Inhoudsopgave

Part 1 : PRELIMINARIES

1 Chapter 1 : An Overview of Database Management

1.1 Introduction

Eerste definitie databanksysteem

> Een databanksysteem is een systeem dat records bijhoudt, het systeem zelf bevindt zich op een computersysteem.

Eerste definitie databank

> Een databank is een container van een collectie van gegevensbestanden, die zich eveneens op een computersysteem bevinden. De databank zit meestal zelf in gegevensbestanden.

Operaties die kunnen worden uitgevoerd op het systeem door systeemgebruikers

- Nieuwe files toevoegen aan de databank (ADDING)
- Gegevens in de bestaande files invoegen (INSERTING)
- Data opvragen van de bestaande files (RETRIEVING)
- Verwijderen van data uit bestaande files (DELETING)
- Veranderen van data in bestaande files (CHANGING)
- Verwijderen van bestaande files uit de databank (REMOVING)

Voorbeeld 1.1 (pagina 4 tem. 6)

> Een eerste kennismaking met databanksystemen. Figuren 1.1, 1.2 en 1.3 worden hierin gebruikt.
>
> **Notities :**
>
> - SELECT, INSERT, ... worden statements, commando's of operatoren genoemd.
> - SQL = Structured Query Language (internationale standaard). In SQL noemt men een file TABEL, met records RIJEN en fields KOLOMMEN.

- Er zijn verschillend datatypes mogelijk voor de kolommen.
- Nieuwe termen :
  - relatie = tabel
  - kolom = attribuut
  - rij = tuple
  - primary key = uniek attribuut van een tabel

1.2 What is a database system?

Tweede definitie databanksysteem

> Een databanksysteem is een systeem met als doel informatie te bewaren en gebruikers toe te staan om informatie op te vragen en aan te passen. De verschillende mogelijkheden geboden door zo een systeem hangen onder meer af van de grootte en kracht van de onderliggende machine.

Veschil tussen data en informatie

> Data is wat bewaard wordt in de databank en informatie duidt men aan als de betekenis van de data geïnterpreteerd door de gebruikers.

Figuur 1.4 : Vereenvoudigd databanksysteem (pagina 7)

> **Notities :**

- Vier grote componenten van het systeem :
  - data
  - hardware
  - software
  - users

Data

Definitie single-user system

Een single-user system is een systeem waarbij slechts 1 gebruiker toegang heeft tot de databank op elk moment.

Definitie multi-user system

Een multi-user system is een systeem waarbij meerdere gebruikers toegang hebben tot de databank op hetzelfde moment. Het objectief van dit soort systeem is te doen alsof een multi-user de enige gebruiker is van het systeem. Speciale problemen bij multi-user systemen zijn interne problemen voor het systeem, deze zijn niet zichtbaar voor de gebruiker (Chapter 16).

Verschillende soorten data

- Data bewaard in 1 databank
- Data gespreid over verschillende databanken
- Integrated data : bij deze data wordt de databank beschouwd als een unificatie van verschillende (te onderscheiden) files. Men tracht zo weinig mogelijk zelfde info op te slaan, men bekijkt het als 1 geheel. Deze data komt voor bij zowel kleine als grote systemen.
- Shared data : bij deze data kan eenzelfde databank geshared worden tussen verschillende gebruikers. Dus verschillende gebruikers hebben toegang tot dezelfde data, op hetzelfde moment. Deze data komt enkel voor in grote systemen.
- Personal/Application specific databank : databank dat niet geshared wordt.

Hardware

Hardware bestaat uit :

- Opslagmedia : secondair geheugen (om data op te slaan) + Input/Output ontwerpen
- Processoren en hoofdgeheugen die gebruikt worden om de uitvoering van de databanksysteem software te ondersteunen.

Software

Database Management System (DBMS)

> De DBMS is eigenlijk de software - de laag tussen de fysische databank(data) en de systeemgebruikers - behandelt al de aanvragen voor toegang tot de databank. Alle geziene operaties (insert,delete,...) worden voorzien door de DBMS. De DBMS zorgt er dus voor dat de gebruiker zich niets hoeft aan te trekken van de hardware. De DBMS is de belangrijkste software component van het systeem, maar is niet de enige.

Users

Drie klassen van gebruikers :

- Applicatie-programmeurs : zij schrijven databankapplicaties in een bepaalde programmeertaal. Deze programma's hebben toegang tot de databank via een aanvraag (SQL) aan de DBMS. De programma's zelf kunnen online (interactief) of offline zijn.
- Eindgebruikers : deze gebruikers krijgen toegang tot de databank via online programma's of door gebruik van een interface aangeboden door het systeem zelf. De interfaces zelf worden ondersteund door online applicaties (*built-in*, query language processor). Er bestaan ook *menu-driven* interfaces (voor gebruikers leek aan de IT-sector, mbv. menu's en boxen worden requests uitgevoerd). en *command-driven* interfaces (query languages, voor gebruikers uit IT-sector, je kan meer).
- Databank administrator (DBA) : deze gebruiker heeft de grote verantwoordelijkheid, de gebruiker met alle privileges.

1.3 What is a database?

Tweede definitie databank

> Een databank is een collectie van persistente data, die gebruikt wordt bij sommige applicatiesystemen. De data is persistent omdat eens het geäccepteerd is door de DBMS, het enkel kan verwijderd worden met een bepaalde request. (niet als side-effect)

Entiteiten en relationships

Een entiteit is elk onderscheidbaar object dat moet worden gerepresenteerd in een databank. De relationships zijn dus links tussen de basisentiteiten, deze zijn ook deel van de data, zoals de basisentiteiten. Ze moeten dus ook in de databank gerepresenteerd worden.

Figuur 1.6 : Entity/Relationship diagram (pagina 12 tem 14)

Notities :

- De relationships zijn bidirectioneel : ze kunnen in beide richtingen doorlopen worden.
- Binaire relationships : relationships tussen twee entiteiten (ook : teraire, n-aire). Opmerken dat een ternaire relationships niet equivalent is met een combinatie van drie binaire relationships. Men spreekt van een *connection trap*.
- Entiteiten en relationships worden voorgesteld door relaties.
- Relatie wordt gebruikt in termen van relationele databanken (tabel). Er is een wezenlijk verschil.
- Bill of materials : een relationship van 1 entiteit (de relationship is nog altijd binair).
- Er kunnen meerdere relationships zijn tussen dezelfde set van entiteiten.
- Relatie equivalent met entiteit : enkel als we een andere definitie van entiteit beschouwen (= elk object waarover we informatie willen opslaan).

Eigenschappen

Aangezien een entiteit kan gezien worden als elk object waarover we informatie willen opslaan, kunnen entiteiten dus eigenschappen bezitten. Deze eigenschappen moeten ook gerepresenteerd worden in de databank. Deze eigenschappen kunnen simple of complex zijn. (en ze kunnen gerepresenteerd worden door simpele of complexe datatypes).

Data

We kunnen data en databanken nog op een andere manier benaderen. Data zijn eigenlijk gegeven feiten, eigenlijk een logische ware propositie. Dus een databank is eigenlijk een collectie van ware proposities.

Data Model (in relationeel model)

SQL is gebaseerd op het relationele model van data.

Definitie relationeel model

> Een relationeel model is de data gerepresenteerd als rijen in tabellen (tupels in relaties) en de rijen als ware proposities. Operatoren zijn voorzien voor operaties uit te voeren op de rijen van een tabel. Dit model is één van vele data modellen.

Definitie data model

> Een data model is de abstracte, logische definitie van objecten, operatoren,... die samen een abstracte machine vormen waarmee gebruikers kunnen interageren. De objecten modelleren de structuur van de data. De operatoren leggen de houding vast.

Model vs. Implementatie

> De implementatie van een datamodel is de fysieke realisatie, de gebruikers moeten het model kennen, maar ze moeten niet de implementatie kennen.

Andere defintie van een Data Model

> Een datamodel is een model van persistente data van één of andere organisatie. Een datamodel is vooreerst zoals een programmeertaal. Anderzijds is een datamodel een specifiek programma geschreven in die taal.

1.4 Why database ?

Voordelen van een databank in geval van single-user

- Papier sparen
- Compactheid (geen papieren bestanden)
- Snelheid (machine rapper dan de mens)
- Minder lastig werk (beter met machines)
- Up to date
- Beveiliging (tegen verlies of onwettige toegang)

Dit geldt ook voor multi-user systemen, maar deze hebben nog een bijkomend voordeel : het databanksysteem voorziet de organisatie met gecentraliseerde controle van de gegevens.

Data Administrator en Databank Administrator

Data Administrator (DA)

De DA is de persoon in de organisatie met de centrale verantwoordelijkheid over de data (bij gecentraliseerde controle van de gegevens). Hij/zij begrijpt de data, bevindt zich op een senior management level. Hij/zij beslist welke data op te slaan en beslist hoe de data te beheren. Hij/zij is manager $\neq$ technieker.

Databank Administrator (DBA)

De DBA is de technische persoon om de data te implementeren, opgegeven door de DA. Hij/zij is IT-specialist en zorgt ervoor dat het systeem opereert met goede performantie. Het kan ook een team van mensen zijn.

Voordelen van databank toegang door centrale controle

- Data kan geshared worden.
- Overtolligheid wordt verminderd : geen overtollige bestanden, voorkomen van overtollige opslagruimte. Het is de taak van de DBMS.

- Inconsistentie kan vermeden worden : stel dat een feit gerepresenteerd wordt door twee entries en DBMS weet dit niet, dan zal bij een update, er maar één geupdated worden, de ander niet. Dus dit leidt tot incorrecte info aan de gebruiker.
- Transactie ondersteuning kan worden aangeboden : transactie gaat enkel door als alles goed loopt, anders wordt er niets gedaan.
- Integriteit kan worden gehandhaafd : probleem met verkeerde info, oplossen door aanbieden van gecentraliseerde controle en bieden van beperkingen (te controleren bij updates). Integrity constraints moeten gecontroleerd worden.
- Veiligheid (security constraints)
- Balanceren van het systeem : DBA doet wat het beste is voor het systeem.
- Standaarden : enkel voordelen bij verplaatsen van data tussen de systemen.
- data onafhankelijkheid

Categoriën van databanksystemen afhankelijk van hun datastructuur

- Oudste systemen : hiërarchisch opgebouwd (men moet aantal accessen minimaliseren), netwerkdatabanken, ...
- Hedendaagse systemen : objectdatabanken

1.5 Data independence

Soorten data onafhankelijkheid

Er zijn twee soorten data onafhankelijkheid : fysische en logische.

Data-afhankelijkheid niet toegestaan

- Aanpassen van fysieke representatie en toegangstechnieken kunnen de applicatie drastisch aantasten, omdat de applicaties gebouwd zijn rond deze kennis van representatie.
- Verschillende applicaties kunnen een verschillende kijk hebben op dezelfde data.

- De DBA moet de vrijheid hebben om de fysische representatie en toegangstechnieken te veranderen, zonder dat bestaande applicaties moeten veranderd worden.

Definitie dataonafhankelijkheid

De immuniteit van applicaties om de fysieke representatie en toegangstechnieken te veranderen, hieruit volgt dat de applicaties niet afhangen van een fysieke representatie of toegangstechniek. Dataonafhankelijkheid is een reden om het datamodel te scheiden van zijn implementatie. Dataonafhankelijkheid moet toelaten dat de databank kan groeien.

Enkele definities

- *Figuur 1.7 (pg 23)*
- stored field : de kleinste eenheid van opgeslagen data. (vb partnumber) De databank bevat veel instanties van elk verschillend type van het stored field. (vb srew, lid,...)
- stored record : collectie van stored fields (vb parts)
- stored file : collectie van alle bestaande instanties van één type van stored record.

Wat wil de DBA kunnen veranderen ?

- representatie van numerieke data (fixed vs floating)
- representatie van karakter data (ASCII vs Unicode)
- eenheden van numerieke data (inches vs cm)
- data coding (Red =1, Blue = 2, ...)
- data materialisatie : logical field komt overeen met stored field (direct) of juist niet (indirect)
- structuur van stored records
- structuur van stored files (1 disk vs verschillende)

1.6 Relational systems and others

Wat is een relationeel systeem (kort)

De data gezien door de gebruikers zijn tabellen, de operatoren die voor handen zijn halen nieuwe tabellen uit oude.

Waarom wordt een systeem relationeel genoemd?

Relation is een mathematische term voor tabel.

Verschillende soorten databanksystemen

Figuur 1.8 (pg 27)

Bij relationele systemen ziet de gebruiker tabellen, bij andere systemen ziet de gebruiker ook andere datastructuren. Een karakteristiek van relationele systemen is dat er geen pointers meedoen (toch niet zichtbaar voor de gebruiker), bij andere systemen kan dat wel het geval zijn (vb. als ze met boomstructuur werken). De oudste systemen zijn inverted lists, hierarchic en netwerk systemen (volgens de datastructuren en operatoren die ze gebruiken).

2 Chapter 2 : Database System Architecture

2.1 Introduction

ANSI/SPARC architectuur

We trachten een framework op te bouwen die kan dienen als architectuur.

2.2 The three levels of the architecture

De drie levels van de ANSI/SPARC architectuur

- Internal level : opslag level, dichtst bij fysieke opslag, houdt zich bezig met de manier van opslag. (implementatielevel,machine-geöriënteerd)
- External level : gebruiker logisch level, dichtst bij de gebruikers, houdt zich bezig met hoe de data wordt gezien door de gebruiker. (modellevels,gebruiker geöriënteerd) (individuele gebruikers)
- Conceptual level : gemeenschappelijk logisch level, indirectie tussen de twee andere levels. (modellevels) (hele community gebruikers)

Tussen internal en external level is er fysische data onafhankelijkheid, tussen external en conceptual level is er logische data onafhankelijkheid.

Figuur 2.1 : De drie levels van de architectuur (pagina 34)

Views

Aangezien de gebruiker meestal niet geïntresseerd is in de gehele databank, bestaan er veel verschillende *external views*, die elk bestaan uit een abstracte representatie van een deel of van de gehele databank. Er is juist één *conceptual view*, die bestaat uit een abstracte representatie van de gehele databank. Er is ook juist één *internal view*, met de representatie van de databank zoals die is opgeslaan. External en conceptual levels, zijn model levels (user-oriented) en het internal level is een implementatie level (machine-oriented).

Figuur 2.2 : Een voorbeeld van de drie levels (pagina 35 tem. 36)

Notities :

- Conceptual level : databank met info over een entiteitstype met zijn eigenschappen en plaatsinname.
- Internal level : geeft aantal bytes voor opslag per record en de naam van de opgeslagen record type (stored record + stored field).
- External view van PL/I gebruiker : representatie van de data door PL/I record in bepaalde declaratie volgens PL/I regels.
- External view van COBOL gebruiker : representatie van de data door COBOL record in bepaalde declaratie volgens COBOL regels.
- Mappings : Corresponderende data kan verschillende namen hebben op de verschillende levels. Het systeem moet op de hoogte zijn van deze correspondenties (mappings).

Drie levels van de architectuur bij relationele systemen

- Internal level : dit level is niet relationeel, want de objecten die daar worden opgeslagen zullen geen relationele tabellen zijn. Het zullen objecten zijn zoals die voorgesteld worden bij internal levels van andere systemen.
- Conceptual level : dit level is relationeel , want de objecten op dat level zullen relationele tabellen zijn en de operatoren zullen relationele operatoren zijn.
- External view : de view zal relationeel zijn.

