systems and others

- Niet-relationele systemen: inverted list, hierarchic system, network system,...
- Relationele systemen: databank a.d.h.v. tabellen. Operaties op tabellen leveren nieuwe tabellen op.
- Object en object/relationeel: part VI (volgend jaar?)
- multidimensioneel: volgend jaar?
- logische databanken: volgend jaar?

Hoofdstuk 2

Database system architecture

2.1 The three levels of architecture

De ansi/sparc architectuur is verdeeld in 3 lagen.

- interne laag: is de laag het dichtst bij de fysieke laag en houdt zich bezig met de interne opslag van de informatie. *(implementation level)*
- externe laag: is de laag het dichtst bij de gebruiker. Dit is hoe de data waargenomen wordt door de gebruiker. *(model level)*
- conceptuele laag: is een laag van indirectie tussen de twee voorgaande. *(model level)*
- mapping: zorgt voor de overeenkomsten tussen de lagen onderling. ttz er bestaat een mapping *intern/conceptual* en een mapping *conceptual/extern*.

2.2 The external level

Elke gebruiker heeft een taal tot zijn beschikking. Voor een applicatieprogrammeur is dit een conventionele taal (C++, Java, ...), voor een eindgebruiker is dit meestal een *query*taal. (SQL) Al deze talen hebben een *data sublanguage* (DSL), dit is een gedeelte van de *hostlanguage* die zich bezig houdt met databankbeheer. Deze DSL bevat twee onderdelen: een *data definition language* (DDL) en een *data manipulation language* (DML). De hosttaal en de DSL kunnen *tighly* ofwel *loosely* gekoppeld zijn. In tighly gekoppelde talen is er geen verschil te zien tussen DSL en de rest van de taal.

De view van een gebruiker noemt men een external view. Voor die gebruiker is die view de database. Een external view bestaat uit veel verschillende external records van verschillende types. Elke external view is gedefinieerd d.m.v. een *external* schema dat geschreven is in de external DDL.

2.3 The conceptual level

De conceptual view is een abstracte voorstelling van alle informatie in de databank. De conceptual view is een verzameling van conceptual records. De conceptual view is gedefinieerd d.m.v. een conceptual schema dat geschreven is in conceptual DDL. Als er data onafhankelijkheid moet zijn, mag dit schema geen gebruik maken van indices, hashings, pointers,... of eender welk ander opslag- of toegangsdetail. Als het conceptuele niveau dataonafhankelijk is, zal de volgende laag eveneens dataonafhankelijk zijn.

2.4 The internal level

De internal view is een representatie van de hele databank op een laag niveau. Het bestaat uit een verzameling internal records of ook stored records. We zijn dus nog steeds een niveau hoger dan het puur fysieke niveau, maar dat is afhankelijk van het systeem en dus niet aan de orde in de architectuur van een databank.

Het interne niveau is beschreven a.d.h.v. een internal schema dat gebruik maakt van internal DDL. Omvat ook indexen, fysiek volgorde,...

2.5 Mappings

- Conceptual/internal: definieert de overgang van conceptuele laag naar interne laag. Als er veranderingen gebeuren aan de opslagstructuur, moet deze mapping mee gewijzigd worden zodat het conceptuele schema onveranderd blijft. Dit is fysische dataonafhankelijkheid.
- External/conceptual: definieert de overgang tussen de gebruikerslaag en de conceptuele laag. Van dit soort kunnen er vele zijn, voor elke gebruiker een ander. Dit zorgt voor logische data onafhankelijkheid.

2.6 The database administator

De taken van de DBA in detail:

- Het conceptuele schema definiëren: ook wel *logical database design*. Welke informatie over welke entiteiten moet bijgehouden worden.
- Het interne schema definiëren: *physical database design*. Hoe de informatie uit stap 1 moet bijgehouden worden.
- Contact met gebruikers: De externe schema's definiëren voor de verschillende gebruikers. En ook technische ondersteuning.
- definiëren van *security* en *integrity constraints*
- definiëren van schema's voor het back-uppen en evt. herstellen van een databank.
- Performantie bewaken en evt. reageren op veranderingen in eisen d.m.v. *tuning* en reorganisaties van gegevens.

2.7 The database management system

De DBMS is de software die alle toegang tot een databank regelt:

- De gebruiker stuurt toegangsaanvraag in een querytaal.
- DBMS accepteert aanvraag en analyseert
- DBMS bekijkt: external schema van gebruiker,corresponderende mapping naar conceptual, conceptual schema, mapping naar internal en stored databankschema.
- DBMS voert de nodige opdrachten uit op de opgeslagen gegevens.

Een DBMS moet volgende functies hebben:

- Datadefinitie: De DBMS moet een DDL-processor of een DDL-compiler hebben. De DBMS moet ook DDL-definities begrijpen.
- Datamanipulatie: DBMS moet een DML-processor of compiler hebben. DML-aanvragen kunnen onderverdeeld worden in geplande (*operational applications*) en ongeplande(*decision support*). Geplande zijn meestal embedded in voorgeschreven programma's terwijl ongeplande vaak via een *query language processor* komen.
- Optimalisatie en uitvoering: de DML-aanvragen worden verwerkt door een *optimizer* om dan door een *runtime-manager* uitgevoerd te worden.
- Informatiebeveiliging en integriteit: DBMS moet beperkingen toestaan en kunnen uitvoeren.
- Informatie herstellen en overeenstemmen: zie later.
- Datadictionary: Dit is een databank over de databank en bevat alle metainformatie. Dus alle schema's, mapping, security-, integrity-constraints. Evt. ook extra informatie over welk programma/gebruiker, welk gedeelte van de databank gebruikt. Er moet ook een omschrijving van zichzelf gegeven worden.
- Performantie: alles moet zo efficiënt mogelijk gebeuren.

De DBMS dient dus eigenlijk om een user-interface te leveren voor de eigenlijke databank.

2.8 Client/server architecture

Men kan databasesystems bekijken als een server/client architectuur. Waarbij de gebruikers (en hun programma's) *clients* zijn en de DBMS de server. Het systeem kan dus in twee delen gesplitst worden. Als we die nu op verschillende machines, die met elkaar verbonden zijn door een communicatienetwerk, uitvoeren hebben we *distributed processing*.

2.9 Utilities

De DBMS producent zal vaak een aantal onderhoudsprogramma's meeleveren die rechtstreeks op het interne niveau werken. Dit is nodig voor opstarten van een nieuwe databank, back-uppen, reoganiseren, statisch performantie onderzoek, ...

2.10 Distributed processing

De voordelen van distributed processing zijn:

- Parallelle verwerking: waardoor de performantie toeneemt.
- Server kan een aangepaste machine zijn, geoptimaliseerd is voor het werken met een DBMS.
- De clientmachine kan een workstation zijn die zeer gebruiksvriendelijk is voor de gebruikers.
- Verschillende *clients* zullen tegelijkertijd toegang kunnen hebben tot de databank.

Naast het eerder vermelde client/server model, kan elke gebruiker ook een volledig databanksysteem op zijn machine hebben staan. Als deze verschillende databanken zich voordoen als 1 grote en dit op transparante wijze, dan spreken we van een *distributed database system*.

Hoofdstuk 3

An Introduction to Relational Databases

3.1 An informal look at the relational model

Het relationele model heeft drie aspecten:

- datastructuur: de databank wordt door de gebruikers enkel waargenomen als tabellen. Dit zorgt voor een logische structuur.
- dataintegriteit: a.d.h.v. van *constraints*
- datamanipulatie: restrict (een aantal rijen uit de tabel halen), project (een aantal kolommen uit een tabel halen) en join (tabellen samenvoegen a.d.h.v. een gemeenschappelijke kolom.)

Het relationele model voldoet aan de sluitingseigenschap(*closure*), aangezien het resultaat van elke operator op een tabel terug een tabel geeft. Ook gebeuren alle bewerkingen steeds op een volledige tabel (*set-at-a-time*). Het is dus mogelijk om geneste operaties te definiëren. Tussenresultaten moeten niet altijd volledig *materialized* zijn. Men kan ook tussenresultaten rechtstreeks doorverwijzen naar de volgende operatie(*pipelined evaluation*).

The Information Principle: De hele databankinhoud wordt op slechts 1 manier weergegeven. Nl. a.d.h.v. expliciete waarden in de kolommen van een rij. Dit wil zeggen dat er op logisch niveau geen pointers zijn.

De belangrijkste integriteitconstraints zijn de *primary key* en de *foreign keys* die naar primary keys in andere tabellen verwijzen om het logische verband tussen die tabellen weer te geven.

Een formelere definitie: Het relationele model bevat volgende vijf componenten:

- Een niet-gesloten verzameling van scalaire types (kleinste betekenisvolle eenheid). Dit impliceert dat je er zelf kan bij definiëren.
- Een relatie-type generator met de bijhorende interpretatie.
- Mogelijkheid om relationele variabelen te definiëren.

- Een relationele toewijzing om relationele waarden aan zo'n relationele variabele toe te kennen.
- Een niet-gesloten verzameling van relationele operatoren om van relationele variabelen nieuwe relationele variabelen af te leiden.

3.2 Relations and relvars

algemeen	**relationeel informeel**	**formeel**
file	tabel	relatie
record	rij	tuple
veld	kolom	attribute

Relatievariabele is een benoemd object waarvan de waarde kan veranderen. De relatiewaarde is de waarde die de relvar op dat moment heeft. (de eigenlijke tabelinhoud)

3.3 What relations mean

Kolommen hebben een geassocieerd data-type, dus alle data in die kolom is van hetzelfde type. Dit kan een *user-defined* type zijn.

Elke relatie (relatie-waarde) heeft twee onderdelen:

- Heading: een verzameling kolom-naam/data-type paren.
- Body: een verzameling rijen die voldoen aan de gegeven heading.