Figuur 2.3 : Gedetailleerd systeem architectuur (pagina 37)

2.3 The external level

Definitie external level

Het external level is het individuele gebruikerslevel en de gebruiker kan applicatieprogrammeur of eindgebruiker (of DBA, deze heeft ook intresses bij de andere twee levels) zijn.

Elke gebruiker heeft een taal ter beschikking

Voor de applicatieprogrammeur, zullen die talen programmeertalen zijn zoals Java, C++ of een taal eigen aan het systeem (*fourth-generation of 4GL's*). De eindgebruikers gebruiken query talen (SQL) of speciale menu -en formdriven talen.

Elke taal bevat een *data subtaal (DSL)*, dit is een subset van de totale taal dat zich specifiek bezighoudt met databankobjecten en operaties. DSL is ingebed in de host taal (deze zijn op hun beurt verantwoordelijk voor verschillende faciliteiten die los staan van databanken). Een voorbeeld van DSL is SQL (deze kan interactief gebruikt worden of ingebed in andere talen).

DSL en HL worden *tightly coupled* (gemakkelijker voor de gebruiker, meer werk voor systeemontwerpers) genoemd als de talen dicht tegen elkaar aanleunen en moeilijk te onderscheiden zijn (vb Oracle). Ze worden *loosely coupled* genoemd als ze gemakkelijk te onderscheiden zijn.

Elke DSL bestaat uit een *Data Definition Language (DDL)*, die de definitie of declaratie van de objecten ondersteunt en uit een *Data Manipulation Language (DML)*, die zorgt voor de manipulatie van de objecten.

External view

De external view is de individuele kijk van de gebruiker op de databank. Het is dus de inhoud van de databank, gezien door een bepaalde gebruiker. Het is dus eigenlijk de databank zelf gezien door de gebruiker.

External record

Een external view bestaat dus uit vele instanties van vele types van external record (niet noodzakelijk gelijk aan opgeslagen records). De DSL van de gebruiker is gedefinieerd in termen van external records.

External schema

Elke external view is gedefinieerd dmv external schema. De external schema bestaat uit definities van de verschillende external record types in de beschouwde external view. De external schema is geschreven door gebruik van de DDL (external DDL).

2.4 The conceptual level

Conceptual view

De conceptual view is de representatie van de gehele informatie inhoud van de databank, in abstractere vorm dan inwelke de data fysiek opgeslagen is. Het is de data zoals ze is, view van de totale databankinhoud (verschillend van de data gezien door de gebruikers).

Conceptual record

Een conceptual view bestaat dus uit vele instanties van types van conceptual record (niet gelijk aan opgeslagen records of aan record gezien door gebruikers).

Conceptual schema

Conceptual schema is een definitie van de conceptual view. De conceptual view is gedefinieerd dmv het conceptual schema. Deze schema's bevatten definities van verschillende conceptuele record types. De conceptual schema is geschreven door gebruik van de DDL (conceptual DDL), er moeten geen details zijn voor het opslaan van low-level objecten of toegangstechnieken (door fysieke data onafhankelijkheid). Enkel definities over de informatie inhoud. In deze schema's zitten ook veiligheid en integriteit.

2.5 The internal level

Internal view (stored database)

De internal view is een low-level representatie van de gehele databank.

Internal record (stored record)

Een internal view bestaat dus uit vele instanties van types van internal record (gelijk aan de ANSI/SPARC term om de opgeslagen records aan te duiden, wel nog verschillend van de fysieke records (*blocks of pages*).

Internal schema (stored database defenition)

De internal view is gedefinieerd dmv internal schema. Deze schema's bevatten definities van verschillende internal record types en ook de indices, hoe velden zijn opgeslagen,... . De internal schema is geschreven door gebruik van de DDL (internal DDL).

2.6 Mappings

Verschillende mappings

Er is één conceptual/internal mapping en verschillende external/conceptual mappings.

- Conceptual/Internal mapping : correspondentie tussen de conceptual view en stored database. Het toont hoe conceptual records gerepresenteerd zijn op het internal level. Als er een verandering in de stored database plaatsvindt, moeten we de mapping ook veranderen. (dan kan de conceptual schema invariant blijven, taak van de DBA/DBMS). Het is de sleutel voor fysische (gebruikers immuun voor veranderingen op fysieke structuur van de stored database) data onafhankelijkheid.
- External/Conceptual mapping : correspondentie tussen een external view en de conceptuele view. Het is de sleutel tot logische (gebruikers imuun voor veranderingen aan de logische structuur van het conceptual level) data onafhankelijkheid.
- Fysieke data onafhankelijkheid : wanneer gebruikers en programma's imuun zijn aan veranderingen in de fysieke structuur van de stored database.
- Logische data onafhankelijkheid : wanneer gebruikers en programma's imuun zijn aan veranderingen in de logische structuur van de databank (dus veranderingen op het conceptuele level).
- External/External mappings

2.7 The database administrator

DA maakt de strategische en politieke beslissingen (op abstracte level). DBA voorziet technische ondersteuning om die beslissingen te implementeren (gehele controle op technisch niveau).

Verschillende taken van de DBA op technisch niveau

- Conceptual schema definiëren : DA moet beslissen welke informatie in de databank moet gehouden worden, dus om de

entiteiten te identificeren en ook om de informatie tot die entiteiten te identificeren (dit is logisch/conceptueel databank design). Hierna zal de DBA het corresponderende conceptual schema creëren, mbv. de conceptual DDL. De gecompileerde vorm van dat schema wordt door de DBMS gebruikt om te kunnen antwoorden op de vragen van de gebruikers (de ongecompileerde vorm is een referentiedocument voor de systeem gebruikers).

- Internal schema definiëren : DBA beslist HOE data op te slaan (dit is fysische databank design). DBA creëert de internal schema, mbv internal DDL. DBA moet ook de conceptual.internal mapping definiëren. Het fysische design moet altijd na het logisch design (HOE na WAT).
- Verbinden met gebruikers : DBA moet verbinden met gebruikers en verzekeren dat de data aanwezig is en moet external schema's schrijven, mbv. external DDL. De external/conceptual mapping moet gedefinieerd worden.
- Veiligheid -en integriteit constraints definiëren : dit is een onderdeel van het conceptual schema. De conceptual DDL moet beperkingen kunnen opleggen.
- Dump/restore schema's definiëren : wanneer een organisatie gebonden is aan een databank, moet de operatie van het systeem succesvol zijn. Er moet mogelijkheid zijn om data te herstellen, met zo weinig mogelijk effecten op de rest van het systeem. DBA moet een Damage Control Schema definiëren. Goed idee om data te spreiden (*Very Large Database (VLDB)*).
- Performantie en reageren op veranderingen : DBA moet performantie bereiken dat 'het beste is voor de organisatie' en juiste veranderingen aanbrengen (*tuning*) om de performantie hoog te houden (conceptual level constant proberen houden).

2.8 The database management system

DBMS

De DBMS is de software die alle toegang tot de databank behandeld. Conceputeel gebeurt er dit :

- Een gebruiker vraagt toegang tot request, mbv. een bepaalde DSL(vb SQL).
- De DBMS aanvaardt de request en analyseert deze.
- De DBMS inspecteert het external schema van de gebruiker, de corresponderende external/conceptual mapping, het conceptual schema, de conceptual/internal mapping en de opgeslagen databank definitie.
- De DBMS voert de nodige operaties uit op de opgeslagen databank.

Functies van DBMS

- *Figuur 2.4 : Belangrijke DBMS functies en componenten (pg 46)*
- Data definitie : DBMS moet datadefinities aanvaarden in source vorm en ze omzetten naar de juiste object vorm. DBMS moet DDL processor of DDL compiler componenten voorzien voor elk van de verschillende DDLs. De DBMS moet deze DDLs begrijpen.
- Data manipulation : DBMS moet aanvragen kunnen behandelen. De DBMS moet een DML processor of DML compiler componenten voorzien om te kunnen omgaan met DML. DML aanvragen kunnen planned (request is voorzien, DBA garandeert goede performantie) of unplanned (request is niet voorzien, fysieke databank kan of kan niet gepast zijn voor deze request) zijn.
- Optimalisatie en uitvoering : DML requests moeten behandeld worden door de optimizer component, met als doel de request efficient te implementeren. Deze requests worden dan uitgevoerd door de run-time manager.
- Data beveiliging en integriteit : de DBMS moet requests die ingaan tegen de veiligheid en integriteit afwijzen.
- Data herstel en concurrency : de DBMS (transaction manager of TP monitor) moet herstel en concurrency controle voorzien.
- Data dictionary : dit is een databank op zichzelf, een systeem databank, data over de data. Deze bevat de schema's, de constraints,...
- Performantie : DBMS moet zijn taken zo efficient mogelijk uitvoeren. Het doel van de DBMS is de databank voorzien van een user interface. De interface zelf bevindt zich lager dan wat de gebruikers niet zien. De interface is op het external level.

Verband Operating System en databank

Als er iets fout gaat met DBMS, wordt dit opgevangen door het Operating System. DBMS is een groot stuk software die draait op het Operating System. DBMS maakt gebruik van componenten uit het Operating System.

2.9 Data Communications

Data communication manager

De data communication manager (DC) is geen deel van de DBMS en behandelt alle berichten (transmissies van berichten) van en naar applicaties en de DBMS. De DBMS zorgt voor de databank, de DC behandelt alle berichten van en naar de applicaties die de DBMS gebruiken. We spreken dan van DB/DC system.

2.10 Client/Server Arhitecture

Figuur 2.5 : Client/Server architectuur (pagina 49)

Notities :

- Het doel van een databanksysteem is de ontwikkeling en de uitvoering van databankapplicaties te ondersteunen. Het systeem kan bekeken worden als een eenvoudige twee-delige structuur met een server (back end) en client(s) (front ends).
- De bedoeling van een databank systeem is om de ontwikkeling en uitvoering van de databankapplicaties te ondersteunen.
- De server is juist de DBMS zelf.
- De clients zijn de verschillende applicaites die bovenup de DBMS draaien (zowel user -en built in applicaties). Voor de server zijn die applicaties allemaal hetzelfde, namelijk de interface (external interface) op de server.

Verschillende soorten applicaties

- user-written : gewone applicatieprogramma's

- Vendor-provided applications (*tools*) : applicaties met als doel bij te staan in het ontwerp en creatie en uitvoering van andere applicaties. (Eindgebruikers kunnen applicaties creëren zonder de talen te kennen). (Vb. spreadsheets, Query language processors)

Distributed processing

Als gevolg van de 2 delen client en server en zodat de mogelijkheid er is client en server op verschillende mahines te draaien. Dus verschillende machines vormen een netwerk zodat een taak kan gespreid worden over verschillende machines in dat netwerk..

2.11 Utilities

Definite utilities

Utilities zijn programma's ontworpen om de DBA te helpen met bepaalde administratieve taken. Sommige werken op external level, andere op internal level.

Verschillende soorten utilities

- Load : initiële databank creëren van een gewoon databestand
- Unload/reload (dump/restore) : om de gehele databank te unloaden of om backup te verzorgen
- Reorganization : om data te reorganiseren
- Statistical : om performantie te verbeteren
- Analysis : om statistieken te analyseren

2.12 Distributed Processing

Definitie distributed processing

Distribued processing betekent dat verschillende machines een netwerk kunnen vormen, zodat één enkele datataak kan worden verdeeld over de verschillende machines in het netwerk. (*parallel processing*, wanneer de machines fysiek dicht bij elkaar zijn in een parallel systeem). Communicatie tussen de verschillende machines gebeurt door een netwerk management software (uitbreiding van de DC manager of afgescheiden software component).

Figuur 2.6 : Clients en servers op verschillende machines (pagina 52)

Notities :

Voordelen van deze organisaties:

- Client en server zijn parallel, dus de antwoordtijd verbetert.
- Betere DBMS performantie
- Betere interfaces voor gebruiker, dus gemakkelijker gebruik voor de gebruiker
- Verschillende clients hebben toegang tot een server. Dus een databank kan gedeeld worden door verschillende gebruikers. *Figuur 2.7 : Eén server, meerdere clients (pagina 53)*
- Elke machine kan zich ook gedragen als server voor sommige gebruikers en de client voor andere. *Figuur 2.8 : Elke machine is client en server (pagina 54)*
- Een client kan toegang hebben tot verschillende servers op twee manieren :
  - Toegang tot elke server, maar één per één (je moet weten waar de data zich bevindt).
  - toegang tot verschillende servers gelijktijdig (lijkt op één server).
  - Dit resulteert in *distributed database system.* Dit betekent dat één enkele applicatie de mogelijkheid moet hebben om transparant te werken op data dat gespreid is langs verschillende databanken, gerund door verschillende DBMS, draaiend op verschillende machines, gesupporteerd door verschillende besturingssystemen en geconnecteerd door verschillende netwerken. Transparant betekent hier dat moet blijken dat alles draait op één DBMS, op een machine

3 Chapter 3 : An Introduction to Relational Databases

3.1 Introduction

Relationele systemen

De theoretische basis voor zulke systemen. Een informele definitie tot relationele systemen.

3.2 An informal look at the relational model

Aspecten van het relationeel model

Relationele systemen zijn gebaseerd op een formele basis of theorie (het relationeel model van data).

- Structureel aspect : de data wordt gezien als tabellen door de gebruiker.
- Integriteit aspect : tabellen voldoen aan integrity constraints.
- Manipulatief aspect : er zijn operatoren beschikbaar voor de gebruiker om tabellen af te leiden mbv. restrict, project en join.

Voorbeeld 3.1 (pagina 60 tem. 61)

Dit zijn voorbeelden mbt. restrict, project en join. Figuren 3.1 en 3.2 worden hierin gebruikt.

Definitie closure

Closure is de sluitingseigenschap : het resultaat van de drie operatoren restrict, join en project is op zichzelf een tabel, dus de output van een operatie kan de input worden van een andere operatie (nested relational expressions). We spreken op het conceptual level.

Definitie materialized evaluation

Dit is wanneer tussenliggende resultaten al een tabel vormen in het systeem. Dit is tegengestelde van *pipelined evaluation*, hier worden tussenliggende resultaten doorgegeven aan andere operaties.

Definitie set-a-time operaties

De resultaten zijn volledige tabellen, niet enkele rijen. Ze bevatten sets van rijen.

Enkele puntjes

- Het relationele systeem vraagt enkel dat de databank door de gebruiker gebruikt wordt als tabellen. Tabellen zijn de logische structuur in een relationeel model. De tabellen representeren een abstractie van hoe de data is fysiek is opgeslagen. Conceptual en external level in het relationele systeem zullen beide relationeel zijn, maar het internal level niet. Het is dus enkel begaan met hoe de gebruiker de databank ziet.
- Relationele systemen houden zich aan Het informatie principe : de gehele informatie inhoud van de databank wordt voorgesteld door een manier : als expliciete waarden in kolomposities in rijen in tabellen. Er zijn geen pointers om twee tabellen te verbinden ! De verbinding wordt gerepresenteerd door de aanwezigheid van de waarden.

Voorbeeld 3.2 (pagina 63)

Voorbeeld voor een eerste kennismaking met primary key en foreign key.

Een Formele definitie

Het relationeel model bestaat uit 5 componenten

- collectie van scalaire types (+boolean)
- relatie type generator : om nieuwe types aan te maken
- relatievariabelen : kunnen in de loop van de tijd waarden aannemen
- relationele assignatie operatie : om waarden toe te kennen aan variabelen
- collectie van relationele operatoren (relationele algebra): om waarden van andere waarden af te leiden

3.3 Relations and relvars

Waarom wordt de databank relationeel genoemd?