We kunnen nu heading zien als een predicaat met als parameters de kolommen. Elke rij in de body is nu een ware propositie die verkregen wordt door in het predicaat waarden in te vullen voor de parameters.

3.4 Optimization

Relationele talen zijn niet procuderaal. Ze doen aan *automatic navigation*. De gebruiker zegt wat hij wil, niet hoe hij het wil. De *optimizer* uit de DBMS kiest uit de mogelijke oplossingen de efficiëntste, rekening houdend met grootte van de tabellen, indices,...

3.5 The catalog

De cataloog geeft een overzicht van alle mappings, schema's. Ook bevat hij beschrijvende informatie over de databank, nl. een beschrijving van elke tabel (aantal kolommen/rijen, naam,...) en van elke kolom(naam, tabel waar ze toebehoort,datatype,...). Deze informatie is opgeslagen in systeemtabellen (*system relvars*) die net als andere relvars aangesproken kunnen worden.

3.6 Base relvars and views

De gegeven tabellen worden base-relvars genoemd, hun waarden base-relations. Tabellen die hiervan afgeleid zijn noemt men derived-relations. Meestal bestaat er ook een speciaal soort van afgeleide benoemde relvar, nl. een view. Deze worden ook wel derived-relvars genoemd. Deze views worden in de cataloog allen bewaard als view-defining expression. Er wordt geen kopie van de data bijgehouden. Bewerkingen op een view zijn direct zichtbaar in de basistabellen. Dit zorgt dus voor een logische informatieonafhankelijkheid.

Het verschil met ANSI/SPARC. Daar gebruikt men basis relvars als conceptueel niveau en views op het externe niveau.

3.7 Transactions

Een transactie is een logische eenheid werk. Dus moet voldoen aan volgende eigenschappen:

- *Atomic*: ofwel moet de hele transactie gebeuren, ofwel niets. Zelfs als het systeem halverwege crasht.
- Duurzaam: eens er een commit uitgevoerd is, moet de data persistent zijn.
- Geisoleerd: de tussentijdse operaties mogen niet zichtbaar zijn voor andere transacties.
- Serialiseerbaar: Als men twee transacties tegelijk uitvoerd, moet dit hetzelfde resultaat geven als wanneer men ze na elkaar zou uitvoeren.

Deel II

The Relational Model

Hoofdstuk 4

Types

4.1 Introduction

Een type is een verzameling van waarden, ook wel een domein genoemd. Een type is ofwel *system-defined* ofwel *user-defined*. Bij elk type horen een aantal *operators*, die zin hebben voor dat type. Het is bijvoorbeeld zinloos om datums te vermenigvuldigen,maar wel zinnig om ze te vgln.

4.2 Values vs. variables

Een waarde is een individuele constante zonder locatie in tijd of ruimte. Een waarde kan wel voorgesteld worden in het geheugen d.m.v. een codering. Deze voorstellingen hebben wel een plaats en tijd. Waarden kunnen per definitie niet ge-update worden, bij het updaten verandert de waarde en is het die waarde dus niet meer. Waarden kunnen arbitrair complex zijn.

Een variabele is een plaatshouder voor een voorstelling van een waarde. Variabelen kunnen geupdate worden door er een andere waarde in te steken.

Er is ook nog een verschil tussen de voorstelling van een waarde als model concept en de fysieke representatie wat een implementatie concept is.

4.2.1 Values and Variables are Typed

Elke waarde heeft een type en heeft dus een soort vlag om duidelijk te maken welk type het is. Alle waarden zijn van precies één type, dat nooit verandert.

Sommige operatoren kunnen werken met verschillende types van waarden, men noemt ze dan *polymorphic*.

4.3 Types vs. Representations

Types zijn onderwerp van het model, terwijl representaties onderwerp zijn van de implementatie. De fysische representatie zal altijd verborgen moeten zijn voor de gebruiker om dataonafhankelijkheid te garanderen. Operators zullen gedefinieerd zijn in termen van types, niet a.d.h.v. fysische representaties.

4.3.1 Scalar vs. nonscalar types

Een niet-scalair type is een type van dewelke de waarden expliciet een verzameling van direct toegankelijke componenten bevat die zichtbaar zijn voor de gebruiker.

Een scalair type is een type dat niet niet-scalair is. Scalaire types worden ook wel een *encapsulated* of *atomic* genoemd.

4.3.2 Possible Respresentations, Selectors and THE_ operators

Elk type moet minsten één mogelijke representatie hebben. Deze representatie is niet verborgen voor de gebruiker en heeft mogelijk componenten die eveneens zichtbaar zijn aan de gebruiker. Iedere mogelijk representatie heeft ook altijd twee operatoren:

- selector: Kan een waarde van het geselecteerde type maken door een waarde te verschaffen voor iedere component van de gekozen mogelijk representatie.
- THE_ operator: Laat de gebruiker toe om de overeenkomstige componenten van de mogelijke representatie te benaderen.

Zie ook voorbeelden Bp. 117

4.4 Type definition

Nieuwe types kunnen gemaakt worden door een expliciete *TYPE-statement* of door een type generator.

Type-statement a.d.h.v. een voorbeeld

```
TYPE WEIGHT POSSREP { D DECIMAL (5,1)
CONSTRAINT D > 0.0 AND D < 5000.0 };
```

Het voorgaande geeft een type-constraint voor WEIGHT. Als er geen expliciete constraint opgegeven wordt, wordt die standaard *waar* genomen.

4.5 Operators

- = kan voor alle types infix gebruikt worden
- <,> kunnen infix gebruikt worden als het om een ordinaal type gaat, d.w.z. dat > zinnig gedefinieerd kan worden.

4.5.1 Type Conversions

De selector kan gezien worden als een type-converter van de component types naar het type van het type. Er wordt gebruik gemaakt van *strong-typing*, t.t.z. alle waarden hebben een type, en alle functies nemen argumenten van een bepaald type, en geven een bepaald type terug.

4.5.2 Concluding remark

Ondersteuning voor operators impliceert het volgende:

- Het systeem weet welke expressies geldig zijn en wat het type is van het resultaat van elke expressie.
- Dit wil dus zeggen dat de totale verzameling van types een gesloten verzameling is, die zeker het type *boolean* bevat als men moet kunnen vergelijken.
- Als het systeem weet wat het type is van elke geldige expressie, dan weet het ook welke *assignments* en *comparisons* geldig zijn.

4.6 Type Generators

Het is een speciaal soort operator die een type teruggeeft in plaats van een gewone value. Als voorbeel een array.

Hoofdstuk 5

Relations

5.1 Tuples

Preciese definitie van een tuple (rij):

Gegeven een collectie types T_i (i=1,...,n) niet noodzakelijk verschillend. Een tuple waarde t is een verzameling van geordende tripletten $< A_i, T_i, v_i >$ met A_i een unieke attribuutnaam, T_i een typenaam en v_i een waarde van het type T_i.

- n is de graad van t
- het geordende triplet noemt men ook wel een component van t
- $< A_i, T_i >$ is een attribuut van t. De waarde v_i is de attribuutwaarde en T_i attribuuttype
- de volledige verzameling van attributen is heading van t
- tuple type is bepaald door de heading van t

Een voorbeeld:

MP:P#	**LP:P#**	**QTY:QTY**
P2	P4	7

De graad is hier 3

$$\begin{array}{cc} A_1\text{=MP} & T_1\text{=P\#} \\ A_2\text{=MP} & T_2\text{=P\#} \\ A_3\text{=QTY} & T_3\text{=QTY} \end{array}$$

5.1.1 Properties of Tuples

- Elke tuple heeft exact 1 waarde voor elk van zijn attributen.
- Er is geen ordening in de componenten van een tuple.
- Elke deelverzameling van een tuple is terug een tuple.

5.1.2 Operators on tuples

- Gelijkheidsoperator
- Selectoroperator
- Assignmentoperator
- Tuple join,project en restrict
- Extract, om de attribuutwaarden uit een tuple te halen

5.2 Relation Types

Preciese definitie van een relation (tabel).
Een relatie waarde r bestaat uit een heading en een body.

- Heading van r is de heading zoals gedefinieerd in 5.1.
- Body van r is een verzameling van tuples die allemaal dezelfde heading hebben,nl. die van r. De kardinaliteit van r is het aantal elementen in die verzameling.

Het relatie type is bepaald door de heading van r, en heeft dezelfde attributen en graad.

5.3 Relation Values

Eigenschappen van relaties:

- Relaties zijn genormaliseerd(1NF): Elke tuple bevat exact n waarde voor elk van zijn attributen. Alle relaties voldoen aan 1NF.
- Attributen zijn niet geordend: Dit komt omdat de attributen een verzameling vormen. Een verzameling ondersteunt geen ordening.
- Tuples zijn niet geordend: idem
- Er zijn geen duplicate tuples: Aangezien de body van een relatie een verzameling is, kan die geen twee identieke tuples bevatten.

5.3.1 Relation vs. Tables

Tabellen hebben een aantal gebreken om relaties voor te stellen:

- Typenamen staan niet in labels
- Kolommen zijn geordend, attributen zijn dat niet
- Rijen zijn geordend, tuples niet
- Tabel kan dubbele rijen bevatten, een relatie niet
- Tabellen moeten minstens 1 kolom hebben, relaties niet minsten 1 attribuut

- Tabellen mogen *null* bevatten, relaties niet
- Tabellen zijn 2-dimensionaal, relaties niet

Om een tabel dus als een relatie te interpreteren moeten er *rules of interpretation* zijn.

5.3.2 Relation-Valued Attributes

Aangezien relaties een type zijn en er voor een attribuut eender welk type kan gespecifieerd worden, kunnen attribuutwaarden dus relaties zijn.

5.3.3 Relations with No Attributes

Elke relatie heeft een verzameling attributen, aangezien de lege verzameling ook een verzameling is, bestaat er een relatie zonder attributen. Die relatie kan ten hoogste n tuple bevatten, nl. lege tuple. TABLE-DEE is de lege relatie met tuple, TABLE-DUM de lege relatie zonder tuple.

5.3.4 Operators on Relations

- Relationele vergelijkingen: Men kan twee relaties vgln. m.b.v. volgende operatoren $=, \neq, \subset, \subseteq, \supset, \supseteq$.
- *membership operator*
- Extractieoperator, om tuple uit relatie met n tuple te halen.

5.4 Relation Variables

2 soorten: base relvars en views.

5.4.1 Base Relvar Definition

- Heading, body, attributen, tuple, graad, ... zijn ook geldig voor relvars
- Alle mogelijke waarden van een relvar zijn van hetzelfde relationele type
- Candidate keys en foreign keys
- Het is mogelijk om voor de attributen default waarden op te geven bij creatie van de relvar.

5.4.2 Updating Relvars

Het relationele model bevat een relationele toekenningsoperator om waarden toe te kennen aan relvars. De relationele toekenningsoperator en afgeleiden (insert,delete,update) werken allemaal op verzamelingen (evt. met maar 1 element). Met updaten wordt bedoeld dat je de oude waarde vervangt door een nieuwe.

5.4.3 Relvars and Their interpretation

Een heading van een relvar kan gezien worden als een predicaat en alle tuples die voorkomen in de relvar kunnen gezien worden als ware proposities verkregen door argumenten in het predicaat in te vullen.

Closed world assumption: alleen en alle in de relvar opgenomen tuples zijn ware proposities, alle andere zijn vals.

Hoofdstuk 6

Relational Algebra

6.1 Introduction

Relationele algebra is een collectie operatoren die relaties als operanden hebben en relaties als resultaat geven. De eerste relationele algebra had 8 operatoren. De eerste vier zijn de klassieke operatoren op verzamelingen: *union*, *intersection*,*difference* en *Cartesian product*. Allemaal licht aangepast omdat hun operanden nu relaties i.p.v. verzamelingen zijn. En dan een groep van speciale relationele operatoren: *restrict*, *project*,*join* en *divide*.

Nuttige uitbreidingen van deze acht operatoren: *extend* en *summarize*.

6.2 Closure revisited

De output van eender welke relationele operatie is terug een relatie. Dit wil zeggen dat we geneste relationele expressies kunnen schrijven. We hebben dus *relation type inference rules* nodig om te bepalen wat, gegeven de inputtypes, de outputtypes zullen zijn. Om te zorgen dat er geen interferentie is, tussen de headers van de verschillende relaties bij een operaties, zullen we de operator *rename* definiëren.

6.3 The Original Algebra: Syntax

```
<relation exp>
  ::= RELATION {<tuple exp commalist>}
     | <relvar name>
     | <relation op inv>
     | <with exp>
     | (<relation exp>)

<relation op inv>
 ::= <project> | <non project>

<project>
 ::= <relation exp> { [ALL BUT] <attribute name commalist>}
```

```
<non project>
 ::= <rename> | <union> | <intersect> | <minus>
     | <times> | <where> | <join> | <divide>

<rename>
 ::= <relation exp> RENAME (<renaming commalist>)

<union>
 ::= <relation exp> UNION <relation exp>

<intersect>
 ::= <relation exp> INTERSECT <relation exp>

<minus>
 ::= <relation exp> MINUS <relation exp>

<times>
 ::= <relation exp> TIMES <relation exp>

<where>
 ::= <relation exp> WHERE <bool exp>

<join>
 ::= <relation exp> JOIN <relation exp>

<divide>
 ::= <relation exp> DIVIDEBY <relation exp> PER <per>

<per>
 ::= <relation exp> | (<relation exp>,<relation exp>)

De <relation exp> mag nooit een <non project> zijn met
uitzondering van INTERSECT waarbij ze evt. allebei,
of 1 van beide ook een INTERSECT mag zijn.

<with exp>
 ::= WITH <name intro commalist> : <exp>

<name intro>
 ::= <exp> AS <introduced name>
```

6.4 The Original Algebra: Semantics

Noot vooraf: De operandi moeten van hetzelfde type zijn.

6.4.1 Union

Union bevat alle tuples van a en b, maar elimineert duplicate tuples.

6.4.2 Intersect

A INTERSECT b is een relatie van hetzelfde type als a en b, met een body die bestaat uit alle tuples die in a en in b zitten. De inputrelaties moeten van hetzelfde type zijn.

6.4.3 Difference

A MINUS b is een relatie van hetzelfde type als a en b, met een body die bestaat uit alle tuples die in a zitten, maar niet in b. De inputrelaties moeten van hetzelfde type zijn.

6.4.4 Cartesian Product

A en b hebben geen gemeenschappelijke attributen. De heading is de unie van de heading van a en de heading van b. A TIMES b bevat alle tuples t zodat t een union is van een tuple uit a en één uit b.

6.4.5 Restrict

a WHERE X θ Y bevat alleen de tuples uit a die voldoen aan de θ-restrictie.

6.4.6 Project

a{X,Y,...,Z} is een relatie met enkel de vermelde attribute van a.

6.4.7 Join

Natural join

Relaties a en b hebben een aantal attributen gemeenschappelijk Y_i en een aantal attributen uniek, respectievelijk X_i en Z_i. De *natural join* is dan de relatie met de heading X,Y,Z en ale body de tuples waarvan de y-waarde in a overeenkomt met een y-waarde in b. Als a en b geen gemeenschappelijke attributen hebben dan krijg je a TIMES b.

θ-join

θ-join is gedefiniëerd als (a TIMES b) WHERE X θ Y. Als één van de twee attributen X of Y weg geprojecteerd wordt en de ander goed hernoemd, dan hebben we een *natural join*

Equijoin

Een equijoin is een θ-join waarbij θ = is.

6.4.8 Divide

a DIVIDEBY b PER c. Dan is a het deeltal, b de deler en c de bemiddelaar. Bestaat uit alle tuples {X x} uit a zodat {X x,Y y} in c zit voor alle tuples {Y y} in b.

6.5 What is Algebra for

De fundamentele intentie van relationele algebra is het schrijven avn relationele expressies. Deze expressies kunnen voor allerlei zaken gebruikt worden, waaronder dataopvraging. Een aantal mogelijkheden:

- Dataretrieval
- Updaten van data
- Integrity constraints
- Definiëren van *derived relvars*
- Definiëren van stabiliteitseisen
- Definiëren van *security constraints*

De algebra kan gebruikt worden als basis voor optimalisatie omdat relationele expressies *high-level*, symbolische representaties zijn. De algebra kan ook gebruikt worden als maatstaf om de kracht van een taal te meten.

6.6 Further Points

6.6.1 Associativity and Commutativity

Associatief en commutatief:

- UNION
- INTERSECT
- TIMES
- JOIN

MINUS voldoet aan geen van beide.

6.6.2 Some Equivalences

- r WHERE TRUE ≡ r
- r WHERE FALSE ≡ *empty*
- r { X, Y, ..., Z} ≡} r als X, Y, ..., Z alle attributen van r zijn.
- r { } ≡ TABLE_DUM als r=*empty*;anders TABLE_DEE
- r JOIN r ≡ r INTERSECT r ≡ r UNION r ≡ r
- r JOIN TABLE_DEE ≡ r
- r TIMES TABEL_DEE ≡ r
- r UNION *empty* ≡ r MINUS *empty* ≡ r
- *empty* INTERSECT r ≡ *empty* MINUS r ≡ *empty*

6.6.3 Some Generalizations

Stel s een verzameling van relaties.

- Als s maar 1 relatie(r) bevat, dan is de join, union en intersection van alle relaties van s gewoon r.
- Als s geen relaties bevat:
  - join is TABLE_DEE
  - union is lege relatie van het type s
  - intersection is de universele relatie van het type s

6.7 Additional Operators

6.7.1 Semijoin

De semijoin van a met b is de join van a en b geprojecteerd over a.

6.7.2 Semidifference

Gedefiniëerd als a MINUS (a SEMIJOIN b). Dus eigenlijk de tuples van a die geen equivalent hebben in b.

6.7.3 Extend

Geeft dezelfde relatie terug, maar met een extra attribuut dat bepaald wordt a.d.h.v. berekenbare expressie. EXTEND a ADD exp AS z.

Aggregate operators

Het doel is het afleiden van een enkele scalaire waarde van de waardes van een bepaald attribuut. Enkele typische voorbeelden: COUNT, SUM, AVG, MAX, MIN, ALL, ANY

6.7.4 Summarize

De extendoperator geeft een horizontale samenvatting per tuple, summarize geeft een verticale samenvatting per attribuut. SUMMARIZE a PER b ADD summary AS z geeft als resultaat een relatie met header {b,z}. Ook moet b een projectie van a zijn.

6.7.5 Tclose

Tclose staat voor transitieve sluiting. Zie hfst 24.

6.8 Grouping and Ungrouping

Men kan een aantal attributen van een relatie groeperen tot een aparte relatie en zo *relationvalued* attributen te krijgen. Omgekeerd kan je ook relaties ungroupen om hun attributen terug te krijgen.

Reversibel:

- Eerst group en dan ungroup is reversibel en geeft indentieke relatie terug.
- Eerst ungroup en dan group is soms omkeerbaar. Relatie r met relation-valued attribuut X is *reversibly ungroupable* a.s.a. r geen lege relaties als X-waarde heeft en er een candidte key bestaat die X niet als component heeft.

Hoofdstuk 7

Relational Calculus

7.1 Introduction

De relationele algebra levert het systeem een methode om een relatie te construeren a.d.h.v. gegevens een gegeven relatie (verzamelingenleer). Relationele calculus geeft enkel een methode om de gewenste relatie te definiïren in functie van de gegeven relaties. (predicaatlogica) Een fundamentele onderdeel is de *range variable*: een *range variable* is een variabele waarvan de toegestane waarden de tuples van een bepaalde relatie zijn. De originele relationele calculus is ook gekend als tuple calculus.