Relatie betekent : (mathematisch) tabel en een relationele databank is eigenlijk een tabel met data. (Termen in relationeel model : Tabel = relatie (relation value) , Record = tuple, Field = attribuut, Relation Variabele = relvar).

Waarom werd relatie in de eerste plaats ingevoerd?

Relationele systemen zijn gebaseerd op het relationele model en deze is op zijn beurt een abstracte theorie van gegevens gebaseerd op aspecten uit de wiskunde.

Figuur 3.3 : Relvar EMP na het verwijderen van een rij (pagina 65 tem. 66)

Notities :

We onderscheiden relatie variabelen en relatie waarden. Bij het verwijderen, invoegen of updaten van een rij, wordt eigenlijk de oude relatiewaarde(per se) vervangen door een nieuwe.

3.4 What relations mean

Types

Kolommen hebben geassoscieerde datatypes. Gebruikers kunnen eigen types definiëren , dus eigen typologie op niveau van de relaties/attributen.

Elke relatievalue heeft twee delen

- Heading : set van kolommen, predikaat = die relatie heeft een attribuut van dat type, tuple moet true geven voor de heading van de relatie.
- Body : set van rijen die voldoen aan de heading. Elke rij van de body is een ware propositie.

Types vs. relaties

Types zijn sets van dingen waar we over kunnen praten (zelfstandige naamwoorden), relaties zijn sets van dingen die we zeggen over dingen waarover we kunnen praten (zinnen). Types en relations zijn noodzakelijk, voldoende en verschillend.
Relvars hebben ook predicaten, het predikaat dat gemeenschappelijk is voor alle mogelijk relation values van de beschouwde relvar.

Voorbeeld 3.3 (pagina 67-68)

Dit is een voorbeeld mbt. relaties. Figuur 3.4 wordt hierin gebruikt.

3.5 Optimization

Definitie relationele talen

Relationele talen zijn non-proceduraal (hoger level van abstractie). Het systeem moet de juiste tuples geven. Hoe ? Dit weet je niet als gebruiker. Liefst zo snel mogelijk. Dus de automatische navigatie gebeurt door relationele systemen. In niet-relationele systemen moet de gebruiker zelf zorgen voor de navigatie.

Figuur 3.5 : Automatische vs. manuele navigatie (pagina 69)

Definitie optimizer

De optimizer is een component van de DBMS die beslist hoe de automatische navigatie wordt uitgevoerd. Hij moet voor elke request van de gebruiker een efficiënte weg vinden om de request te implementeren. (scannen of met index,...)

Notities :

De gebruikers specifiëren enkel welke data ze willen. Er wordt enkel een projct en select gedaan.

3.6 The Catalog

Definitie cataloog of woordenboek

De cataloog wordt voorzien door de DBMS. Het is de plaats waar al de schema's en corresponderende mappings worden bijgehouden. Deze houdt gedetailleerde informatie bij, dat intressant is voor het systeem zelf. De cataloog zelf bestaat uit relvars. Gebruikers kunnen vragen stellen aan de cataloog. Een cataloog bevat descriptors voor de verschillende items , die intressant zijn voor het systeem.

Voorbeeld 3.4 (pagina 71 tem.72)

Notities :

Waarom kan alles niet in een tabel ?

> Dat gaat technisch niet. En in een tabel kan je geen predikaat creëren die alle attributen bevat. Het is geen vaste structuur, want verschillende attributen. Predikaat kan niet gedefinieerd worden.

3.7 Base relvars and views

Enkele definities

- base relvar : de origineel gegeven relvars (tabellen in een relationeel systeem). Relationele systemen hebben voorzieningen om base relvars te creëren. Base relvars moeten een naam hebben (named relvar).
- base relation : de waarde van de base relvar (set van tuples in een relationeel systeem).
- derived relation : een relatie die geen base relation is, maar die kan bekomen worden uit de base relation mbv. relationele expressies.
- view : base relvar met naam, waarvan de waarde een derived relation is op elk moment (named ,derived relvar ,virtual relvar). Een view moet je ook creëren.
- view-defining expression : wordt niet geëvalueerd, maar enkel onthouden door het systeem. Men houdt het enkel bij in de catalog. De gebruiker denkt dat het een relvar is, en men kan er operaties op uitvoeren alsof het een base relvar was. Veranderingen aan de onderliggende relvar zal merkbaar zijn in de view en omgekeerd.

- *Voorbeeld 3.5 (pagina 72 tem. 75)*

Views in relationele context vs. ANSI/SPARC architecture

Op het external level van deze architectuur, wordt de databank gezien als een external view, gedefinieerd door een external schema. In relationele systemen is een view een named, derived , virtual relvar.

Verschillen tussen base relvar en views

- Base relvars bestaan echt op de manier dat zij data representeren dat fysiek opgeslagen is in de databank (wel niet echt ...).
- Views, daarentegen, bestaan niet echt , maar voorzien verschillende manieren van kijken naar de echte data.

3.8 Transactions

Definitie transactie

Een transactie is een logische werkeenheid. De gebruiker moet in staat zijn het systeem te informeren wanneer verschillende operaties deel uitmaken van eenzelfde transactie mbv. BEGIN TRANSACTION, COMMIT en ROLLBACK.

Enkele puntjes

Een transactie = ASID

- Atomic : ofwel gaat de transactie door ofwel niet.
- Serializable : zelfde actie geeft altijd zelfde resultaat op zelfde moment.
- Isolated : veel mensen tegelijk, maar iedere gebruiker toch afgezonderd van een ander, geen effect op elkaar.
- Durable : blijvend effect op de databank.

3.9 The supplier-and-parts database

Figuur 3.8 : De suppliers-and-parts databank (pagina 77)

4 Chapter 4 : An Introducion to SQL

Part 2 : THE RELATIONAL MODEL

5 Chapter 5 : Types

5.1 Introduction

Definitie type

Een type is een set van waarden. Elke waarde, elke variabele, elke parameter, elke read-only operator, elke relationeel attribuut bezit een type. In de implementatie van een databank is het aantal types eindig, want het geheugen is ook eindig. Wanneer een relvar het type X bezit wil dit eigenlijk zeggen dat waarden van dat attribuut van dat type X zijn.
Enkele puntjes :

- Types worden ook domains genoemd.
- Het type INTEGER bijvoorbeeld is de set van alle integers, we zeggen dat het de set is van alle integers die in staat zijn om gerepresenteerd te worden in de computer.
- Elk type heeft een set van operatoren, die toegepast kunnen worden op waarden van het beschouwde type. De operator heeft een parameter van het beschouwde type.

Soorten types

- sytem defined (built in)
- user defined

5.2 Values vs. variables

Logisch vershil tussen values en variables

- Value : Individuele constante, een value heeft geen plaats in de tijd en ruimte. Ze kunnen wel voorgesteld worden door representaties en die hebben wel een plaats in tijd en ruimte (elke variabele kan dezelfde value hebben op een bepaald tijdstip). Een value kan niet worden geüpdated.
- Variabele : Houder van de representatie van een value. Een variabele heeft een plaats in tijd en ruimte. Variables kunnen geüpdated worden (de huidige waarde van een variabele kan vervangen worden door een nieuwe, de variabele blijft dezelfde).

Value per se vs. voorkomen van die value

We nemen een voorbeeld om dit duidelijk te maken. De integer value 3 komt maar één keer voor in de set van integers, maar verschillende variables kunnen een voorkomen van die value bevatten als hun value. Deze voorkomens kunnen bijvoorbeeld decimaal of binair opgeslagen zijn. Dus er is duidelijk ook een onderscheid tussen het voorkomen (model) van een value en de fysieke representatie van dat voorkomen (implementatie). Het is duidelijk dat gebruikers willen weten of ze twee variabelen kunnen vergelijken met een voorkomen van dezelfde value, maar ze willen niet weten of de fysieke representatie van die voorkomens overeenkomt.

Values and variables are typed

Elke value heeft een type. Een gegeven value heeft altijd exact een type.
Onderscheiden types hebben geen values gemeen (de types zijn disjoint):

- Value 3 heeft het type integer, dan zal string niet het type zijn van een value 3.
- Elke variabele heeft een type, dwz. elke mogelijke value van de variable is een waarde van dat type.
- Elk attribuut van elke relvar heeft een type, dwz. elke mogelijke waarde van het attribuut is een waarde van dat type.
- Elke operator die een resultaat teruggeeft is van een bepaald type, dwz. elke waarde die als resultaat wordt teruggegeven, is een waarde van dat type.
- Elke parameter van een operator heeft een type, dwz. elke mogelijk argument dat moet gesubstitueerd worden in de parameters heeft een waarde van dat type. Opmerken dat we read-only -en update operatoren hebben. Bij updat-operatoren moet elk argument een variabele zijn, geen waarde, van het beschouwde type. Elke expressie is van een bepaald type.
- Een *polymorphic operator* is een operator die gedefinieerd is in termen van een bepaalde parameter P en de argumenten, corresponderend met P kunnen van verschillende types zijn. Vb. operator =

5.3 Types vs. representations

Enkele opmerkingen

> De fysieke representatie van de waarden van een type moeten verborgen zijn voor de gebruiker. Zo creëren we data onafhankelijkheid. Soms worden gegevenstypes abstract genoemd (ADT) om het punt toe te lichten dat een type moet te onderscheiden zijn van zijn fysieke representatie.

Scalar vs. nonscalar types

Een type moet scalair of niet-scalair zijn :

- Een non-scalar type is een type welke zijn waarden expliciet gedefinieerd zijn om een set van gebruiker-zichtbare, direct toegankelijke componenten te hebben. In het bijzonder relatie-types, want deze hebben zowel typles als attributen als gebruiker- zichtbare componenten. Vb. ...
- Een scalar type is een type dat niet non-scalar is. Vb. QTY,Point,...

Values van het type T zijn scalar of non-scalar als T scalar is non-scalar is. Dus een non-scalar value heeft een set van gebruiker- zichtbare componenten, en scalar value niet.

Possible representation, selectors and THE operators

Definitie possible representation

> Een possible representation is een representatie van waarden van het scalar type (dus normaal niet zichtbaar voor de gebruiker). Deze bevat gebruiker-zichtbare componenten.

Voorbeeld 5.1 (pagina 116)

> Voorbeelden op possible representations.

Automatische definitie van enkele operatoren

> Declaratie van possible representation veroorzaken (door implementatie) volgende operatoren :

- Selector : deze operator staat de gebruiker toe om een waarde te selecteren of te specifieren van een bepaald type door een waarde te voorzien voor elke component van de possible representation.
- THE_operator : staat de gebruiker toe toegang te hebben tot de possible representation-componenten van de waarden van een bepaald type.
- *Voorbeeld 5.2 (pagina 117)*

  Notities :

  Selectors hebben altijd dezelfde naam als de possible representation. THE_operators hebben namen van de vorm THE_C met C de naam van de corresponderende component van de corresponderende possible representation. Selectors zijn een veralgemening van *literals*.
- *Voorbeeld 5.3 (pagina 117 tem. 119)*
- Een possible representation kan ook gedefinieerd worden in termen van user-defined types, niet enkel in termen van system-defined types.
- Zowel scalar als non-scalar types kunnen possible representation hebben.

5.4 Type defenition

Nieuwe types definiëren

Men kan nieuwe types definiëren aan de hand van het bepaalde statements of met behulp van type generators.

Voorbeeld 5.4 (pagina 119 tem. 121)

Notities :

- BNF : Backus Normal/Noun Form voor een scalar type.
- DROP : Type verwijderen indien het geen nut meer heeft. De naam van het type dat je wil verwijderen moet user-defined zijn. Hierdoor zal het type niet meer gekend zijn door het systeem. DROP zal niet gebeuren als het type nog in gebruik is.

5.5 Operators

Soorten operatoren

- *Voorbeeld 5.5 (pagina 122 tem. 124)*
- user-defined operator
- = vergelijkings operator
- > operator = *ordinal type* : type waarop > wordt uitgevoerd.
- read-only operatoren . We merken op dat alle update operatoren kunnen uitgedrukt worden in termen van :=, dit kan ook in *multiple form* := ... , := ... ;
- update operatoren
- := is de enige update operator, hiermee kan men al de rest definiëren. Ze kunnen ook gebruikt worden in multiple form.
- THE_pseudo-variable : THE_pseudovariable is een THE_operator die aan de linkerkant van de assignatie voorkomt. Deze kunnen genest worden. Ze zijn niet noodzakelijk.
- DROP : operator verwijderen , deze operator moet wel user-defined zijn.

Type concersions

Voorbeeld 5.6 (pagina 124 tem. 126)

Voorbeelden van type conversies.

Concluding remarks

De ondersteuning van operatoren heeft een aantal implicaties :

- Het systeem weet welke expressies valid zijn, en het systeem kent het type van het resultaat voor elk van de valid expressies.
- De collectie van types voor een geven databank is gesloten, dwz. dat elk type gekend is door het systeem.

5.6 Type generators

Definitie type generator

> Een type generator is nog een manier om types de definiëren. Het is een speciale operator die een type teruggeeft in plaats van een waarde. Vb. TUPLE, RELATION en array is een generated type (meestal system-defined en nonscalar).

Voorbeeld 5.7 (pagina 127 tem. 128)

> Voorbeelden op type generatoren.

5.7 SQL facilities

6 Chapter 6 : Relations

6.1 Introduction

We bespreken relatietypes, waarden en variabelen en relaties opgebouwd uit tuples.

6.2 Tuples

Precieze definitie tuple

Gegeven :

Collectie van types T_i en i=1,..,n en deze moeten niet verschillend zijn.

Definitie :

- Een tuple t bij die types is een set van geordende tripletten $< A_i, T_i, v_i >$ en noemt een component van t. Met A_i = attribuutnaam, T_i = typenaam , v_i is waarde van het type.
- n is de graad van t
- Het geordend duo $< A_i, T_i >$ is een attribuut van t.
- De heading is de set van attributen.
- het tuple type wordt bepaald door de heading.
- *voorbeeld 6.1 (pagina 142)*

Properties of tuples

Enkele eigenschappen van tuples :

- Elk tuple bevat juist één waarde voor elk van zijn attributen.
- Er is geen ordening in tuple, want sets hebben geen ordening.
- Elke subset van een tuple is op zichzelf ook een tuple (idem heading).
- Een typle met graad n is n-ary (nullary).

The tuple type generator

Voorbeeld 6.2 (pagina 143 tem. 144)

Voorbeelden mbt. de tuple generator.

Operators on tuples

Precieze definitie van tuple equality

- Tuple equality wordt als volgt gedefinieerd : $t_1 = t_2 \leftrightarrow$ ze hebben dezelfde attributen A_i en de waarde v_1 van A_i in t_1 = waarde van v_2 van A_i in t_2.
- Duplicates $\leftrightarrow t_1 = t_2$, dus alle nul-tuples zijn duplcaten van elkaar (dus het is DE nul-tuple).
- tuple types zijn geen ordinal types !

Voorbeeld 6.3 (pagina 144 tem.146)

Enkele operatoren die van toepassing zijn op tuples : project, join, wrap, unwrap ,... . < en > zijn niet toepasbaar op tuples.

Tuples types vs. possible representation

Even opmerken dat beide termen verschillend zijn, alhoewel hun definitie sterk op elkaar lijkt.

6.3 Relation types

Precieze definitie relatie

Een relatie bestaat uit twee delen :

- Heading : tuple heading , een relatie heeft dezelfde attributen en zelfde graad als de heading. Degree = aantal kolommen.