Een alternatieve versie, de *domain calculus*, bevat *range variables* die variëren over een domein (types) in plaats van over een relatie. Zie ook 7.7.

7.2 Tuple Calculus

7.2.1 Syntax

Zie boek p 215 e.v.

7.2.2 Range Variables

Range variabelen zijn geen variabelen zoals in een programmeertaal, het zijn logische variabelen. Ze staan specifiek voor tuples.

7.2.3 Free and Bound Variable References

Range variables komen in gebonden of vrije toestand voor in WFF's(well-formed formula). References naar range variables in een <*bool exp*> zijn in die expressie vrij a.s.a.:

- Als de boolexpressie onmiddelijk volgt op WHERE
- Een referentie naar dezelfde range variable verschijnt in het proto tuple direct voor de WHERE

Referenties naar range variabelen in het proto tuple zijn vrij.

Een WFF waar alle variabelen gebonden zijn noemt men gesloten, anders is zo'n WFF open.

7.2.4 Quantifiers

EXISTS is de existentiële kwantor, FORALL de universele.

7.2.5 Relational Operation

```
<relation op inv>        //relation operator invocation
 ::= <proto tuple> [WHERE <bool exp>]
```

7.3 Examples

Prenex normal form van een WFF is de vorm waar alle kwantoren vooraan staan. Iedere WFF kan in deze vorm gebracht worden.

7.4 Calculus vs. Algebra

Codd's reductie algoritme zet elke calculus-expressie om in een equivalente algebra-expressie. De uitkomst zal enkel de 8 originele operatoren van Codd bevatten.

- Bepaal voor elke rangevar de range, evt. al restricted als dat duidelijk is aan WHERE.
- Maak cartesisch product van de bekomen ranges.
- Restrictie van het product om de join condities in WHERE te vervullen. Dit is de equijion.
- Pas de kwantoren van links naar rechts toe:
  - FORALL rx: divideby r
  - EXISTS r: projecteer de attributen van r weg.

De calculus expressie is dus gelijk aan een geneste algebra-expressie. Dus de 8 operatoren van Codd vormen een goede *target language* om een calculus te implementeren. (er zal dan na reductie ook nog optimalisatie plaatsvinden)

Een taal is relationeel compleet als ze even krachtig is als de calculus. Een taal is *computational complete* als alle berekenbare functies berekend kunnen worden. Voor de algebra hebben we daarvoor EXTEND en SUMMARIZE.

7.5 Computational capabilities

De calculus bevat reeds analoge operatoren voor EXTEND en SUMMARIZE omdat:

- Een mogelijke vorm voor een prototuple is een <tuple selector inv> en deze kan arbitrair complex zijn.

- De comparanden in een boolexp kunnen arbitrair complexe expressies zijn.
- Het eerste argument van een <agg op inv> moet een relatie-expressie zijn.

7.6 Domain Calculus

Verschil met tuple calculus is dat range vars hier over domeinen (types) gaan ipv over relaties. De domein calculus ondersteunt ook een extra operator: de *membership condition*

7.7 Query-By-Example

Dit is een voorbeel van een taal gebaseerd op domein calculus, hoewel QBE ook delen van tuple calculus bevat. De syntax is intuïef, eenvoudig en is gebaseerd op het idee vaan aantekeningen maken op lege tabellen. Het is eenvoudiger dan tuple calculus omdat er geen kwantoren nodig zijn.

Hoofdstuk 8

Integrity

8.1 Introduction

Integrity: accuraatheid of correctheid van de data in een databank, waardoor dat die een weerspiegeling van de realiteit blijft. Deze integriteit wordt afgedwongen door constraints. Constraints hebben ook een semantisch aspect, ze zorgen er nl. voor dat data een betekenis krijgt.

8.2 A Closer Look

Integriteits beperkingen zijn in het algemeen beperkingen voor de waarden van een variabele of voor de combinatie van een aantal variabelen. Het feit dat een variabele van een gegeven type is, is op zich dus al een beperking. Nu bestaan er nog andere soorten beperkingen dan deze *a priori* beperkingen. Men kan al deze beperkingen in een vorm gieten: **ALS***een bepaald tuple in een bepaalde relvar zit* **DAN** *voldoet dat tuple aan een bepaalde voorwaarde.* Dit is een logische implicatie, waarbij het deel voor de dan het antecedent noemt en het deel erna het consequent.

8.3 Predicates and Propositions

Relationele expressies zijn predicaten en hun variabelen zijn parameters. Als we een predicaat willen testen, geven we als argument de huidige waarden van de variabelen. We krijgen dus een booleaanse expressie die geen variabelen bevat, een propositie dus. Een propositie is ofwel waar ofwel vals.

8.4 Relvar Predicates and Database Predicates

Het relvar predicaat voor R is de logische conjunctie van alle constraints die R vermelden. Elk van die constraints is op zich ook een constraint. Het database predicaat is de conjunctie van alle relvar predicaten.

The Golden Rule: Geen enkele update operatie mag ooit aan de relvar een waarde geven waardoor het relvarpredicaat vals zou worden.

The Golden Rule (tweede versie): Geen enkele update operatie mag ooit aan de database een waarde geven waardoor het databasepredicaat vals zou worden.

8.5 Checking the Constraints

De implementatie van een databank moet van de formele constraints de juiste controles afleiden die uitgevoerd moeten worden vóór een tuple ge-insert wordt. Als een databank orthogonaal gemaakt is, zal het tuple maar voor één relvar geschikt zijn en dus maar gecontroleerd worden door 1 relvar predicaat.

The Golden Rule verplicht dat integrity constraints voldaan zijn aan de grenzen van statements. Een eenheid van integriteit is dus de statement.

8.6 Internal vs. External Predicates

Predicaten die begrepen, uitgevoerd worden door het systeem en die formeel gedefiniëerd zijn worden interne predicaten genoemd. Er bestaan nl. ook externe predicaten. Interne predicaten zeggen wat de data betekend voor het systeem, terwijl de externe predicaten vertellen wat de data voor de gebruiker betekend. Een gegeven intern predicaat is de benadering van het systeem voor een bepaald extern predicaat. Interne predicaten voldoen niet aan de Close World Assumption, terwijl externe predicaten hier wel aan voldoen.

8.7 Correctness vs. Consistency

Een extern predicaat is de bedoelde interpretatie voor een relvar. Het intern predicaat geeft een criterium voor het accepteren van een update van een relvar.

8.7.1 The system cannot enforce truth, only consistency

Het systeem kan er niet voor zorgen dat er alleen ware uitspraken in de databank staan, het kan er alleen voor zorgen dat er geen tuples in staan die een integrity constraint overtreden.

$$Correct \Rightarrow consistent$$

$$Inconsistent \Rightarrow incorrect$$

8.8 Integrity and Views

Alle voorgaande paragrafen over integriteit gaan over relvars in het algemeen dus over views in het bijzonder.

8.9 A Constraint Classification Scheme

Er zijn 4 categoriën van constraints:

- Databaseconstraint

- Relvarconstraint
- Attributeconstraint
- Typeconstraint

8.9.1 Type Constraints

Een type constraint is de specificatie van de waarden die een bepaald type kan hebben.

8.9.2 Attribute Constraints

Worden meestal gewoon constraints genoemd. Is gewoon een declaratie dat een bepaald attribuut in een bepaalde relvar van een bepaald type is.

8.9.3 Relvar and Database Constraints

Relvarconstraints gaan over 1 relvar, terwijl databaseconstraints over 2 of meer relvars gaan. Databaseconstraints kunnen overgangsconstraints zijn. Een transitieconstraint kan aanduiden of de waarde van een relvar/database kan overgang van de ene naar de andere waarde.

8.10 Keys

8.10.1 Candidate Keys (CK)

Stel K is een verzameling vn attributen van relvar R, dan is K een *candidate key* van R a.s.a. K voldoet aan volgende twee voorwaarden:

- Uniciteit: geen enkele legale waarde van R bevat twee verschillende tuples met verschillende waarde voor K.
- Irreducibel: Geen enkel echte deelverzameling van K heeft de uniciteitseigenschap.

Elke relvar heeft minstens 1 *candidate key*, maar er kunnen er meerdere zijn. Candidate keys kunnen enkelvoudig of meervoudig zijn. Ze zorgen voor een basis tuple-niveau addresseringsmechanisme.

- Alle relvars, waaronder views, hebben candidate keys. Dus systeem moet een manier hebben om candidate keys af te leiden.
- Een *superset* van een candidate key is een *superkey*(SK). Een superkey is uniek, maar niet noodzakelijk irreducibel.
- Als SK een superkey is voor relvar R en A is een attribuut van R, dan moet de functionele afhankelijkheid SK $\rightarrow$ A zeker gelden. A $\rightarrow$ B: B is functioneel afhankelijk van A $\Leftrightarrow$ met elke A-waarde correspondeerd juist één B-waarde.

8.10.2 Primary Keys (PK) and Alternate Keys

Het is mogelijk dat een bepaalde relvar meerdere kandidaatsleutels heeft, dan kan je er één als primaire sleutel te kiezen en de andere worden dan alternatieve sleutels.

8.10.3 Foreign Keys (FK)

Een *foreign key* is een verzameling attributen van relvar R1 waarvan de waarden moeten gelijk zijn aan de waarden van een kandidaatsleutel van R2. Een foreign key kan, evenals een kandidaatsleuten, enkel- of meervoudig zijn. Een FK stelt een verwijzing voor naar het tuple met de overeenkomstige CK waarde. De constraint dat de waarden van FK en die van CK gelijk moeten zijn is de *referencial constraint*. *Referencial integrity* impliceert dat er voor elke FK waarde een gelijk CK waarde moet bestaan. Als B naar A wijst, moet A bestaan.