- Body : set van tuples met allemaal dezelfde heading met kardinaliteit = aantal rijen.

Definitie relatietype

Een relatie type wordt bepaald door de heading en de graad van de relatie.

Terminologie

Voorbeeld 6.4 (pagina 146 tem. 147)

- heading = schema = intension
- body = extension
- relatie bevat body, die op zijn beurt tuples bevat
- n-ary relatie = relatie van graad n
- elke subset van een heading is een heading
- elke subset van een body is een body

The relation type generator

Men voorziet ook een relatie type generator. Met elk relatietype komt een relatie selecter operator overeen. De attributen van een relatie type kunnen van verschillende types zijn.

6.4 Relation values

Eigenschappen van relation values

- Elke tuple bevat juist één waarde per attribuut. Wanneer een relatie hieraan voldoet, spreekt met van *normalized relation*. Elke relatie is in 1NF.
- Er is geen ordening bij de attributen, de heading van een relatie heeft een SET van attributen dus geen ordening, geen positie, wel een naam.
- Er is geen ordening bij tuples, de body van een relatie is een SET van tuples. Doordat er geen ordening is, is er maar 1 set van operatoren. HL kan wel ordeningsoperatoren voorzien, maar in een relationeel model bestaat dit niet.
- Er zijn geen duplicate tuples, ook weer omdat sets geen duplicaten mogen bevatten.

Relations vs. tables

Als we een relatie zien als een tabel moeten we een aantal regels van interpretatie beschouwen, vb. achter elke kolom zit een naam en een type,... . Een relatie is wat de definitie zegt dat het is, een tabel is een concrete tekening. Een relatie heeft een simpele representatie, dus gemakkelijk in gebruik.

Relation-valued attributes

Attributen kunnen relation-valued zijn, relation types kunnen gebruikt worden als basis voor het definiëren van attributen van relations. Dus we kunnen relations hebben met attributen wiens values weer relations zijn. Nesting is mogelijk.
Figuur 6.2 : Een relatie met een relation-valued attribuut

Relations with no attributes

Een relatie heeft een set attributen en een lege set is ook een set, dus een relatie zonder attributen is mogelijk, dan bestaat de heading uit een lege set van attributen.

Een relatie zonder attributen kan juist één tuple bevatten, nl. DE nul-tuple. Er zijn twee relaties met graad 0:

- TABLE_DEE : met 1 tuple YES
- TABLE_DUM : met 0 tuples NO

Operators on relations

Relational comparison

other operators

Relation type inference

Men kan gemakkelijk het type van het resultaat van een relationele expressie achterhalen door de definitie mbv types.

ORDER BY

Men kan tuples sorteren zonder $<$ *en* $>$, het is geen relationele operator, want het geeft geen relatie terug en het is geen functie, want verschillende outputs zijn mogelijk.

6.5 Relation variables

Base relvar Definition

Voorbeeld 6.5 (pg 156-157)

Updating relvars

Voorbeeld 6.6 (pg 158-160)

relvars end their interpretation

De heading heeft een predikaat, de bedoelde interpretatie.
De tuples zijn ware proposities (verkregen via het predikaat door substitutie van de argumenten van het juiste type voor de parameters van het predikaat).

Closed World Assumption : de body bevat op elk ogenblik enkel en alleen tuples die corresponderen met ware proposities op dat tijdstip.

6.6 SQL facilities

7 Chapter 7 : Relational Algebra

7.1 Introduction

Definitie relationele algebra

> De relationele algebra is een collectie van operatoren, die relaties als operand nemen en die een relatie teruggeven als resultaat.

Definitie originele algebra

> De originele algebra bestaat uit acht operatoren, verdeeld in twee groepen van vier :
>
> - De traditionele set operatoren : union, intersection, difference en Cartesisch product (zodanig opgebouwd dat ze relaties als operand hebben).
> - De speciale set operatoren : restrict (select), project, join en divide.
> - Er bestaat ook nog een primitieve set van operatoren, deze kan je gebruiken om andere instructies te schrijven : union, restrict, project, Cartesisch product en difference.
> - *Figuur 7.1 : De acht originele operatoren (pagina 174)*

Enkele opmerkingen

- Deze operatoren zijn generiek : ze zijn toepasbaar op elke specifiek relatie type
- De operatoren zijn shorthand
- Deze operatoren zijn read-only : ze updaten hun operands niet, ze zijn toepasbaar op de relation values (de huidige values van de relvar)
- Gebruik van operatoren van relvars per se vs. operatoren van relvars op de huidige waarden van de relvars

7.2 Closure revisited

Definitie relationele closure eigenschap

Deze eigenschap is het feit dat de output van een gegeven relationele operatie, op zijn beurt een relatie is. Dit is dus een gesloten geheel van operatoren. We kunnen zoveel nesten als we willen.

Opmerking bij de closure eigenschap

De closure eigenschap zegt dat elke relatie een heading moet hebben ! En bovendien moet het systeem weten wat het is. Het komt erop neer dat het resultaat een welkbekend relationeel type moet bezitten. Dus eigenlijk moeten we relationele operaties definiëren op een manier zodat het gegarandeerd is dat elke operatie een resultaat produceert met een eigen relatie type (dus met eigen attribuutnamen).

Relation type inference rules

Onderstel een type van de relatie-inputs voor gelijk welke relationele operator, dan kan daaruit het type van de output volgen. Als we dus een set hebben van zulke regels, dan zal een willekeurige expressie een resultaat produceren dat ook een welgedefinieerd type heeft en in het bijzonder een wel-gedefinieerde set van attribuut namen.

Introduceren van een nieuwe operator RENAME

Deze operator gaat attributen in een specifieke relatie hernoemen. De input is een relatie, de output ook met een gewijzigde attribuutnaam. Rename is een expressie die een bepaalde value aanduidt. Dankzij rename heeft de relationele algebra de dot-qualified attribute names niet nodig (SQL wel, vb. teams.id).

Voorbeeld 7.1 (pagina 177)

Voorbeelden van rename. S relvar is niet veranderd, RENAME is enkel een expressie.

7.3 The original algebra : syntax

Voorbeeld 7.2 (pagina 178 tem. 179)

Syntax van de verschillende operatoren.

7.4 The original algebra : semantics

Union

Definitie relationele union operator

> Gegeven twee relaties a en b van hetzelfde type, dan is de union van de twee relaties, a UNION b, een relatie van hetzelfde type, met als body tuples t, zoals ze aanwezig zijn in relatie a, relatie b of in beide relaties. We merken op dat relaties nooit duplicate tuples bevatten ! Union is gebaseerd op tuple equality (... t is equal aan een bepaalde tuple uit a of b ...)

Figuur 7.2 : union (pagina 181)

Intersect

Definitie relationele intersect operator

> Gegeven twee relaties a en b van hetzelfde type, dan is de intersection van de twee relaties, a INTERSECT b, een relatie van hetzelfde type, met als body alle tuples t zodat t voorkomt in zowel a als b (a EN b).

Figuur 7.2 : intersect (pagina 181)

Difference

Definitie relationele difference operator

> Gegeven twee relaties a en b van hetzelfde type, dan is de difference tussen de twee relaties, a MINUS b, een relatie van hetzelfde type, met als body alle tuples t, die tot relatie a behoren, maar niet tot relatie b. (zodat t behoort tot a en niet tot b)

Figuur 7.3 : difference (pagina 181)

Union, intersect en difference vereisen dat de beschoude relvars van het zelfde type zijn.

Product

Definitie product (relationeel Cartesisch Product)

> Het Cartesian product van twee relaties a en b, a TIMES b, waarbij relatie a en relatie b geen gelijke attribuutnamen hebben, is dus een relatie met een heading dat de union (niet relationeel) van de headings van relatie a en relatie b is, met een body met alle tuples t, zodat t de union (niet relationeel) is van een tuple uit relatie a en een tuple uit relatie b. Wanneer er wel twee gelijke attribuutnamen voorkomen, moeten we deze eerste hernoemen met rename. De kardinaliteit van het resultaat is het product van de kardinaliteiten van relatie a en relatie b. De graad van het resultaat is de som van de graden van relatie a en relatie b.

Figuur 7.3 : product (pagina 183)

Restrict

Definitie restrict

> Gegeven een relatie a met attributen X en Y, en θ is een operator, zodat de booleaanse uitdrukking X θ Y goedgevormd is en dat voor gegeven values voor X en Y, de uitdrukking X θ Y een ware value geeft. Dan is de θ-restiction (of gewoon restriction) van relatie a op de attributen X en Y , a WHERE X θ Y, een relatie met dezelfde heading als relatie a en met een body die bestaat uit alle tuples van relatie a waarvoor de expressie X θ Y waar geeft voor de tuple in kwestie.

Algemene definitie restrict

> Gegeven een relatie a met attributen X,Y,... en p een ware functie met parameters een set van de attributen van de relatie a, dan is de restrictie van relatie a, a WHERE p, een relatie met zelfde heading als relatie a en een body met alle tuples van relatie a zodat p waar geeft voor de tuple in kwestie.

Voorbeeld 7.3 (pagina 183 tem. 184)

Notities :

- P is een predikaat en is een restriction condition.
- Er zijn simple restriction conditions en nonsimple restriction conditions
- Restriction kan uitgedrukt worden in termen van union, intersect en minus (equivalenties).

Figuur 7.4 : restrict (pagina 184)

Project

Definitie project

Gegeven een relatie a met als attributen X,Y,... . De projection van de relatie a op X,Y,... , a {X,Y,...}, is een relatie met een heading afgeleid van de heading van de relatie a, zonder de attributen die niet vermeld waren in {X,Y,...}. De body met alle tuples, {X x ,Y y,...}, zodat deze tuples voorkomen in relatie a met X value x, Y value y, ... De projection geeft een verticale subset van een gegeven relatie, de subset werd bekomen door al de attributen die niet in de kommalijst voorkwamen en alle duplicate tuples te elimineren. We kunnen ook aangeven over welke attributen we niet willen projecteren, a {ALL BUT X,Y,...}.

Geen enkel attribuut kan meer dan een keer vernoemd worden in de kommalijst

Deze stelling klopt, omdat de attributen in een relatie allemaal verschillende namen moeten hebben.

Figuur 7.5 : projection (pagina 185)

Join

Definitie natural join

> Gegeven twee relaties a en b met attributen X_1,X_2,...,X_m,Y_1,Y_2,...,Y_n en Y_1,Y_2,...,X_n,Z_1,Z_2,...,Z_p en $X_i \neq Z_i$, voor alle i (mede dankzij operator rename) en elk attribuut Y_i heeft hetzelfde type in zowel relatie a als in relatie b. Stel nu dat X,Y en Z samengestelde attributen zijn, dan is de natural join van relatie a en relatie b, a JOIN b, een relatie met heading $\{X, Y, Z\}$ en een body met alle tuples {X x ,Y y,Z z}, zodat een tuple voorkomt in relatie a met X value x en Y value y en zodat een tuple voorkomt in relatie b met Y value y en Z value z. Joins zijn niet altijd tussen de foreign key en de primary key, maar meestal wel. Gebaseerd op tuple equality.

Figuur 7.6 : natural join (pagina 186)

Speciale joins

- Als n = 0 (relatie a en relatie b hebben geen gemeenschappelijke attributen), dan komt a JOIN b overeen met a TIMES b^2.
- Als m=p=0 (relatie a is van het zelfde type dan relatie b), dan komt a JOIN b overeen met a INTERSECT b.

Definitie θ-join

> Op basis van een vergelijkingsoperator , verschillend van de gelijkheidsoperator. Gegeven twee relaties a en b en deze hebben geen attributen gemeen. Laat relatie a een attribuut X hebben en relatie b een attribuut Y en X θ Y is een ware uitdrukking (met θ een vergelijkingsoperator), dan is de θ-join van relatie a op attribuut X met relatie b op attribuut Y, gedefinieerd als het resultaat van {(a TIMES b) WHERE X θ Y} te evalueren. Het is dus een relatie met dezelfde heading als het cartesian product van relatie a en relatie b en met een body met alle tuples t, zodat t voorkomt in dat cartesian product en dat de expressie $X\theta Y$ waar geeft voor die tuple t. Een speciaal geval is wanneer θ gelijk is aan de = operator, in dat geval noemt men de θ-join, de *equijoin*

(relatie met twee attributen bevatten wiens values gelijk zijn in elk tuple van de relatie, wanneer de ene wordt weggeprojecteerd en de andere gerenamed, dan hebben we een natural join). De θ-join is geen primitieve operator.

Figuur 7.8 : groter dan - join (pagina 187)

Divide

Definitie division

Gegeven twee relaties a en b met attributen $X_1, X_2, ..., X_m$ en $Y_1, Y_2, ..., Y_n$, waarbij de namen van X_i verschillend is van de namen van Y_j. Laat een relatie c volgende attributen bezitten : $X_1, X_2, ..., X_m, Y_1, Y_2, ..., Y_n$ (relatie c heeft een heading, gelijk aan de heading van a union b). Laat X en Y samengestalde attributen zijn, dan is de division van relatie a bij relatie b per relatie c, a DIVIDEBY b PER c, een relatie met heading $\{X\}$ en met body van alle tuples $\{Xx\}$ die voorkomen in relatie a, zodat een tuple $\{Xx, Yy\}$ in relatie c voorkomt voor alle tuples $\{Yy\}$ die voorkomen in relatie b (dwz. het resultaat bestaat uit de X values van relatie a, wiens corresponderende Y values in relatie c alle Y values van relatie b bevatten). We noemen a de dividend, b de divisor en c de mediator. Gebaseerd op tuple equality.

Figuur 7.8 : division (pagina 189)

Twee soorten divide

- Small Divide : met maar één relationele expressie
- Great Divide : met en kommalijst van relationele expressies

7.5 Examples

Voorbeeld 7.4 (pagina 190 tem. 191)

7.6 What is the algebra for?

Fundamentele bedoeling van de algebra

De fundamentele bedoeling van de algebra is om het schrijven van relationele expressies toe te laten. Deze kunnen dan leiden tot data retrieval (opvragen van data).

Mogelijke toepassingen voor relationele expressies

- retrieval
- update
- volledigheids beperkingen definiëren
- relvars definiëren
- stabiliteit
- veiligheid
- de expressies geven een high-level, symbolische representatie van de bedoeling van de gebruiker
- de expressies zijn het onderwerp van transformatieregels

De algebra geeft dus een basis voor optimalisatie.

Definitie relationeel complete taal

Een taal is relationeel compleet als ze even sterk (krachtig) is als de algebra, dus als de expressies van de taal de definitie van elke relatie toelaten en die relaties kunnen gedefinieerd worden met behulp van expressies uit de originele algebra.

7.7 Further points

De eigenschappen met betrekking tot de originele operatoren

Associativity and commutativity

- ASSOCIATIEF : union, intersect, times, join, relationeel cartesian product
- COMMUTATIEF : union, intersect, times, join, relationeel cartesian product

Some equivalences

Voorbeeld 7.5 (pagina 194 tem. 195)

Some generalizations

Definitie dyadic operatoren

Dyadic operatoren zijn operatoren die juist twee relaties als operand nemen, vb. join, union en intersect. Het is ook mogelijk dat n groter is dan 1.

Definities in geval van nul of één relatie

Stel s als een set van relaties, allemaal van het zelfde type ingeval van union en intersection.

- Als s juist één relatie r bevat, dan is de join, union en intersection van alle relaties in s, gedefinieerd door enkel r.
- Als s geen enkele relatie bevat dan :
  - de join van alle relaties in s is gedefinieerd als TABLE DEE
  - de union van alles relaties in s is gedefinieerd door een lege relatie van het zelfde type
  - de intersection van alle relaties in s is gedefinieerd door de universele relatie van hetzelfde type, dit is de unieke relatie van dat type dat alle mogelijke tuples bevat met heading H, waarbij H de heading is van hetzelfde type.

7.8 Additional operators

Voorbeeld 7.6 (pagina 196)

De syntax van de additionele operatoren.

Semijoin

Definitie semijoin

> Gegeven twee relaties a en b met attributen X_1,X_2,...,X_m,Y_1,Y_2,...,Y_n en Y_1,Y_2,...,X_n,Z_1,Z_2,...,Z_p en $X_i \neq Z_i$, voor alle i (mede dankzij operator rename) en elk attribuut Y_i heeft hetzelfde type in zowel relatie a als in relatie b. Stel nu dat X,Y en Z samengestelde attributen zijn, dan is de semijoin van relatie a en relatie b, a SEMIJOIN b, equivalent aan volgende definitie : $\{aJOINb\}\{X, Y\}$, dus het is de join van relatie a en relatie b geprojecteerd over de attributen van relatie a. De body van het resultaat zijn de tuples van relatie a die een bijbehorend deel hebben in relatie b.

Semidifference

Definitie semidiffirence

> De semidiffirence tussen relatie a en relatie b, a SEMIMINUS b, is equivalent aan volgende definitie : a MINUS $\{aSEMIJOINb\}$. De body van het resultaat zijn de tuples van relatie a die geen bijbehorend deel in b hebben.

Voorbeeld 7.7 (pagina 197)

Extend

Bedoeling van extend

> Extend neemt een relatie en geeft een andere relatie terug die identiek is aan de gegeven relatie , behalve dat deze een bijkomend attribuut bevat (values die bekomen zijn door een bepaalde berekeningsuitdrukking). Het is een expressie, dus ze kan genest worden. Deze expressie verandert de relvars niet ! Ze geeft enkel een waarde.

Figuur 7.9 : extend (pagina 197 tem. 198)

Definitie extend

> De value van de extension EXTEND a ADD exp AS Z geeft als resultaat een relatie met als heading, de heading van relatie a, uitgebreid met een attribuut Z. De body van het resultaat bestaat uit alle tuples t zodat t een tuple is van relatie a, uitgebreid met een value voor attribuut Z, berekend door het evalueren van exp op die tuple van relatie a. Het resultaat heeft eenzelfde kardinaliteit als relatie a en de graad van het resultaat is gelijk aan de graad van relatie a plus 1.

Voorbeeld 7.8 (pagina 198 tem. 199)

Definitie aggregate operators

> Deze operatoren leiden een scalaire value af van de values die voorkomen in enkele gespecifieerde attributen van een specifieke relatie (meestal een derived relation). Vb. COUNT, SUM, AVG, MAX, MIN, ALL en ANY. In geval van een empty set: COUNT,SUM (0), MAX (laagste value vh type), MIN (hoogste value vh type), ALL (true), ANY (false) en AVG (exc).

Summarize

Definitie summarize

> De value van een summarization, SUMMARIZE a PER b ADD summary AS A, is een relatie als volgt gedefinieerd :
>
> - elk attribuut van relatie b moet een attribuut zijn van relatie a, b mag Z niet bevatten.
> - de heading van het resultaat bestaat uit de heading van b, uitgebreid met het attribuut Z
> - de body van het resultaat bestaat uit alle tuples t, zodat t een tuple is van relatie b, uitgebreid met een value voor attribuut Z. De value van Z is berekend door summary te evalueren over alle tuples van relatie a, die dezelfde values hebben voor attributen $A_1, ..., A_n$ als tuple t. Het resultaat

heeft dezelfde kardinaliteit als relatie b, en de graad van het resultaat is gelijk aan de graad van relatie b plus 1. Gebaseerd op tuple equality.

Voorbeeld 7.9 (pagina 201 tem. 202)

Tclose

Definitie tclose

Gegeven binaire relatie met attributen X en Y, beide van hetzelfde type T, dan is de transitive closure van relatie a, TCLOSE a, een relatie a+ met een heading die dezelfde is van relatie a en een body die een superset is van relatie a. De tuple $\{Xx, Yy\}$ komt voor in a+ als en slechts als deze voorkomt in relatie a of als er een sequentie van values bestaat $z_1, z_2, ..., z_n$ van het type T, zodat de tuples $\{Xx, Yz_1\}, \{Xz_1, Yz_2\}, ..., \{Xz_n, Yy\}$ allemaal in relatie a voorkomen. De body van a+ bevat de body van a als subset.

7.9 Grouping and ungrouping

Definitie group en ungroup

Aangezien relaties attributen kunnen hebben die op zichzelf relaties zijn, hebben we operatoren nodig om deze te groupen en ungroupen. Deze dienen dan voor de mappping tussen relaties die zulke speciale attributen bevatten en tussen relaties die zulke speciale attributen niet bevatten. Bij grouping zijn er geen fouten mogelijk, bij ungrouping wel.

Voorbeeld 7.10 (pagina 204 tem. 206)

R GROUP $\{A1, ..., An\}$ as B
Hiervan is degree = $nR - n + 1$ met n = degree relvar R.

R UNGROUP B
Hiervan is degree = $nR + n - 1$

group, dan ungroup : OK
ungroup, dan group : NOK

Als relatie r een relatie-valued attribuut RVX heeft, dan is r ungroupable als en slechts als :

- geen enkel tuple van r een lege relatie als value heeft van RVX
- RVX is functioneel afhankelijk van de combinaties van alle andere attributen in r.

8 Chapter 8 : Relational Calculus

8.1 Introduction

Verschil tussen relationele calculus en relationele algebra

De relationele calculus is een alternatief voor de relationele algebra. Het grote verschil tussen de twee, is het volgende : De algebra voorziet een set van expliciete operatoren, die kunnen gebruikt worden om het systeem te vertellen hoe enkele gevraagde relaties te CONSTRUEREN. De calculus vooziet een notatie om de DEFINITIE van die gevraagde relatie in termen van gegeven relaties weer te geven.De gebruiker definieert karakteristieken van het gevraagde resultaat. Het systeem moet beslissen hoe de uitwerking moet gebeuren. Dus we kunnen zeggen dat de calculus beschrijfbaar is en de algebra voorschrijvend. De calculus beschrijft wat het probleem is, de algebra schrijft een procedure voor om het probleem op te lossen. De algebra is proceduraal, de calculus is non-procuderaal. Maar de algebra en de calculus zijn logisch equivalent : voor elke algebraïsche expressie, is er een equivalente calculus en voor elke calculus expressie is er een equivalente algebra. Het verschil gaat dus over de stijl : de calculus is dichter bij de natuurlijke taal, terwijl de algebra dichter bij de programmeertaal aanleunt. De calculus is gebaseerd op de predicaat calculus. Deze relationele calculus is een toegepaste vorm van de predikaat calculus, speciaal bedoeld voor relationele databank. QUEL is een taal gebaseerd op de predikaat calculus.

Voorbeeld 8.1 (pagina 213)

Fundamenteel element van de calculus

Dit fundamentele element is de range variabele, dit is een variabele die zich strekt over een speciefieke relatie. Dus een variable, wiens toegestane values, tuples van die relatie zijn. Als range variabele V ranges over relatie r, dan zal op elk gegeven ogenblik, de expressie V een bepaalde tuple van relatie r als waarde hebben. Er zal een tuple calculus nodig zijn, omdat de values van de range variabelen tuples zijn. Dus de originele relationele calculus wordt de tuple (relation) calculus. Een expressie in de tuple calculus bestaat uit een proto tuple + WHERE clause met WFF.

Defintie domain calculus

Deze calculus is een alternatieve calculus , in welke de range variables ranges over domeinen (types) in plaats van over relaties. QBE (Query-By-Example) is een domain calculus taal. De taal is relationeel compleet.

Voorbeeld 8.2 (pagina 214)

8.2 Tuple calculus

Syntax

Voorbeeld 8.3 (pagina 215 tem. 217)

Definitie WFF

Booleaanse expressies in de relationele calculus, worden meestal aangduid door WFF = well-formed-formulas.

Range variables

Voorbeeld 8.4 (pagina 217)

Enkele voorbeelden van range variable definities.

Free and bound variabel references

Elke referentie naar een range variable is ofwel vrij ofwel gebonden in een WFF

Laat V een range variable zijn en p en q zijn WFF, dan :

- Referenties tot V in NOT p zijn vrij of gebonden in die WFF als ze vrij of gebonden zijn in p. Referenties tot V in p AND q en p OR q zijn vrij of gebonden in die WFF , als ze vrij of gebonden zijn in p of q.

- Referenties tot V die vrij zijn in p, zijn gebonden in EXISTS V(p) en FORALL V(p). Andere referenties tot range variables in p zijn vrij of gebonden in EXISTS V(p) en FORALL V(p) , als ze vrij of gebonden zijn in p.

Voorbeeld 8.5 (pagina 218)

Quantifiers

Definitie FORALL en EXIST

> EXISTS en FORALL zijn twee quantifiers, EXISTS is de existentiële en FORALL is de universele quantifier. We kunnen zeggen : als p een WFF is in welke V vrij is, dan zijn EXISTS V(p) en FORALL V(p) beide valid WFF, en V is gebonden in elk van hen. De eerste zegt : Er bestaat minstens een value van V die p waar maakt. De tweede betekent : voor alle values van V, is p waar. We definiëren EXISTS formeel als een iterated OR en FORALL als een iterated AND. Beide quantifiers zijn definieerbaar in termen van elkaar : FORALL V(p) = NOT EXISTS V (NOT p).

Voorbeeld 8.6 (pagina 219 tem. 220)

Free and bound variable references revisited

Voorbeeld 8.7 (pagina 220 tem. 221)

Definitie open WFF

> Dit is een WFF die minstens één vrije referentievariable bevat. Dit wordt ook een predikaat genoemd.

Definitie gesloten WFF

> De WFF in welke alle referentievariabelen gebonden zijn. Dit wordt ook een propositie genoemd.

Relational operations

Enkele syntaxregels

- Alle referenties tot range variables in een proto tuple moeten vrij zijn in dat proto tuple.
- Een referentie tot een range variables in de WHERE clause kan vrij zijn alleen als een referentie tot diezelfde range variable (noodzakelijk vrij) voorkomt in de corresponderende proto tuple.

Voorbeeld 8.8 (pagina 221 tem. 222)

8.3 Examples

Voorbeeld 8.9 (pagina 223 tem.225)

Definitie prenex normal form

Alle quantifiers komen vooraan in de WFF.

8.4 Calculus vs. algebra

Algebra is minstens zo krachtig als de calculus

Dit kan aangetoond worden mbv. een algoritme dat een willekeurige expressie van de calculus kon herleid worden tot een semantisch equivalente expressie in de algebra.

Voorbeeld 8.10 (pagina 226 tem. 228)

Dit voorbeeld wordt gehanteerd om het algoritme op een illustratieve manier aan te tonen. Bij dit voorbeeld gebruiken we figuur 8.1.

Waarom werden de acht operatoren gedefinieerd?

De acht operatoren voorzien een doeltaal als voertuig voor een mogelijke implementatie van de calculus. De operatoren zijn ook een maatstaf voor het meten van de kracht van een willekeurig gegeven databanktaal. Een taal is relationeel compleet als het op zijn minst zo krachtig is als de calculus, dus , als elke relatie definieerbaar door een bepaalde expressie van de calculus, ook definieerbaar is door een bepaalde expressie uit die taal. Aangezien zowel algebra als calculus relationeel compleet zijn, vormen zij dus een goede basis voor talen bedoeld voor eindgebruikers. Een gegeven taal L is ook compleet als deze analoge operatoren bevat en dat de operanden van een operator uit L, gepresenteerd kan worden door expressies uit L. Voorbeelden van een complete taal is SQL en QUEL.

8.5 Computational capabilities

Voorbeeld 8.11 (pagina 230 tem.231)

8.6 SQL facilities

8.7 Domain calculus

Definitie domain calculus

De domain calculus verschilt van de tuple calculus omdat de range variables rangen over domains (ypes) in plaats van over relaties. Het syntactisch verschil is dat we de booleaanse expressie membership condition noemen.

Voorbeeld 8.12 (pagina 240 tem. 242)

8.8 Query-by-example

Bedoeling QBE

QBE is een taal gebaseerd op de domain calculus en is gebaseerd op de idee om entries te maken in lege tabellen.

Voorbeeld 8.13 (pagina 243 tem. 247)

De syntax is aantrekkelijk en simpel. Men maakt gebruik van voorbeeld elementen (rangevariabelen) en literal values. Er zijn geen quantifiers. Er is een gemakkelijke vergelijkingstest. Editeren van tabellen is mogelijk. AND en OR worden ondersteund. Alsook EXIST, maar NOT EXIST NIET (!!!), daarom is QBE niet relationeel compleet. QBE ondersteunt ook enkele aspecten van de tuple calculus. Ook grouping mogelijk en aggregate operatoren.

9 Integrity

9.1 Introduction

Definitie integrity constraint

> Dit is een booleaanse expressie die geassocieerd is met een databank en is bedoeld om altijd waar te geven. Constraints moeten formeel gedeclareerd zijn bij de DBMS en de DBMS voert ze uit. Het declareren is het gebruiken van de kenmerken van de taal, het uitvoeren van de DBMS is dat de DBMS er op toeziet dat de updates goed worden uitgevoerd in functie van de constraints, en als men ingaat tegen die constraints, moet de DBMS deze weigeren. Eens de constraint gedeclareerd is, moet het systeem er op toezien dat de databank deze beperkingen opvolgt. Wanneer het geaccepteerd wordt, wordt de constraint opgeslagen in de catalog. Je kan ook constraints droppen met het statement DROP.

Voorbeeld 9.1 (pagina 254)

9.2 A closer look

Algemene definitie integrity constraints

> Een integrity constraint is in het algemeen een constraint op de values van variabelen of op de values van een combinatie van variabelen, die toegelaten is om te veronderstellen. Het feit dat een gegeven variabele een type heeft, is dit op zich al een constraint, dus een a priori constraint. En in het algemeen : het feit dat elk attribuut van een gegeven relvar van een bepaald type is, representeert dit een a priori constraint op die relvar.

Voorbeeld 9.2 (pagina 255)

Als een bepaalde tuple voorkomt in een bepaalde relvar, dan voldoet de beschouwde tuple aan een bepaalde conditie.

Definitie logische implicatie

Een logische implicatie heeft als algemene vorm : IF p THEN q, met p (antecedent) en q (consequent) booleaanse expressies. De gehele expressie is vals als p waar is en q vals is. En de expressie is waar als p vals is en q waar.

Voorbeeld 9.3 (pagina 256 tem. 257)
IF certain tuples appear in certain relvars,
THEN those tuples satisfy a certain condition.

Tutorial D examples

Voorbeeld 9.4 (pagina 257 tem. 258)

9.3 Predicates and propositions

Voorbeeld 9.5 (pagina 258)

Notities :

Dit voorbeeld is een booleaanse expressie. Er komt een variabele in voor, nl relvar S. We kunnen dus niet zeggen wat de value van de expressie is, totdat we een value substitueren in die variable. Deze expressie is dus een predikaat en de variabele S is een parameter tot dat predikaat. Wanneer we de constraint willen checken, moeten we een argument voorzien die de huidige value is van relvar S, zodat de expressie kan geëvalueerd worden. Wanneer we dit gedaan hebben, bekomen we dus een waar of valse expressie, die geen variables meer bevat, dit noemt men een propositie. Dus wanneer een constraint wordt gedefinieerd, is het een predikaat, wanneer de constraint wordt gecheckt en er wordt gesubstitueerd, dan spreken we van een propositie.