Een bepaalde relvar kan zowel verwijzen als verwezen worden. R3 → R2 → R1. Het pad van R3 naar R1 wordt ook *referencial path* genoemd. Relvars kunnen ook naar zichzelf verwijzen *self-referencing*. Er kunnen ook *referencial cycles* gevormd worden. Rn → ... → R2 → R1 → Rn

8.10.4 Referential action

Als er een update gebeurd aan een relvar(i.h.b. DELETE) kan het zijn dat er compensatie handelingen moeten gebeuren om ervoor te zorgen dat er geen referencial constraints overtreden worden. Twee veelgebruikte acties zijn CASCADE en RESTRICT. In het geval van CASCADE worden, als er een CK verwijderd wordt, alle FK die aan die CK gelijk zijn ook gedelete. In het geval van RESTRICT kan je enkel een CK verwijderen waarvoor er geen FK's meer bestaan.

Hoofdstuk 9

Views

9.1 Introduction

Views zijn afgeleide virtuele relaties, waarvan de naam en de view-definiërende expressie in de cataloog van de databank opgeslagen worden.

Queries op views kunnen op verschilllende manieren opgelosd worden. Eén ervan is de substitutiemethode, waarbij in elke expressie waarin de view voorkomt de naam van de view vervangen wordt door de view-definiërende expressie. Dit werkt wegens de sluitingeigenschap van de relationele algebra.

Men kan een view ook verwijderen nadat het gedefiniëerd is, als er geen enkel ander view gebruik maakt van de view in kwestie.

9.2 What are views for

De verschillende redenen voor het ondersteunen van views:

- Views zorgen voor een kortere notatie ofwel mogelijkheid tot gebruik van *macro's*
- Views laten toe dat verschillende gebruikers dezelfde data op hetzelfde moment anders waarnemen.
- Views zorgen automatisch voor de beveiliging van niet zichtbare items.
- Views kunnen voor logische informatieonafhankelijkheid zorgen.

9.2.1 Logical Data Independence

Logische informatieonafhankelijkheid kan gezien worden als de immuniteit van gebruikers voor het veranderen van de logische representatie van de data. Er zijn tweee aspecten aan logische dataonafhankelijkheid:

- Groei: als een databank groeit kan het zijn dat er nieuwe soorten data in opgeslagen worden. Door middel van views kan men ervoor zorgen dat oude gebruikers enkel de voor hun vertrouwde gegevens te zien krijgen.

- Herstructurering: soms zal het nodig zijn om de data anders logisch te ordenen, ttz in andere/verschillende relvars. M.b.v. views kan het systeem er toch voor zorgen dat oude users nog steeds de data aangeboden krijgen op een voor hen vertrouwde manier. Opgelet, het kan zijn dat er geen ondubbelzinnige mapping bestaat van de nieuwe naar de oude versie, waardoor informatieonafhankelijkheid onmogelijk is.

9.2.2 Two Important Principles

Views hebben twee nogal verschillende doelen:

- Gebruikers die views definiëren zijn zich bewust van de view-defining expressie en kunnen die als macro gebruiken.
- Gebruikers die enkel geinformeerd zijn dat een bepaalde view bestaat, kunnen die view enkel beschouwen als een base relvar omdat ze niet beter weten.

Hieruit volgt dat er geen arbitrair en onnodig onderscheid gemaakt moet worden tussen base- en derived relvars. Dit is *The Principle of Interchangeability*. Vanuit het standpunt van de gebruiker zijn alle relvars (base en derived) even belangrijk. En dus de keuze welke relvars echt zijn en welke niet is arbitrair. Dit is *The Principle of Database Relativity*

9.3 View Retrievals

Het resultaat van een retrieval op een view is hetzelfde als de retrieval doen op de materialisatie van de view-defining expressie. In praktijk is het meestal handiger om het substitutie principe te gebruiken. Retrievals kunnen gebruik maken van materialisaties, maar updates niet, omdat die direct moeten toegepast worden op de originele relvars.

9.4 View Updates

Views zijn per definitie, net als alle andere relvars, updateble. Men kan gebruik maken van dezelfde substitutie methode, maar er treedt een probleem op: views zijn virtueel zodat opgezocht moet worden welke onderliggende relaties aangepast moeten worden.

9.4.1 The Golden Rule Revisited

Golden Rule mag niet geschonden worden: Geen enkele update operatie mag ooit een waarde toekennen aan een relvar zodat het predicaat van die relvar zou evalueren tot vals.

9.4.2 Towards a View-updating Mechanism

Een aantal principes die in orde moeten zijn voor een systematische aanpak van het update-probleem:

- De mogelijkheid om views te updaten is een semantisch probleem, geen syntactisch.
- View update moet correct werken in het geval dat een view een baserelvar is.
- Update regels moeten symmetrie behouden waar dat van toepassing is.
- Update regels moeten de verschillende triggered actions in rekening nemen (cascade delete)
- Update kan gezien worden als DELETE-INSERT, zonder controle van integriteit na de delete.
- INSERT moet naar INSERT gemapt worden, DELETE naar DELETE.
- Regels moeten recursief toepasbaar zijn.
- Regels mogen niet veronderstellen dat de databank *well-designed* is.
- INSERT en DELETE moeten elkaars tegengestelde zijn.

9.4.3 Union

V = A UNION B

- Insert T: T wordt toegevoegd aan A, B of aan A en B al naar gelang T voldoet aan PA, PB, PA $\wedge$ PB.
- Delete T: verwijder T uit A en B indien dit mogelijk is. Dit kan evt. aanleiding geven tot cascade.
- Update T: delete T (zonder cascade) en voeg nieuwe T toe zoals bij insert. Dit kan tot gevolg hebben dat T migreert.

9.4.4 Intersect

V = A INTERSECT B

- insert T: als T voldoet aan PA $\wedge$ PB dan toevoegen aan A en B.
- delete T: T uit A en B verwijderen
- update: T moet voldoen aan PA $\wedge$ PB.

9.4.5 Difference

V = A MINUS B
T moet voldoen aan PA en niet aan PB

9.4.6 Restrict

V = A WHERE p
T moet voldoen aan PA $\wedge$ p

9.4.7 Project

V = A{X}
$\exists$ y $\in$ Y zodat {X:x,Y:y} voldoet aan PA

- insert T: T wordt aangevuld met default voor y en deze aangevulde versie wordt in A toegevoegd.
- delete T: alle waarden van A met de overeenkomstige X worden gewist.
- update T: verwijder alle overeenkomstige en steek er dan een nieuwe in met default voor y.

9.4.8 Join

Het eerste deel van het predicaat moet voldoen aan PA en het tweede deel aan PB.

- insert T: voeg a-deel aan A toe en b-deel aan B
- delete T: verwijder a-deel uit A en het b-deel uit B
- update T: verwijder oude T en steek nieuwe in de plaats

9.4.9 Extend

V= EXTEND A ADD <expr> AS X

- insert T: Als T voldoet aan PA dan wordt T toegevoegd zonder extra gedeelte.
- delete T: T wordt verwijderd uit A
- update T: verwijder, voeg toe

Deel III

Database Design

Hoofdstuk 10

Functional Dependencies (FD)

10.1 Introduction

Een *functional dependece* is een veel-één relatie van de ene verzameling van attributen naar een andere verzameling van attributen van een relvar.

10.2 Basic Definitions

Functionele afhankelijkheid: Laat R een relvar zijn en X en Y arbitraire deelverzamelingen van de verzameling van attributen van R. Dan zeggen we dat Y functioneel afhankelijk is van X. In symbolen: $X \rightarrow Y$. Enkel en alleen als in elke mogelijke geldige combinatie van R, elke X waarde juist 1 Y-waarde geassocieerd heeft. Of ook: als twee tuples hun X waarde gelijk hebben, dan hebben ze ook hun Y waarde gemeen.

Als relvar R voldoet aan de FD $A \rightarrow B$ en A is geen kandidaatsleuten, dan is er zekere redundantie in R.

10.3 Trivial and Nontrivial Dependencies

Een FD is triviaal als het onmogelijk is dat ze faalt. Een FD is triviaal a.s.a. de rechtse kant van de pijl een deelverzameling is van de linkse kant.

10.4 Closure of a Set of Dependencies

De verzameling van alle FDs die afgeleid kunnen worden van een een gegeven verzameling S van FDs noemen we de sluiting van S en noteren we als S^+.

Afleidingsregels:

- Reflexiviteit: B een deelverzameling van A dan $A \rightarrow B$
- Vermeerdering: $A \rightarrow B \Rightarrow AC \rightarrow BC$
- Transitiviteit: $A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C$

Dit zijn alle nodige en voldoende regels, ze geven de exacte sluiting van S. Er kunnen nog bijkomende regels gedefiniëerd worden:

- Decompositie: $A \rightarrow BC \Rightarrow A \rightarrow B \wedge A \rightarrow C$
- Unie: $A \rightarrow B \wedge A \rightarrow C \Rightarrow A \rightarrow BC$
- Compositie: $A \rightarrow B \wedge C \rightarrow D \Rightarrow AC \rightarrow BD$

10.5 Closure of a Set of Attributes

Het is mogelijk om de deelverzameling van de sluiting van S met een bepaalde linkerkant te berekenen.

10.6 Irreducible Sets of Dependencies

Stel S1 en S2 twee verzamelingen van FDs. Als elke FD die volgt uit S1 ook volgt uit S2 dan is S2 een *cover* voor S1. Als S1 een cover is voor S2 en S2 één voor S1 dan zijn S1 en S2 equivalent.

We definiëren een onreduceerbare verzameling van FDs als volgt:

- De rechtste kant van de pijl(*dependant*) heeft maar 1 atrribuut
- De linkse kant van de pijl (*determinant*) is op zijn beurt niet reduceerbaar. Dit wil zeggen dat geen enkel attribuut weg mag zonder de sluiting S^+ te wijzigen.
- Geen enkele FD in S kan weg zonder de sluiting S^+ te wijzigen.