9.4 Relvar predicates and database predicates

Definitie predikaat relvar

Een gegeven relvar R is het onderwerp van verschillende constraints, dan zal DE relvar predikaat voor R de logische AND zijn, van alle constraints die toepasbaar zijn op relvar R.

Eerste versie van De Gouden Regel

Geen enkele update operator mag ooit een value toekennen aan een relvar dat er voor zorgt dat zijn relvar predikaat vals wordt.

Definitie databank predikaat

Gegeven databank D en laat D enkele relvars bezitten, met bijbehorende predikaat relvars, dan is de databank predikaat gelijk aan de conjunctie van al de relvar predikaten.

Tweede algemene versie van De Gouden Regel

Geen enkele update operator mag ooit een value toekennen aan een databank dat er voor zorgt dat zjn databank predikaat vals geeft.

9.5 Checking the constraints

Checken van constraints : implementatie

Voorbeeld 9.6 (pagina 260)
Bij de implementatie gaat men de formele expressie van de constraint gebruiken om de gepaste checks, die moeten worden uitgevoerd op tuples, daaruit te impliceren, vooraleer de invoeging te doen.

Checken van constraints : model

De Gouden Regel impliceert dat al het checken onmiddellijk gebeurt, omdat deze regel spreekt in termen van updateoperatoren, en niet in termen van transacties.

9.6 Internal vs. external predicates

Definitie internal predicates

> De relvar predikaten en databank predikaten zijn deel van de databank definitie (formeel gekend) en deze predikaten worden uitgevoerd door het systeem. Daarom spreken we van interne predikaten. Interne predikaten is wat de data betekent voor het systeem. Het systeem zelf kan enkel deze predikaten verstaan. De gebruikers kunnen beide predikaten verstaan.

Definitie external predicates

> De externe predikaten is wat da data betekent voor de gebruikers.

Voorbeeld 9.7 (pagina 263)

> **Notities :**
> Een gegeven tuple komt voor in een gegeven relvar op een gegeven moment als en slechts als die tuple de relvar zijn externe predikaat waar maakt op dat moment. Hier uit volgt dat een gegeven relvar alle en enkel die tuples bevat die corresponderen met de ware instanties van de relvar zijn externe predikaat op dat moment. Externe en interne predikaten komen niet altijd overeen.

9.7 Correctness vs. consistency

Extern predikaat

> Het extern predikaat van een gegeven relvar is de bedoelde interpretatie voor die relvar. Het is belangrijk voor de gebruiker, niet voor het systeem. Het interne predikaat is dus eigenlijk het criterium voor het accepteren van updates op de relvar. Het systeem kan het extern predikaat niet kennen van een gegeven relvar. Het systeem weet wel de corresponderende interne predikaat en dat is hetgene het systeem zal uitvoeren. Het systeem kan niet de waarheid uitvoeren, wel de consistentie. Een tuple kan voldoen aan het interne predikaat en toch niet in de databank te zitten. Closed World Assumption is niet van toepassing bij interne predikaten.

Het systeem kan de waarheid niet uitvoeren, de consistentie wel

Het systeem kan niet garanderen dat de databank enkel ware proposities bevat, het kan enkel voorkomen dat de databank data bevat die niet aan de constraints voldoet.

Enkele stellingen

- Als de databank enkel ware proposities bevat, dan is ze consistent, maar het omgekeerde is niet noodzakelijk waar.
  Correctheid impliceert consistentie.
- Als de databank inconsistent is, dan zal het op zijn minst in valse propositie bevatten, maar het omgekeerde is niet noodzakelijk waar.
  Inconsistentie impliceert incorrectheid.
- We kunnen zeggen : Correctheid impliceert consistentie (niet omgekeerd) en inconsistentie impliceert incorrectheid (niet omgekeerd). En correct wil zeggen dat alles overeenkomt met de reële wereld.

9.8 Integrity and views

Constraints vs. views

Constraints hebben betrekking op algemene relvars, dus ook op views. Deze hebben ook relvar predikaten, interne en externe.

Voorbeeld 9.8 (pagina 265 tem. 266)

9.9 A constraint classification scheme

Soorten constraints

- Een databank constraint
- Een relvar constraint
- Een attribuut constraint
- Een type constraint

Type constraints

Een type constraint is een specificatie van de values die het type in kwestie opmaken. Type constraints kunnen gechecked worden tijdens de uitvoering van een selector invocatie. Type constraints worden altijd onmiddellijk gechecked en zodanig dat geen enkele relvar ooit een value kan bekomen voor een attribuut in een tuple die niet van het juiste type is. Type constraints worden gedropt, door het type zelf te droppen.

Voorbeeld 9.9 (pagina 266 tem. 267)

Attribute constraints

Een attribuut constraint noemen we een a priori constraint, maw. het is juist een declaratie dat een specifiek attribuut van een specieke relvar een specifiek type heeft. Attribuut constraints kunnen gedropt worden, door het attribuut zelf te droppen (dus eigenlijk ook de relvar).

Voorbeeld 9.10 (pagina 267)

Relvar and database constraints

Een relvar constraint betrekt juist een relvar, terwijl een databank constraint twee of meer relvars betrekt. Relvar en databank constraints kunnen *transition constraint* zijn. Dit is een constraint op de legale transitie dat een gegeven variable (in het bijzonder een relvar of databank) kan maken van de ene value naar de andere. Transition constraint wordt niet toegepast bij type of attribuut constraints.

Voorbeeld 9.11 (pagina 268)

9.10 Keys

Candidate keys

Laat R een relvar zijn, bij definitie heeft de set van alle attributen van R de unieke eigenschap dat op elk moment, geen twee tuples in de value van R duplicaat kan zijn van een ander. Meestal heeft een subset van de set van alle attributen van R ook die unieke eigenschap. Elke relvar heeft minstens één kandidaatsleutel.

Definitie Kandidaatsleutel

> Laat K een subset zijn van de attributen van relvar R. Dan is K de kandidaatsleutel voor R als en slechts als het de twee volgende eigenschappen bezit :
>
> - Uniek : geen legale value van R bevat ooit twee verschillende tuples met de zelfde value voor K.
> - irreduciebel : geen eigen subset van K is uniek op zichzelf.

Irriduciebel en minimalistisch zijn verschillend

> Deze twee termen zijn verschillend, want als je zegt dat de kandidaatsleutel K_1 minimaal is, wil dat niet zeggen dat een andere kandidaatsleutel K_2 niet kan gevonden worden met minder componenten.

Voorbeeld 9.12 (pagina 270)

> **Notities :**
>
> Een kandidaatsleutel kan een samengestelde kandidaatsleutel zijn, indien deze twee of meer attributen bevat. Een simpele kandidaatsleutel is een kandidaatsleutel die niet samengesteld is. Kandidaatsleutels worden voor veel redenen gebruikt : ten eerste is het een shorthand voor een bepaalde relvar constraint en de kandidaatsleutels voorzien een basis *tuple -level addressing mechanism* in het relationele model. Dwz. De enige systeemgegarandeerde weg om een bepaalde tuple vast te leggen is door een kandidaat sleutel value. Kandidaatsleutels zijn dus fundamenteel voor de succesvolle operatie van een relationeel systeem.
>
> *Voorbeeld 9.13 (pagina 270)*

Enkele punten

- Meerdere CKs zijn mogelijk.
- Zowel base relvars als views hebben kandidaatsleutels.
- Een superkey van een CK is een superset van een kandidaatsleutel, deze heeft een unieke eigenschap, maar niet noodzakelijk de irriduciebele eigenschap. Een kandidaatsleutel is een speciaal geval van de superkey.
- Als SK een superkey is voor relvar R en A is een attribuut van R, dan houdt de functional dependency SK→A noodzakelijk R in.
- Er is niet noodzakelijk een index op de kandidaatsleutel.

Primary keys and alternate keys

Indien een relvar twee of meer kandidaatsleutels heeft, dan moet juist één van die sleutels een primary key zijn en de andere worden alternate keys genoemd. Als er maar één kandidaatsleutel is, dan wordt deze meteen de primary key, dus elke relvar heeft steeds één primary key.

Voorbeeld 9.14 (pagina 271)

Foreign keys

Een vreemde sleutel is de set van attributen van een relvar R_2 wiens values moeten matchen met values van een bepaalde kandidaatsleutel van een bepaalde relvar R_1.
Voorbeeld 9.15 (pagina 272)

Definitie vreemde sleutel

> Laat R_2 een relvar zijn. Dan is een vreemde sleutel in R_2 een set van attributen van R_2, FK genoemd zo dat :
>
> - er een relvar R_1 bestaat met een kandidaat sleutel CK (R_1 en R_2 niet noodzakelijk verschillend).

- het mogelijk is om een subset van de attributen van FK te hernoemen, zodat FK FK_{bis} wordt en FK_{bis} en CK zijn van het zelfde tuple type.
- elke value van FK in de huidige value van R_2 geeft een value voor FK_{bis} die identiek is aan de value van CK in een bepaalde tuple in de huidige value van R_1.

Enkele punten

- Meestal moet men geen hernoeming doen. We veronderstellen dus FK en FK_bis identiek.
- Elke value van FK moet voorkomen als value van CK, het omgekeerde is niet nodig.
- FK is simpel of samengesteld naargelang CK is simpel of samengesteld.
- Een FK value representeert een referentie voor de tuple die de gematchte CK value bevat (referenced tuple). *referential constraint* is de constraint dat values van FK moeten matchen met values van CK. Daarbij is R_1 de referenced relvar en R_2 de referencing. *referential integrity* is het systeem die er op toeziet dat de databank geen foute FK bevat.
- *Voorbeeld 9.16 (pagina 273 tem. 274)*
  Voorbeeld over *referential diagram* (voorstelling van referential constraints).
- Een gegeven relvar kan zowel referenced als referencing zijn. Hiermee kan een referentiepad gevormd worden.
- CK en FK moeten niet verschillend zijn, men spreekt van self-refencing, hieruit kunnen referenie cirkels ontstaan (als er een referentiepad bestaat van Rn naar Rn).
- referential integrity : de databank mag geen unmatched FK values bevatten.
- FK als shorthand voor databank constraint of meer (zie referential actions)

Referential actions

Beschouwing :
Stel dat men een tuple wil deleten uit de suplier relvar waarvan het Id = S1. Dit wordt gedaan. Maar de databank bevat ook een relvar met shipment (met shipments voor S1) en deze worden niet verwijderd.
Wanneer het systeem de referential constraint nakijkt van shipments naar suppliers, dan is er een fout, dus er wordt een exceptie gegenereerd.

Bedoeling :
Men kan zorgen voor een compenserende actie, die garandeert dat het resultaat de voldoet aan de constraint.

Mogelijkheden :

- ON DELETE CASCADE : het systeem verwijdert ook alle shipments voor S1. Bij toevoeging van deze regel bij de definitie van de FK, wordt er dus een delete regel bijgevoegd. En CASCADE is de referential action voor die regel.
- DELETE oparties zijn restricted (RESTRICT als referential action) (beperkt) tot het geval waar er geen matching shipments zijn. vb. tuple wel in referenced relvar, maar in geen enkele andere referencing relvar, dan is er geen probleem om te deleten.
- NO ACTION : (gevaar voor fouten)

Enkele punten :

- we hebben ook een update rule nodig (zelfde principes).
- andere referential actions zijn mogelijk
- bij referential pad kunnen DELETE regels falen, in dat geval moet alles falen.
- database updates zijn altijd atomic

9.11 Triggers (a digression)

9.12 SQL facilities

10 Views

10.1 Introduction

Definitie view

> In het relationele model is view een genoemde expressie in de relationele algebra. Een view-defining expressie wordt niet geëvalueerd door het systeem, maar wel onthouden (wordt bijgehouden in de cataloog). We merken nog op dat het substitutie proces (het proces die de view-definging expressie substitueerd voor de view-naam) voornamelijk werkt door de relationele closure. Closure impliceert dat waar een simpele relvar R kan voorkomen in een expressie, dan kan daar een relationele expressie in de plaats voorkomen (als de expressie van hetzelfde type is). Relaties zijn gesloten onder de relationele algebra. Updates bij de onderliggende data , zijn direct zichtbaar, updates op de view zal direct en autmatisch toegepast worden op de onderliggende data. Gebruiker kan werken met de view alsof het een base relvar is.

Voorbeeld 10.1 (pagina 295 tem. 297)

> Dit voorbeeld maakt gebruik van figuur 10.1.

Further examples

Voorbeeld 10.2 (pagina 297)

Defining and dropping views

We kunnen view definiëren en droppen. We merken op dat view defenities het external schema combineren met de external/conceptual mapping, omdat zij zowel specifiëren wat het external object eruit ziet en hoe dat objecten gemapt worden in het conceptual level (de onderliggende relvars). Sommige views ondersteunen de external/external mapping (wanneer een view gedefinieerd wordt in termen van een andere view). Een view droppen, faalt als er nog een andere view naar refereert.

Voorbeeld 10.3 (pagina 298)

10.2 What are views for?

Enkele redenen

- views als korte schrijfwijze, macro
- views staan toe dat zelfde data door verschillende gebruikers op verschillende manieren kan gezien worden op het zelfde tijdstip
- views voorzien automatische veiligheid voor verborgen data : Verborgen data is data die niet zichtbaar is door een bepaalde view, deze data is dus veilig voor toegang in die view.
- *Voorbeeld 10.4 (pagina 298 tem. 299)*
- views kunnen logische data onafhankelijkheid voorzien

Logical data independence

Logische data onafhankelijkheid kan gedefinieerd worden als de immuniteit van gebruikers en gebruikersprogramma's tot veranderingen in de logische structuur van de databank (conceptual level). En views zorgen voor deze onafhankelijkheid.

Twee aspecten

- groei
- herstructureren

Groei

Als de databank groeit , moet de databank definitie ook groeien. Er zijn twee soorten van groei :

- De uitbreiding van een bestaande base relvar
- Het onstaan van een nieuwe base relvar

Geen enkele van deze veranderingen hebben effect op de gebruikers of gebruikerprogramma's.

Herstructureren

De logische plaats van de informatie verandert, maar de gehele informatie blijft onderanderd. De versie voor en na de herstructurering moeten informatie - equivalent zijn. De twee versies moeten informatie-equivalent zijn.
Voorbeeld 10.5 (pagina 300)

Two important principles

Twee bevindingen :

- Een gebruiker die een view V definieert, dus die bewust is van de corresponderende view-defining expressie X, kan de naam V gebruiken overal waar X bedoeld wordt.
- Een gebruiker die enkel geïnformeerd is dat V bestaat en klaar is voor gebruik, is niet bewust van X.

Er moet geen onnoodzakelijk verschil zijn tussen base en derived relvars : *The principle of interchangeability.* We zouden views moeten kunnen updaten, de databank moet niet afhankelijk zijn van de beslissing of de relvar base of derived is.