Hoofdstuk 11

Further Normalization I: 1NF, 2NF, 3NF, BCNF

11.1 Introduction

Normalisatie is rond het concept van normale vormen gemaakt. Door het normalisatieproces is het mogelijk om een relvar in een hogere vorm te brengen. Deze procedure is omkeerbaar wat betekend dat ze verliesloos of gegevensbehoudend is.

11.2 Nonloss Decomposition and Functional Dependencies

We zijn enkele geïntereseerd in decomposities die verliesloos zijn, omdat het process anders niet omkeerbaar is. Verliesloos is heel nauw verbonden met het concept FD. De decompositie operator is een *project* terwijl de recompositie operator een *join* is.

Healths theorema: Laat R{A,B,C} een relvar zijn waar A,B en C verzamelingen van attributen zijn. Als R voldoet aan FD $A \rightarrow B$ dan is R gelijk aan de join van zijn projecties op {A,B} en {A,C}

11.2.1 More on Functional Dependencies

Enkele opmerkigen i.v.m. FDs:

- Niet reduceerbaar: een FD is links-onreduceerbaar als zijn linkse kant 'niet te groot' is
- FD diagrammen: Er zullen altijd pijlen zijn vanuit de kandidaatsleutels. Het is pas als er uit andere attributen pijlen vertrekken dat er problemen onstaan.
- FDs zijn een semantisch begrip, 't zijn een speciaal soort van integrity constraints.

11.3 First,Second and Third Normal Forms

Definitie van 1NF: Een relvar is in 1NF a.s.a. er in elke toegelaten waarde van die relvar elk tuple exact 1 waarde heeft voor elk attribuut.

Dit wil dus zeggen dat elke relvar in 1NF is. Relvars die enkel voldoen aan 1NF zullen een aantal *update-anomalies* vertonen.

Definitie van 2NF: Een relvar is in 2NF a.s.a. het in 1NF is en als elk niet-sleutel attribuut onreduceerbaar afhangt van de primaire sleutel.

Definitie van 3NF: *één kandidaatsleutel die we als primaire sleutel nemen* Een relvar is in 3NF a.s.a. het in 2NF is en elk niet-sleutel attribuut is niet-transitief afhankelijk van de primaire sleutel.

Deze laatste definitie gaat potentieel problemen geven als er meer kandidaatsleutels zijn, kandidaatsleutels samengesteld zijn of elkaar overlappen.

11.4 Dependency Preservation

Het is mogelijk dat je een gegeven relvar op verschillende manieren verliesloos kan uiteenhalen. Dan moet je de decompositie zo kiezen dat de verkregen relvars onderling onafhankelijk zijn. Dit wil zeggen dat je de ene mag aanpassen, zonder op de andere te moeten letten. Dit is bekend als *dependency preservation*

11.5 Boyce/Codd Normal Form

Deze vorm lost het probleem dat ontstond in 3NF m.b.t. meerdere, samengestelde en overlappende kandidaatsleutels op.

Definitie: Een relvar is in BCNF a.s.a. elke niet triviale,links-onreduceerbare FD een kandidaatsleutel heeft als determinant (linkse kant van de pijl).

Ander gezegd, de enige pijlen in FD diagram vertrekken uit kandidaatsleutels.

11.6 A note on Relation-valued Attribute

Het is technisch gezien mogelijk om relationvalued relvars wordt dit meestal gemeden om dat het toch maar de boel compliceerd.

Hoofdstuk 12

Further Normalization II: Higher Normal Forms

12.1 Multi-Valued Dependencies and Fourth Normal Form

Multi-valued dependencies zijn een veralgemening van de FDs uit het vorige hoofdstuk. Dit wil zeggen, elke FD is een MVD, maar omgekeerd niet.

Formele definitie: Stel R een relvar met A, B, C deelverzamelingen van de attributen van R, dan zeggen we dat B meervoudig afhankelijk is van A ($A \twoheadrightarrow B$) a.s.a. in elke toegelaten waarde van R de verzameling van waarden van B die overeenkomen met een bepaald AC paar enkel afhankelijk zijn van A en onafhankelijk van B.

4NF: Relvar R is in 4NF a.s.a. elke keer er deelverzamelingen A en B bestaan van de attributen van R zodat de niet triviale MVD $A \twoheadrightarrow B$ voldaan is dan moeten alle attributen van R functioneel afhankelijk zijn van A. De MVD $A \twoheadrightarrow B$ is triviaal als B een deelverzameling is van A of de unie AB van B en A is de volledige hoofding.

Anders gezegd: Dus de enige niet triviale afhankelijkheden (FD of MVD) in R zijn van de vorm K→X met K een superkey en X een ander attribuut.

Of ook: R is in 4NF a.s.a. R in BCNF is en alle MVDs in R zijn eigenlijk 'FDs' vanuit sleutels.

12.2 Join Dependencies and Fifth Normal Form

Tot hier toe ging het altijd over verliesloze decompositie in twee nieuwe relvars, maar er zijn relvars die niet verliesloos in 2 relvars gesplitst kunnen worden, maar wel in meer.

Definitie *join dependency*: Stel R een relvar met A,B,...,Z deelverzamelingen van attributen van R. We zeggen dat R voldoen aan de JD $\star$ {A,B,...,Z} a.s.a. elke legale waarde van R gelijk is aan de join van zijn projecties over A,B,...,Z.

Het is duidelijk dat JDs de meest algemene vorm van dependencies zijn. Dus we hebben een hoogste vorm van NF die alleen gebaseerd is op dependencies.

5NF: Een relvar R is in 5NF (projectie-join NF) a.s.a. voor elke niet triviale JD die voldaan is door R deze JD volgt uit de kandidaatsleutels van R. Waarbij een JD triviaal is als minstens één van de A,B,... alle attributen van R bevat. En waarbij een JD uit een kandidaat sleutel volgt a.s.a. elke van de A,B,... supersleutels voor R zijn.

12.3 The Normalization Procedure Summarized

- Neem projecties van de originele 1NF relvar om reduceerbare FDs te elimineren. Deze stap levert een 2NF relvar op.
- Neem projecties van de 2NF relvar om transitieve FDs te elimineren. $\Rightarrow$ 3NF
- Neem projecties om overige FDs te elimineren waarvan de determinant geen kandidaatsleutel is. Dit resulteert in een BCNF relvar.
- Neem projecties om alle MVDs te elimineren die geen FDs zijn. $\Rightarrow$ 4NF.
- Neem projecties om JDs die niet volgen uit kandidaatsleutels. $\Rightarrow$ 5NF (als je JDs kunt vinden)

Parallellisme bij BCNF, 4NF en 5NF:

- Een relvar R is in BCNF $\Leftrightarrow$ elke FD die voldaan is door R volgt uit de kandidaatsleutels van R.
- Een relvar R is in 4NF $\Leftrightarrow$ elke MVD die voldaan is door R volgt uit de kandidaatsleutels van R.
- Een relvar R is in 5NF $\Leftrightarrow$ elke JD die voldaan is door R volgt uit de kandidaatsleutels van R.

Doel van normalisatie:

- Elimineren van bepaalde soorten redundantie.
- Het vermijden van bepaalde update eigenaardigheden
- Om een goede representatie van de 'echte' wereld te leveren, één die intuïtief eenvoudig te snappen is.
- Om bepaalde integriteits beperkingen eenvoudig toe te passen, sommige *constraints* volgen uit andere. Dus als we de eerste in de ketting toepassen, zijn alle andere automatisch voldaan.

12.4 A Note on Denormalization

Er zijn zogezegd redenen om niet tot 5NF gaan om een toepassing optimaler te laten werken. Dit zijn drogredenen omdat denormalisatie op het niveau van bestanden zou moeten gebeuren, niet op het niveau van de relvars. Spijtig genoeg ondersteunen huidige producten dit onvoldoende.

Denormalisatie is dus het creëren van redundantie, maar eens we beginnen is het moeilijk te zeggen waar je moet stoppen.

Trouwens 'optimalisatie' door denormalisatie is slechts goed voor een beperkt aantal applicaties terwijl 5NF goed zou zijn voor anderen. Over het algemeen is het dus beter om zo vermogelijk te normaliseren en daar te blijven.

12.5 Other Normal Forms

De theorie tot hiertoe steunde op de *dependency theory*. Enkele ander manieren om NFs te maken:

- *Domain-key NF*: Een relvar R is in DKNF $\Leftrightarrow$ elke constraint op R een logisch gevolg is van de *domain-* en *keyconstraints* die van toepassing zijn op R. Een domain constraint is een verzameling mogelijke waarden voor een attribuut. Key constraints geven aan welke verzamelingen van attributen kandidaatsleutels zijn.
- *Restriction-union NF*: In plaats van project en join te gebruiken als decompositie maakt men hier gebruik van restrictie en union als decompositie operatoren.
- Sixth normal form: Het is mogelijk om generiekere versies van project en join te definiëren, waardoor er dus een generiekere vorm voor dependencies onstaat, waaruit men een nieuwe NF kan afleiden.

Deel IV

Transaction Management

Hoofdstuk 13

Recovery

13.1 Introduction

Met herstellen wordt meestal het herstellen van de databank zelf bedoeld, dit is de databank terug in een correcte staat brengen nadat een probleem is opgetreden. Men maakt gebruik van redundantie om het herstellen mogelijk te maken,redundantie op het fysieke niveau, niet op het logische.

13.2 Transactions

Een transactie is een logische werkeenheid. Dit wil zeggen dat alle opdrachten die in een transactie zitten ofwel allemaal ofwel geen van allemaal uitgevoerd moeten worden. Het is onmogelijk om alle opdrachten gegarandeerd op hetzelfde moment correct uit te voeren. Maar *transaction management* zorgt voor een vergelijkbare dienst: opdrachten worden één voor één uitgevoerd, maar als er iets misgaat worden de reeds uitgevoerde opdrachten terug ongedaan gemaakt. Ook zullen de uitgevoerde opdrachten pas zichtbaar worden voor de buitenwereld nadat de laatste opdracht succesvol afgerond is. Het onderdeel van het systeem dat hier voor zorgt is de *transaction manager*. De COMMIT en ROLLBACK opdrachten zijn de sleutel voor een goed functioneren.