10.3 View retrievals

relationele expressie als functie

Elke relationele expressie kan gezien worden als een relation-valued functie. Laat D een databank zijn en V een view op D (een view wiens definitie expressie X een functie is van D), dan V = X (D). Laat RO een retrieval operatie zijn op V , dan is het resultaat van de retrieval : RO (V) = RO ($X(D)$) . Dus het resultaat van de retrieval is gelijk aan her resultaat als je X toepast op D (materializing een copy van de relatie, die de huidige value is van view V, en dan RO toepassen op die copy).

Een issue over materializing

Materializing kan niet gebruikt worden bij update operaties, want het hele punt van een view up te daten, is juist dat de update zou gebeuren op de gehele onderliggende data, niet juist op een copy van die data.

10.4 View updates

Views zijn updatable bij definitie, want het zijn relvars.

relationele expressie als functie

> Laat D een databank zijn en laat V een view zijn op D, dan V = X (D). Laat nu UO een update operatie zijn op V. UO kan gezien worden als een operatie die het effect heeft zijn argument te veranderen : UO (V) = UO $(X(D))$. We moeten dus nog een UO_bis vinden op D zodat UO $(X(D))$ = X $(UO_{bis}(D))$.
> Alle views zijn updatable, met uitsluiting van diegene die niet aan de integrity constraints voldoen.
> Updates kunnen niet direct geïmplementeerd worden in termen van views per se, want D is het enige dat echt bestaat.

The golden rule revisited

Geen enkele update operatie mag ooit een value toekennen aan een relvar, die zorgt dat de relvarpredikaat vals maakt.

Views hebben ook predikaten (door het Principle of Interchangeability moet het systeem die predikaten kennen). Er moet een set van predicate inference rules worden ingevoerd, zodat als we weten wat de predikaten voor de input zijn van een relationele operatie, dan kunnen we daaruit de predikaten voor de output afleiden.

Voorbeeld 10.6 (pagina 304)

> Dit is een voorbeeld van een view predikaat in het geval van view C = A intersect B, met A en B relvars van hetzelfde type.
> Views erven automatisch bepaalde constraints van relvars waarvan ze afgeleid zijn.

Toward a view-updating mechanism

Enkele principes

> *Voorbeeld 10.7 (pagina 305 tem. 307)*

- The Golden Rule !!
- Het gaat om semantiek, niet om syntactische begrippen.
- Een manier van updaten voor alle relvars.
- Update kan gezien worden als DELETE-INSERT.
- Het predikaat wordt niet gecheckt in het midden van de update, dus het is DELETE-INSERT-check. Het kan zijn dat na de DELETE , er niet voldaan is aan het predikaat, maar de update in het geheel niet, dus er worden geen cascades gedaan tussendoor.
- Recursive applicaties (views van views van ...)
- De updates zijn alles of niets.

We stellen in wat volgt dat A en B relationele expressies zijn, van hetzelfde type en dat PA en PB de predikaten zijn.

Union

Insert regel voor A union B

De nieuwe tuple moet aan PA of PB of beide voldoen. Als het voldoet aan PA, dan is het ingevoegd in A (dit kan als neveneffect hebben, dat het in B ook wordt ingevoegd). Als het aan PB voldoet, wordt ht ingevoegd in B (enkel als het nog niet in B is ingevoegd).
Voorbeeld 10.8 (pagina 307 tem 308)

Delete regel voor A union B

Als de te deleten tuple voorkomt in A, dan wordt het gedeleted uit A (met als neveneffect , dat het kan gedeleted worden uit B). Als de tuple zich nog steeds in B bevindt, wordt hij daar ook gedeleted.
Voorbeeld 10.9 (pagina 308)

update regel voor A union B

De tuple die moet worden geupdated, moet van die aard zijn dat de geupdatede versie aan PA , PB of beide voldoet. Als de tuple die moet geupdated worden, zich in A bevindt, wordt het gedeleted van A, zonder het predikaat van A te controleren. (Dit kan als neveneffect hebben, dat het ook uit B wordt verwijderd). Als de tuple zich nog in B bevindt, wordt deze ook verwijderd. Als de geupdatede versie van de tuple voldoet aan PA, dan wordt deze ingevoegd in A. Als de geupdatede versie van de tuple voldoet aan PB, wordt deze hier ook ingevoegd, indien dit nog niet had voorgedaan.
Voorbeeld 10.10 (pagina 309)
voorbeeld van tuples die overgaan van de ene relvar naar de andere.

Intersect

Het predikaat voor A INTERSECT B is (PA) AND (PB).

Insert regel voor A intersect B

De nieuwe tuple moet aan zowel PA als PB voldoen. Als het zich momenteel niet in A bevindt, wordt het ingevoegd in A (eventueel met neveneffect dat het in B wordt ingevoegd). Als het zich nog niet in B bevindt, wordt het ook in B ingevoegd.
Voorbeeld 10.11 (pagina 308)

Delete regel voor A intersect B

De tuple die moet verwijderd worden, wordt verwijderd uit A (met neveneffect dat het ook uit B wordt verwijderd). Als het zich dan nog in B bevindt, wordt het nog uit B verwijderd.
Voorbeeld 10.12 (pagina 308)

update regel voor A intersect B

De tuple die moet worden geupdated, moet van die aard zijn dat de geupdatede versie aan PA en PB voldoet. De tuple wordt verwijderd uit A. (met eventueel neveneffect dat het wordt verwijderd uit B). Als het zich nog in B bevindt, wordt het daar ook verwijderd. Als de geupdatede versie van de tuple zich niet in A

bevindt, wordt het ingevoegd in A (met neveneffect dat het wordt ingevoegd in B). Als het dan nog niet in B voorkomt, wordt het daar ook ingevoegd.

Difference

De predikaten van A MINUS B zijn (PA) AND NOT (PB).

Insert regel voor A minus B

De nieuwe tuple moet aan PA en niet aan PB voldoen. Het wordt ingevoegd in A.

Delete regel voor A minus B

De tuple die moet gedeleted worden, wordt verwijderd uit A.

update regel voor A minus B

De tuple die moet geupdated worden, moet van die aard zijn dat de geupdatede versie voldoet aan PA en niet aan PB. De tuple wordt verwijderd uit A en de geupdatede versie wordt ingevoegd in A.

Restrict

De defining expressie van de view V is A WHERE p, en het predikaat van A is PA. Dan is het predikaat van V gelijk aan (PA) AND (p).

Insert regel voor A where p

De nieuwe tuple moet voldoen aan PA en p. Het wordt ingevoegd in A.

Delete regel voor A where p

De tuple die moet verwijderd worden, wordt verwijderd uit A.

update regel voor A where p

De tuple die moet geupdated worden moet van die aard zijn dat de geupdatede versie voldoet aan zowel PA en p. De tuple wordt verwijderd uit A. De geupdatede versie wordt ingevoegd in A.

Voorbeeld 10.13 (pagina 310)

Dit voorbeeld bevat verschillende update-situaties en gebruikt hierbij figuur 10.4.

Project

Laat de attributen van A (met predikaat PA) ingedeeld worden in twee disjunte groepen, X en Y. We beschouwen de projectie van A over X : A$\{X\}$. Laat (x) een tuple zijn van die projectie. Dan is het predikaat voor die projectie : Voor alle x, bestaat er een y van Y values, zodat de tuple (x, y) voldoet aan PA.

Insert regel voor A $\{X\}$

Laat (x) de te invoegen tuple zijn. Laat de default waarde van Y y zijn. De tuple (x, y) wordt in A ingevoegd. Kandidaatsleutels hebben geen default, dus een projectie die niet alle kandidaatsleutels van de onderliggende relvar bevatten, laat geen INSERT toe.
Er zijn default waarden nodig bij views. De CKs hebben geen default waarde, dus als er projecties zijn zonder CKs, dan wordt er geen insert toegelaten.

Delete regel voor A $\{X\}$

Alle tuples van A met dezelfde X value als de tuple die moet worden verwijderd, worden verwijderd uit A. Het is handig als X minstens een kandidaatsleutel bevat, zodat de tuple die moet verwijderd worden uit A$\{X\}$ correspondeerd met juit een tuple van A.

update regel voor A $\{X\}$

Stel (x) de te updaten tuple en laat de geupdatede versie (x_{bis}) zijn. Laat a een tuple zijn van A met dezelfde value X x, en laat de value van Y in A y zijn. Zulke tuples worden verwijderd uit A, daarna voor zulke waarde y, wordt de tuple (x_{bis}, y) ingevoegd in A. Hier is y geen default waarde!

Voorbeeld 10.14 (pagina 312)

Dit voorbeeld bevat verschillende update-situaties en gebruikt hierbij figuur 10.5.

Extend

Laat de view-defining expressie V gelijk zijn aan EXTEND A ADD exp AS X, met PA het predikaat voor A. Dan is het predikaat PE voor V gelijk aan : PA (a) AND e.X = exp (a). Met e een tuple van V en a een tuple dat overblijft wanneer de X component van e verwijderd is.

Insert regel

Laat e de te invoegen tuple zijn. e moet voldoen aan PE. De tuple a dat afgeleid is van e door X weg te projecteren , wordt in A ingevoegd.

Delete regel

Laat de te verwijderen tuple e zijn. De tuple a die afgeleid is van e door X weg te projecteren , wordt verwijderd van A.

update regel

Laat de te updaten tuple e zijn en laat de geupdatede versie e_bis zijn. e_bis moet voldoen aan PE. De tuple a , die afgeleid is van e door X weg te projecteren, wordt verwijderd van A. De tuple a_bis, die afgeleid is van e_bis door X weg te projecteren, wordt in A ingevoegd.

Voorbeeld 10.15 (pagina 313)

Dit voorbeeld bevat verschillende update-situaties en gebruikt hierbij figuur 10.6.

Join

Joins zijn altijd updatable.

Vroeger was men van oordeel als men enen tuple deletete uit een join, dan was het geen join meer. Zelfs met een base relvar is het niet altijd mogelijk om één tuple te deleten. Onderliggende relvars updaten kan neveneffecten hebben voor de view (het kan zijn dat de view niet meer aan zijn predikaat voldoet).

Enkele puntjes

- Het is gewoonlijk verondersteld dat het altijd mogelijk, is om een individuele tuple van een base relvar te updaten onafhankelijk van alle andere tuples van die base relvar.
- Op hetzelfde moment, wordt er gerealiseerd dat het niet altijd mogelijk is om een individuele tuple van een view te updaten onafhankelijk van alle andere tuples van die view.

Laat J = A JOIN B een join zijn, met A, B en J met de volgende headings : $\{X, Y\}$, $\{Y, Z\}$ en $\{X, Y, Z\}$ respectievelijk. Laat de predikaten voor A en B PA en PB zijn. Dan is ht predikaat voor J PJ gelijk aan : PA (a) AND PB (b). En hierbij is a de tuple die afgeleid is van de gegeven tuple j van de join, door Z weg te projecteren. En b is de tuple afgeleid van j, door X weg te projecteren.

Insert regel

> De nieuwe tuple j moet aan PJ voldoen. Als de A portie van j niet in A voorkomt, wordt het in A ingevoegd. Als de B portie van j niet in B voorkomt, wordt het ingevoegd in B.

Delete regel

> De A portie van de tuple die moet verwijderd worden, is verwijderd uit A en de B portie is verwijderd uit B.

update regel

De tuple die moet geupdated worden, moet van die aard zijn dat de geupdatede versie voldoet aan PJ. De A portie is verwijderd van A, en de B portie is verwijderd van B. Als dan de A portie van de geupdatede versie van de tuple niet voorkomt in A, wordt deze ingevoegd in A. Als de B portie niet in B voorkomt, wordt deze ingevoegd in B.

Verschillende implicaties van deze regels

Case 1 : one to one

We merken eerst op dat we deze case best vermelden als : $(zero - or - one) - to - (zero - or - one)$
Er is een integrity constraint die verzekert dat voor elke tuple van A er minstens één matching tuple moet zijn in B en vice versa. Dit impliceert dat de set van attributen Y over welke de join genomen wordt, een superkey moet zijn voor zowel A als B.

Voorbeeld 10.16 (pagina 315)

Case 2 : one to many

We merken op dat we deze case best vermelden als : $(zero - or - one) - to - (zero - or - more)$
Er is een integrity constraint dat verzekert dat voor elke tuple van B er minstens een matching tuple is van A. De set van attributen Y, over welke de join genomen wordt, bevat een subset K, zodat K een kandidaatsleutel is voor A en een matching foreign key voor B. $(zero - or - one)$ wordt dan juist een.

Voorbeeld 10.17 (pagina 315 tem 317)

Dit is een voorbeeld mbv figuur 10.7.

Case 3 : many to many

We merken op dat deze case als volgt kan vermeld worden : $(zero - or - more) - to - (zero - or - more)$.
Er is geen integrity constraint om te verzekeren dat we

gebruik maken van Case 1 of Case 2.

Voorbeeld 10.18 (pagina 317 tem. 318)

Dit is een voorbeeld mbv. figuur 10.8.

Other operators

- Rename
- Cartesian product
- Summarize
- Group en ungroup
- Tclose

10.5 Snapshots (a digression)

10.6 SQL facilities

Part 3 : DATABASE DESIGN

11 Chapter 11 : Functional Dependencies

11.1 Introduction

Definitie functionele afhankelijkheid

> Een functionele afhankelijkheid (FD) is eigenlijk een many-to-one relatie tussen een set van attributen en een andere set van attributen binnen een gegeven relvar.
> *Voorbeeld 11.1 (pagina 333 tem. 334)*

11.2 Basic Defenitions

Twee belangrijke verschillen

- CASE a : de value van een gegeven relvar op een bepaald moment (relatie).
- CASE b : set van alle mogelijke values die de gegeven relvar mag veronderstellen op verschillende tijdstippen (base relvar).

CASE a : functional dependence

> Laat r een relatie zijn, en laat X en Y willekeurige subsets zijn van de set van attributen van r. We zeggen dat Y functioneel afhangt van X (X legt Y functioneel vast), X $\rightarrow$ Y, als en slechts als elke X value in r geassocieerd is met juist één Y value in r. Dus telkens wanneer twee tuples van r overeenkomen op hun X value, dan komen ze ook overeen met hun Y value. X noemen we de *determinant* en Y is de *dependent*, deze zijn beide sets van attributen.
> *Voorbeeld 11.2 (pagina 335 tem. 336)*

CASE b : functional dependence

> Laat R een relvar zijn, en laat X en Y willekeurige subsets zijn van de set van attributen van R. We zegen dat Y functioneel afhangt van X, X $\rightarrow$ Y, als en slechts als in elke mogelijk legale value van R, elke X value geassocieerd is met juist één Y value. En elke mogelijk legale value van R, wanneer twee tuples overeenkomen op hun X value, dan komen ze ook overeen op hun Y value.
> *Voorbeeld 11.3 (pagina 336 tem. 337)*

Speciaal geval

We merken op dat als X een kandidaat sleutel is van relvar R, dan moeten alle attributen Y van relvar R functioneel afhankelijk zijn van X. Als relvar R voldoet aan A → B en A is geen kandidaatsleutel, dan bevat R een overtolligheid.

11.3 Trivial and nontrivial dependencies

Definitie triviale afhankelijkheid

Een afhankelijkheid is triviaal als en slechts als de rechterzijde een subset is van van de linkerzijde. Deze triviale afhankelijkheden moeten we elimineren om de grootte van de set van FD te verkleinen.

11.4 Closure of a set of dependencies

Implicaties bij FD

Sommige FD kunnen andere impliceren. Stel dat we een relvar R hebben met attributen A, B en C zodat we enkele FD hebben : A → B en B → C die bestaan voor R. Dan is het eenvoudig te zien dat de FD A → C ook bestaat voor R.