Enkele belangrijke opmerkingen:

- Impliciete ROLLBACK: telkens als het systeem een fout tegenkomt in een test zal er een expliciete ROLLBACK volgen maar het systeem kan niet overal op testen. Dus als het systeem niet eindigt zoals gepland (expliciete ROLLBACK of COMMIT) moet er ook een ROLLBACK gebeuren.
- Bericht behandeling: Een typische transactie zal niet alleen proberen de databank aan te passen, maar ook een bericht teruggeven waarin staat wat er juist gebeurd is.
- Recovery log: Het systeem moet een log of dagboek bijhouden met alle details van alle updates. Als het dan nodig mocht zijn om een bepaalde actie ongedaan te maken, kan dat aan de hand van het logboek.
- Statement atomicity: Het systeem moet ervoor zorgen dat statements in één keer uitgevoerd worden d.w.z. dat ze atomair moeten zijn. Als er

in het midden van een statement een fout optreedt, mag er nog niets gewijzigd zijn aan de databank.

- Het uitvoeren van een programma is het sequentieel uitvoeren van een aantal transacties. COMMIT en ROLLBACK eindigen een transactie, niet het programma.
- Transactie mogen niet genest zijn.
- Correctheid: een databank is per definitie altijd in een consistente staat, maar een transactie moet de databank van correcte staat naar correcte staat voeren.
- Het is met de huidige producten niet mogelijk om meerdere statement tegelijk uit te voeren.

13.3 Transaction Recovery

Een transactie begin met een BEGIN TRANSACTION en eindigt met een COMMIT of een ROLLBACK. Commit zorgt voor een commit-punt. Dat wil zeggen het succesvol einde van een transactie, als er een fout optreedt kan dan steeds teruggekeert worden naar het vorige commit-point. Als er een commit-punt gemaakt wordt gebeurd er het volgende:

- Alle update worden permanent gemaakt. Eens een commit gebeurd is, zal de transactie gegarandeerd nooit meer ongedaan gemaakt worden. (met uitzondering van expliciete gebruikers acties)
- Alle posities binnen de databank zijn verloren (i.h.b. cursors bij SQL) en alle *locks* worden terug vrijgelaten.

Hieruit volgt dat transacties niet alleen een eenheid voor werk zijn, maar ook de eenheid voor herstelling. Het is dus nodig dat een transactie voor de commit een aantekening moet maken in het logboek(*write-ahead log rule*.

Eigenschappen die een transactie zou moeten hebben: (ACID)

- Atomicity: transacties zijn atomair
- Correctness: transactie veranderen de ene correcte staat van een databank in een andere. Het is niet noodzakelijjk dat tijdens elke tussenstap de correctheid bewaard blijft.
- Isolation: transacties zijn van elkaar geisoleerd, dit wil zeggen dat de ene transactie de tussenresultaten van een andere niet kan zien.
- Durability: Eens een commit gebeurd is, moeten de updates persistent zijn in de databank.

13.4 System Recovery

Er zijn twee grote categoriën van falingen:

- System failures hebben invloed op alle transacties die aan de gang zijn, maar beschadigen de databank niet fysiek.
- Media failures doen fysieke schade aan de databank, of een deel ervan en hebben invloed op de transacties die dat deel van de databank gebruiken.

Het systeem gebruikt checkpoints om te weten welke transacties herdaan moeten worden en welke ongedaan. Elke keer als er een checkpoint genomen wordt, worden alle buffer naar de fysieke databank geschreven en wordt er een lijst van alle transacties met hun status naar de fysieke log geschreven.

13.4.1 ARIES

Aries doet eerste de redo's en dan de undo's

- Analyse: Bouw de undo en redo lijst op
- Redo: Hermaak de databank zoals ze was op het moment van de crash;
- Undo: Maak de veranderingen door transacties die niet geCommit hebben ongedaan.

13.5 Media Recovery

De databank terug herstellen van een dump en dan alle transacties die gedaan zijn herdoen. Er moet niets ongedaan worden omdat die gegevens toch verloren zijn in de crash.

13.6 Two-Phase Commit

Twee fase commit is vnl. belangrijk als de transactie te maken heeft met meerdere onafhankelijke bronmanagers. Als de transactie succesvol is, moeten alle betrokken databanken committen. Het is niet zo dat de ene databank rollback kan doen en een andere commit. Twee fase commit zorgt ervoor dat als er een systeem-COMMIT is alle systeemcomponenten de opgelegde taak uitvoeren.

- Voorbereiding: De coordinator meldt alle bronmanagers om klaar te staan voor een einde-transactie. Dus eigenlijk moet elke bronmanager zijn log naar fysieke log schrijven. (De bronmanager heeft dus een permanente copie van het werkt dat hij voor transactie verricht heeft) De bronmanagers antwoorden met OK als alles in orde is.
- Commit: De coordinator schrijft een record met de beslissing over de transactie naar zijn fysieke log. Dan stuurt die naar alle deelnemers zijn beslissing. De deelnemers zijn verplicht om de beslissing uit te voeren.

Als de coordinator van de voorgaande procedure crasht kan het wel eens lang duren eer de deelnemers een bericht krijgen over de beslissing. Het is dus een goed idee om de communicatie manager te betrekken in de twee fase commit.

Hoofdstuk 14

Concurrency

14.1 Introduction

De term *concurrency* wijst op het feit dat een databank vaak meerdere transacties toegang verleent tot dezelfde gegevens. Er is dus duidelijk een mechanisme nodig waarmee ervoor gezorgd kan worden dan samenlopende transactie elkaar niet storen.

14.2 Three Concurrency Problems

Het is mogelijk dat twee transacties die op zich elk correct zijn, toch een verkeerd resultaat opleveren.

14.2.1 The Lost Update Problem

Transaction A	Time	Transaction B
-	↓	-
-	↓	-
RETRIEVE t	t1	-
-	↓	-
-	t2	RETRIEVE t
-	↓	-
UPDATE t	t3	-
-	↓	-
-	t4	UPDATE t
-	↓	-

De update die A maakt zal verloren gaan als B eindigt.

14.2.2 The Uncommitted Dependency Problem

Transaction A	Time	Transaction B
-	↓	-
-	↓	-
-	t1	UPDATE t
-	↓	-
RETRIEVE t	t2	-
-	↓	-
-	t3	ROLLBACK
-	↓	-

A is afhankelijk van een niet gecommitte wijziging in tijdstip t2

Transaction A	Time	Transaction B
-	↓	-
-	↓	-
-	t1	UPDATE t
-	↓	-
UPDATE t	t2	-
-	↓	-
-	t3	ROLLBACK
-	↓	-

De wijziging van A zal verloren gaan bij de ROLLBACK.

14.2.3 The Inconsistent Analysis Problem

ACC 1:40	ACC 2: 50	ACC 3: 30
Transaction A	Time	Transaction B
-	↓	-
-	↓	-
RETRIEVE ACC 1: sum=40	t1	-
-	↓	-
RETRIEVE ACC 2: sum=90	t2	-
-	↓	-
-	t3	UPDATE ACC 3: 30 →20
-	↓	-
-	t4	UPDATE ACC 1: 40 →50
-	↓	-
-	t3	COMMIT
-	↓	-
RETRIEVE ACC 3: sum=110	t2	-
-	↓	-

De som zou eigenlijk 120 moeten zijn, maar doordat een andere transactie de data veranderd heeft tijdens de uitvoering van A, is A fout.

14.2.4 A Closer Look

Als A en B twee gelijktijdige transacties zijn, kunnen volgende problemen optreden bij het gelijktijdig gebruikt van een tuple:

- RR: ze willen beiden t lezen, dit geeft geen problemen
- RW: A wil de data lezen en B schrijven. Als A data leest, B veranderd die en A leest die dan weer, ziet A andere data. Dit is een *nonrepeatable read*.
- WR: A schrijft data en B wil die lezen. Die kan tot resultaat een *uncommitted dependency* geven.
- WW: als A en B beide willen schrijven, zal de één vermoedelijk de data van de andere overschrijven.

14.3 Locking

Alle voorgaande problemen kunnen opgelost worden als en transactie een *lock* op een tuple kan krijgen gedurende de transactie. Dit wil zeggen dat andere transacties er geen toegang tot hebben. Er bestaan twee soort locks:

- Exclusive locks, of ook write locks, weigeren elke request door andere transactie voor een lock.
- Shared locks, of ook read locks, weigeren enkel exclusieve locks door andere transacties omdat reads niet met elkaar in conflict komen.

Het is nu nodig om een locking protocol of data access protocol in te voeren zodat de problemen uit de vorige paragraaf niet meer kunnen optreden.

- Een transactie die een tuple wil opvragen moet er eerste een S-lock op vragen.
- Een transactie die een tuple wil updaten moet er eerst een X-lock op vragen. Als de transactie al een S-lock heeft, moet die geupgraded worden.
- Als een transactie B een lock wil op een tuple dat reeds gelocked is, dan gaat B in een wacht staat. Het systeem moet garanderen dat B niet eeuwig moet wachten (livelock,starvation)
- X-locks en S-locks worden terug vrijgegeven op het einde van een transactie.

Het hiervoor beschreven protocol is *strict two-phase locking*.

14.4 The Three Concurrency Problems Revisited

14.4.1 The Lost Update Problem

Transaction A	Time	Transaction B
-	↓	-
-	↓	-
RETRIEVE t (S-lock on t)	t1	-
-	↓	-
-	t2	RETRIEVE t (S-lock on t)
-	↓	-
UPDATE t (request X-lock on t)	t3	-
wait	↓	-
wait	t4	UPDATE t (request X-lock on t)
wait	↓	wait

Er zijn geen updates verloren, maar A en B zullen voor eeuwig op elkaar wachten om een X-lock op t te krijgen.

14.4.2 The Uncommitted Dependency Problem

Transaction A	Time	Transaction B
-	↓	-
-	↓	-
-	t1	UPDATE t (X-lock on t)
-	↓	-
RETRIEVE t (request S-lock on t)	t2	-
wait	↓	-
wait	t3	ROLLBACK (release X-lock)
wait	↓	-
resume:RETRIEVE t (S-lock on t)	↓	-
-	↓	-

Probleem is opgelost en er treedt geen dead-lock op.

Transaction A	Time	Transaction B
-	↓	-
-	↓	-
-	t1	UPDATE t (X-lock on t)
-	↓	-
UPDATE t(request X-lock)	t2	-
wait	↓	-
wait	t3	ROLLBACK (release X-lock)
-	↓	-
resume:UPDATE t (X-lock on t)	↓	-

Probleem is opgelost en er treedt geen dead-lock op.

14.4.3 The Inconsistent Analysis Problem

ACC 1:40	ACC 2: 50	ACC 3: 30
Transaction A	Time	Transaction B
-	↓	-
-	↓	-
RETRIEVE ACC 1	↓	-
(S-lock on acc1): sum=40	t1	-
-	↓	-
RETRIEVE ACC 2	↓	-
(S-lock on acc2): sum=90	t2	-
-	↓	-
-	t3	UPDATE ACC 3
-	↓	(X-lock on acc3):
-	↓	30 →20
-	↓	-
-	t4	UPDATE ACC 1
-	↓	(request X-lock on acc1):
-	↓	40 →50
-	↓	wait
-	t5	wait
-	↓	wait
RETRIEVE ACC 3	↓	wait
(request S-lock on acc3)	↓	wait
: sum=110	t2	wait
wait	↓	wait

Er wordt geen foutief resultaat afgeleverd, maar het systeem gaat in deadlock.

14.5 Deadlock

Strict two-phase locking kan gebruikt worden om de concurrency problemen op te lossen, maar introduceert wel een nieuw probleem, nl. deadlock. Deadlock is een situatie waarin twee of meer transacties simultaan aan het wachten zijn op elkaar om een lock los te laten voor er verder gegaan kan worden. Het detecteren van een deadlock is eigenlijk het vinden van een cylce in de *Wait-For* graaf. Eén van de deadlockers wordt het slachtoffer en wordt ongedaan gemaakt (locks worden als onderdeel van ROLLBACK vrijgegeven) en de andere kunnen hun transactie afwerken. Meestal zal het systeem dergelijke slachtoffer automatisch terug uitvoeren.

14.5.1 Deadlock Avoidance

Het is mogelijk om deadlocks te vermijden voor ze optreden. Er bestaan twee versies van, die allebei volgens het volgende stramien werken:

- Elke transactie krijgt een unieke tijdsstempel
- Als een transactie A een lock aanvraagt op een tuple dat al een lock heeft van transactie B dan:

  - Wait-Die: A wacht als hij ouder is dan B, en sterft als dat niet zo is (ROLLBACK en herstarten)
  - Wound-Wait: Als A ouder is dan verwondt hij B (B wordt geROLLBACKed en herstart) en A wacht

- Als een transactie herstart wordt, behoudt hij zijn originele tijdsstempel.

Uiteindelijk zal elke transactie uitgevoerd worden, maar het nadeel is dat er teveel ROLLBACKs gebeuren.

14.6 Serializability

Een gegeven uitvoering van een verzameling van transacties is serialiseerbaar als en slechts als ze de uitkomst is van een in serie uitgevoerde uitvoering van dezelfde transacties.

Gegeven een verzameling van transacties, eender welke uitvoering van die transacties noemt men een *schedule*. Het uitvoeren van die transacties één per keer in serie, noemt men een *serial schedule*. En schedule dat niet serieel is noemt men *interleaved*. Twee schedules zijn equivalent als ze,ongeacht de toestand van de databank, hetzelfde resultaat produceren.

Two-phase locking theorem: Als alle transacties gehoorzamen aan het two-phase locking protocol, dan zijn alle mogelijke interleaved schedules serialiseerbaar.

14.7 Recovery Revisited

Als er geen lock systeem aanwezig is, of locks te vroeg worden vrijgegeven, dan is het mogelijk dat er cascaderollbacks optreden, of dat sommige transacties *unrecoverable* zijn.

Stricty two-phase locking garandeert echter dat er geen cascades zijn en dat alles recoverable is, dus moet je dat maar gebruiken.

14.8 Isolation Levels

Isolation level dat behoord bij een bepaalde transactie is de hoeveelheid interferentie die transactie duldt van anderen. Als serialiseerbaar gegarandeerd moet zijn, kan er totaal geen toleratie voor interferentie zijn.

Het maximumniveau is repeatable read (RR) waarbij alle locks tot het einde van de transactie vastgehouden worden. Bij cursor stability (CS) worden read-locks vrijgegeven als ze niet geupgrade worden. Cs voldoet niet aan two-phase locking en is daarom ook niet serialiseerbaar. Daarom is het ook niet gegarandeerd dat de databank van de ene correcte toestand in een andere wordt overgezet.

14.8.1 Phantoms

Het phantom probleem kan enkel optreden als het systeem met minder dan maximale isolatie werkt. Een phantom ontstaat als een transactie niet alles

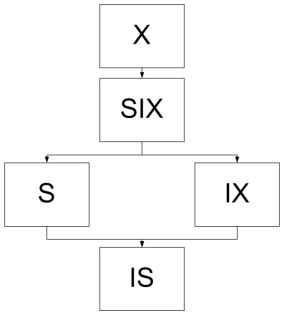
locked wat het logische gezien zou moeten locken. Daardoor kunnen er bij een reread ineens dingen zijn die er daarvoor niet waren.

14.9 Intent Locking

Het is mogelijk om grotere gehelen ineens te locken dan tupels. We spreken dan van *locking granularity*, hoe fijner hoe meer concurrency er mogelijk is, hoe grover hoe minder overhead er nodig is. We introduceren een nieuw locking protocol met 3 nieuwe soorten locks:

- Intent shared locks(IS): een transactie wil S-locks op een aantal individuele tuples van R
- Intent exclusive lock (IX): hetzelfde als IS, maar de transactie wil misschien ook updates dan en dus X-locks nodig hebben.
- Shared Intent Exclusive lock (SIX): transactie kan gelijktijdige lezers verdragen en zal ook een aantal tuples van R updaten.

We kunnen locks nu ordenen in *precedence graph* zie fig. 14.1



Figuur 14.1: Precedence Graph

Locks escalation is een systeem om de overhead te verminderen. Als er een voorgeschreven aantal locks op een object bereikt zijn, zal het systeem een collectie fijne locks vervangen door één grovere lock, waardoor de overhead aanzienlijk verminderd wordt.

Deel V

Further Topics

Hoofdstuk 15

Security

15.1 Introduction

Security betekend data beschermen tegen ongeoorloofde gebruikers. Integrity betekend de data beschermen tegen gemachtigde gebruikers. Er zijn twee algemene mogelijkheden voor beveiliging van een DBMS:

- Discretionary control: Elke gebruiker zal zijn eigen gebruikers rechten hebben. Er zijn amper beperkingen op welke data de ene gebruiker wel en een andere niet kan gebruiken.
- Mandatoy control: Elk object in de database heeft een geassocieerd classificatie niveau. Elke gebruiker heeft ook een zeker toegangsniveau. Dit wil dus zeggen dat data met een bepaalde classificatie enkel toegankelijk zijn voor mensen die het overeenkomstige toegangslevel hebben.

Het maakt niet uit welke van de twee systemen gebruikt wordt, maar er moeten security constraint zijn, en een systeem om die uit te voeren. Ook moet er een verificatiemethode bestaan zodat de juiste security constraints gebruikt worden voor de ingelogde gebruiker. Het is ook mogelijk om users in groepen in te delen zodat de administratie ervan eenvoudiger is.

15.2 Discretionary Access Control

In plaats van te zeggen wat een gebruiker niet mag gaan we met *authorities* definiëren wat hij wel mag.

Een *authority* heeft 4 componenten:

- Naam
- een verzameling rechten
- de relvar waar het over gaat
- de verzameling gebruikers die deze privileges krijgen

Er bestaan verschillende soorten authorities:

- value-(in)dependent

- statistical summary
- context-dependent

15.2.1 Request Modification

Request modificatie is een techniek die gebruikt wordt bij QUEL. Daar wordt het request zodanig aangepast dat het onmogelijk is om een securityconstraint te breken. Enig probleem is wel dat niet alle securityconstraints behandeld kunnen worden.

15.2.2 Audit trails

Een audit trail is een lijst van alle bewerkingen door alle users. Het moet mogelijk zijn,gegeven de juiste authorisatie, om het audit trail te ondervragen, net als de rest van de databank.

15.3 Mandatory Access Control

Het basisidee is dat elk object een classificatielevel heeft en elke user een toegangslevel.

- Gebruiker i kan object j bekijken als de level van i≥j
- Gebruiker i kan object j wijzigen als hun levels gelijk zijn. Dit is nodig omdat het anders mogelijk is om objecten naar een lager/hoger level te brengen.

Er zijn 4 niveaus van veiligheid beschreven door DoD:

- Minimale beveiliging
- Discretionary protection
- Mandatory protection
- Verified protection

15.4 Statistical Databases

Een statische database is een databank die queries toelaat met geaggregeerde informatie (sommen, gemiddeldes,...), maar geen queries toelaat die individuele informatie vrijgeeft.

Het is voor bijna alle statistische databanken mogelijk om een *general tracker* te vinden die kan gebruikt worden om het antwoord te weten te komen voor eender welke niet toegelaten query. Het is in praktijk zelfs meestal niet moeilijk om die te vinden.