Definitie closure voor een set S van FD

De set van alle FD , die geïmpliceerd zijn door een gegeven set S van FD, wordt de closure van S genoemd en genoteerd als S_+.

Inference rules om van S S_+ te bekomen : Armstrongs rules

Laat A, B en C willekeurige subsets zijn van de set van attributen van relvar R en AB is A union B.

- Reflexief (reflexivity) : Als B een subset is van A, dan A → B.
- Toename (augmentation) : Als A → B, dan AC → BC.
- Transitief (transitivity) : Als A → B, en B → C, dan A → C.

Deze regels zijn compleet, want we kunnen ze rechtstreeks bewijzen uit de definitie van funtionele afhankelijkheid : gegeven een set S van FD, en alle FD geïmpliceerd door S kunnen afgeleid worden van S door deze regels te gebruiken. Deze regels zijn ook sound : geen enkele additionele FD (FD niet geïmpliceerd door S) kan op die manier afgeleid worden.
Verdere regels kunnen afgeleid worden uit de vorige :

- Zelf-determinerend (self-determination) : A → A.
- Decompositie (decomposition) : Als A → BC, dan A → B en A → C.
- Unie (union) : Als A → B en A → C, dan A → BC.
- Compositie (composition) : Als A → B en C → D, dan AC → BD.
- Algemene unificatie theorema (general unification theorem) : als A → B en C → D, dan $A \bigcup (C - B) \rightarrow BD$
- *Voorbeeld 11.4 (pagina 339)*

11.5 Closure of a set of attributes

Closure van Z_+ van Z onder S

Gegeven een relvar R, een set Z van attributen van R en een set S van FD die bestaan voor R, dan kunnen we de set van alle attributen vastleggen van R die functioneel afhankelijk zijn van Z. Dit kan door middel van een algoritme.
Voorbeeld 11.5 (pagina 340 tem. 341)

Enkele opmerkingen

- Gegeven een set S van FD, dan kunnen we vertellen of een specifieke FD X → Y volgt uit S, omdat de FD zal volgen, als en slechts als Y een subset is van de closure X_+ van X onder S. We kunnen nu vastleggen of een gegeven FD X → Y zich bevindt in de closure S_+ van S, zonder de closure S_+ uit te zoeken.
- We weten dat een superkey voor een relvar R een set van attributen is voor R, die een kandidaatsleutel omvat van R als een subset. Uit deze definitie volgt dat de superkeys

voor een gegeven relvar R precies die subsets K zijn van de attributen van R zodat de FD K $\rightarrow$ A waar is voor elk attribuut A van R. K is dus de superkey als en slechts als de closure K_+ van K, precies de set is van alle attributen van R. En K is de kandidaatsleutel als en slechts als het een irrudicibele superkey is.

- Gegeven Z een set van attributen van relvar R en S een set van FDs die bij R passen, dan is de set van FDs van R, met Z aan de linkse kant de set van alle FDs van de vorm Z $\rightarrow$ Z', waarbij Z' een subset is van de closure Z_+ van Z onder S.

11.6 Irreducible sets of dependencies

S_2 is een cover voor S_1

Laat S_1 en S_2 twee sets van FD zijn. Als elke FD geïmpliceerd door S_1, geïmpliceerd is door S_2, dus als S_1+ een subset S_2+, dan zeggen we dat S_2 en cover is van S_1. Dit betekent wanneer de DBMS de FD van S_2 uitvoert, dan zullen de FD van S_1 ook uitgevoerd worden.

S_1 en S_2 zijn equivalent

Als S_2 een cover is van S_1 en S_1 is een cover voor S_2, dus $S_1+ = S_2+$, dan zeggen we dat S_1 en S_2 equivalent zijn. Dus als dit voldaan is, en als de DBMS de FD van S_2 uitvoeren, dan zal het automatisch de FD van van S_1 uitvoeren.

Een set S van FD is irrudicibel :

Als en slechts als S voldoet aan drie eigenschappen :

- De rechterkant van elke FD in S juist één attribuut bevat (=singleton-set).
- De linkerkant van elke FD in S ook irriducibel is, dus dat geen enkel attribuut kan verworpen worden van de determinant zonder de closure S_+ te veranderen (dus zonder S te converteren in een set die niet equivalent is aan S). We noemen zo een FD linker-irrudecibel.
- Geen enkele FD kan verworpen worden van S zonder de closure S_+ te veranderen.

- *Voorbeeld 11.6 (pagina 342 tem. 343)*

Definitie irreducibel equivalent

Een set I van FD dat irreducibel is en equivalent tot een andere set S van FD is een irreducibel equivalent van S. Dus gegeven een bepaalde set S van FD dat moet worden uitgevoerd, dan is het voldoende voor het systeem om de FD te vinden en uit te voeren in een irreducibel equivalent I in plaats.

12 Chapter 12 : Further Normalization I : 1NF, 2NF, 3NF, BCNF

12.1 Introduction

Databank design

> Wie zegt dat ons databankdesign de juiste is?
> *Voorbeeld 12.1 (pagina 350 tem. 351)*
> Dit voorbeeld maakt gebruik van figuur 12.1. We zien dat er redundancy kan optreden (overtolligheid). Dit moet vermeden worden adhv. normalizaties.
> We weten dat relation values altijd genormalizeerd zijn en relation variables zijn genormalizeerd zolang hun legale values genormaliseerde relaties zijn, hier uit volgt dat relvars altijd genormalizeerd zijn (volgens het relationele model).
> We zeggen dat relvars en relations altijd in *first normal form* zijn.

Normal forms

Normal form is de basis van de verdere normalisatie. Een relvar is in een bepaalde normal form als het voldoet aan een bepaald voorgeschreven set van voorwaarden.

Figuur 12.2 : Levels van normalisatie (pagina 351)

> **Notities :**
>
> > Er zijn verschillend normal forms gedefinieerd. Alle genormalizeerde relvars zijn in 1NF, sommige 1NF relvars zijn ook in 2NF en sommige 2NF relvars zijn in 3NF. Door onvolledigheden van de 3NF werd de Boyce/Codd normal form (BCNF) ingevoerd. Daarna volgden ook nog 4NF en 5NF (projection-join normal form).

Definitie normalisatie procedure

> Een relvar die zich in een bepaald normal form bevindt, kan vervangen worden door een set van relvars in een andere verlangde vorm. Het is dus de succesvolle reductie van een gegeven collectie

van relvars tot een meer verlangde vorm.
Deze procedure is omkeerbaar, het is mogelijk om de output van de procedure te nemen en te mappen naar de input. Uit het omkeerbaar zijn, volgt dat het normalisatie proces *nonloss* of *information-behoudend* is.

Structure of the chapter

Enkele opmerkingen :

- Het algemene idee van normalisatie is dat de databank ontwerper moet gaan voor relvars in de ultieme normal form (5NF). Maar het ontwerpproces mag niet enkel gebaseerd zijn op normalisatie.
- De normalisatie procedure wordt gebruikt als basis om de verschillende normal forms te introduceren. De ideën van normalisatie kunnen gebruikt worden om te verifiëren of het resulterende ontwerp niet ingaat op de principes van normalisatie.

12.2 Nonloss decomposition and functional dependencies

Het concept nonloss decompositie

Het proces van decompositie is eigenlijk een proces van projectie, dus de decompositie operator in de normalizatie procedure is projectie. Als we zeggen dat er geen informatie verloren is, dan kunnen we ook zeggen : Als we twee relvars joinen, dan vinden we de originele relvar terug. Dus omkeerbaarheid betekent dat de originele relvar gelijk is aan de join van zijn projecties. Zo bekomen we dat de recompositie operator van de procedure join is. De decompositie van een relvar R op projecties $R_1, R_2, ..., R_n$ is nonloss als R gelijk is aan de join van $R_1, R_2, ..., R_n$.

Theorema van Heath

Laat R een relvar zijn met A, B en C als sets attributen van R. Als R voldoet aan de FD A $\rightarrow$ B, dan is R gelijk aan de join van zijn projecties $\{A, B\} en \{A, C\}$.

More on functional dependencies

Enkele bijkomende opmerkingen mbt. FDs :

- Irrucibiliteit (onherleidbaarheid) : Een FD is links-irreducibel als zijn linker zijde niet te groot is.
- FD diagrammen : *Figuur 12.4 : FD diagrammen (pagina 356)*
  Er zijn altijd pijlen uit de kandidaatsleutels. Dus de normalisatie procedure kan gezien worden als het weglaten van pijlen uit sleutels die geen kandidaatsleutels zijn.
- FD als semantisch begrip : FDs zijn een soort van integrity constraints.

12.3 First,second and third normal forms

We veronderstellen in deze sectie dat elke relvar juist één kandidaatsleutel heeft, die tevens de primaire sleutel is.
Informele definitie van 3NF

> Een relvar is 3NF als en slechts als de niet-sleutel attributen beide :
> mutueel onafhankelijk zijn en irreducibel afhankelijk van de primarie sleutel.
> We vermelden dat twee of meer attributen mutueel onafhankelijk zijn als geen enkele van hen FD is van een combinatie van de andere. Zo een onafhankelijkheid impliceert dat elk attribuut onafhankelijk geupdated kan worden van de rest.
> *Voorbeeld 12.2 (pagina 358)*

Definitie first normal form

> Een relvar is in 1NF als en slechts als, in elke legale value van die relvar, elke tuple juist één waarde bevat voor elk attribuut. Dit benadrukt dat elke relvar in 1NF is, en dit is correct. Maar deze vorm is niet gewenst voor een aantal redenen.

Voorbeeld 12.3 (pagina 358 tem. 361)

> **Notities :**

In dit voorbeeld maken we gebruik van figuren 12.5, 12.6, 12.7 en 12.8 .

- Update onregelmatigheden : moeilijkheden met de update operatoren INSERT, DELETE en UPDATE.
- In dit voorbeeld wordt er veel te veel informatie gebundeld, de oplossing is dus : ontbundelen.
- Een andere karakteristiek voor de normalisatie procedure is een ontbundeling procedure : plaats logisch gescheiden informatie in gescheiden relvars.

Definitie second normal form

Een relvar is in 2NF als en slechts als het in 1NF is en elk niet-sleutel attribuut is irreducibel afhankelijk van de primaire sleutel.

Voorbeeld 12.4 (pagina 361 tem. 363)

Notities :

In dit voorbeeld maken we gebruik van figuren 12.9 en 12.10 .

- Een relvar dat in 1NF is, en niet in 2NF, kan altijd gereduceerd worden tot een equivalente collectie van 2NF relvars (projecties), zodat de join van deze projecties de originele relvar in 1NF geeft.
- Eerste stap van de normalisatie procedure : projecties nemen om niet-irreducibele FDs te elimineren :
  Gegeven R $\{A, B, C, D\}$ met primaire sleutel $\{A, B\}$ (en veronderstelling A $\rightarrow$ D)
  dan beveelt de procedure aan om R te vervangen door zijn twee projecties $R_1 en R_2$ als volgt :
  $R_1\{A, D\}$ met primaire sleutel $\{A\}$ en $R_2\{A, B, C\}$ met primaire sleutel $\{A, B\}$ en vreemde sleutel $\{A\}$ met referentie R_1.
  En R kan teruggevonden worden door R_1 join R_2 over de vreemde-primaire sleutel.
- In dit voorbeeld is er nog een gebrek aan mutuele onafhankelijkheid tussen de niet-sleutel attributen.

- We merken transitiviteit : als FD A $\rightarrow$ C en B $\rightarrow$ C zich voordoen, dan houdt de transitieve FD A $\rightarrow$ C ook, en dit leidt tot update onregelmatigheden.
- We zien opnieuw het probleem van te veel gebundelde informatie.
- Oplossing : vervangen van de originele relvar door zijn twee projecties.

Definitie third normal form

Een relvar is in 3NF als en slechts als het in 2NF is en elk niet-sleutel attribuut is non-transitief onafhankelijk van de primaire sleutel. Geen transitieve afhankelijkheden leidt tot geen mutuele afhankelijkheden.

Voorbeeld 12.5 (pagina 364)

Notities :

- Een relvar dat in 2NF is, en niet in 3NF, kan altijd gereduceerd worden tot een equivalente collectie van 3NF relvars.
- Tweede stap van de normalisatie procedure : projecties nemen om transitieve afhankelijkheden te elimineren :
  Gegeven R $\{A, B, C\}$ met primaire sleutel $\{A\}$ (en veronderstelling B $\rightarrow$ C)
  dan beveelt de procedure aan om R te vervangen door zijn twee projecties R_1 en R_2 als volgt :
  $R_1\{B, C\}$ met primaire sleutel $\{B\}$ en $R_2\{A, B\}$ met primaire sleutel $\{A\}$ en vreemde sleutel $\{B\}$ met referentie R_1.
  En R kan teruggevonden worden door $R_1 join R_2$ over de vreemde-primaire sleutel.

Opmerking

Het level van normalisatie behoort tot semantiek, men kan niet zomaar het level bepalen aan de hand van values op een gegeven moment.

12.4 Dependency preservation

Voorbeeld 12.6 (pagina 365 tem. 366)

Definitie onafhankelijke projecties volgens Risanen

> De projecties $R_1 en R_2$ van een relvar R , zijn onafhankelijk als en slechts als volgende zinnen waar zijn :
>
> - Elke FD in R i logisch consequent met die in $R_1 en R_2$.
> - De overeenkomstige attributen van $R_1 en R_2$ vormen een kandidaatsleutel voor minstens één van het paar.
>
> Wanneer projecties onafhankelijk zijn, dan zijn de gekozen decomposities goed.

Definitie atomic relvar

> Een relvar is atomic, wanneer deze niet kan ontleed worden in onafhankelijke projecties.

Definitie afhankelijkheidstoestand = dependency preservation

> De idee dat de normalisatie procedure relvars zou moeten ontleden in projecties die onafhankelijk zijn volgens Risanen, noemen we afhankelijkheidstoestand.

Voorbeeld 12.7 (pagina 336 tem. 367)

> Het concept dependency preservation wordt verder uitgelegd en er wordt een algoritme gegeven om de normalisatie procedure uit te voeren.

12.5 Boyce/Codd normal form

In deze sectie laten we de veronderstelling vallen dat elke relvar juist één kandidaatsleutel heeft. We bekijken het algemene geval : de relvar kan twee of meer kandidaatsleutels hebben, de kandidaatsleutels kunnen samengesteld zijn en deze sleutels kunnen overlappen.

Definities determinant en triviale FD

De determinant is de linkerzijde van een FD (iets waarvan iets afhangt).
Een triviale FD is een FD in welke de linkerzijde een superset is van de rechterzijde.

Formele en informele definitie van BCNF

Een relvar is in BCNF als en slechts als elke niet-triviale, links-irreducibele FD een kandidaatsleutel heeft als zijn determinant.
Een relvar is in BCNF als en slechts als elke determinant een kandidaatsleutel is.
Maw. deze definitie zegt dat er geen andere pijlen zijn, dan pijlen vanuit kandidaatsleutels (er kunnen dus geen pijlen geëlimineerd worden door de normalisatie procedure).

Voorbeeld 12.8 (pagina 368 tem. 373)

Dit voorbeeld maakt gebruik van figuren 12.12, 12.13, 12.14 en 12.15 .

Definitie overlappende kandidaatsleutels

Twee kandidaatsleutels overlappen als ze twee of meer attributen betrekken en als ze minstens één attribuut gemeen hebben.

12.6 A note on relation-valued attributes

13 Chapter 13 : Further Normalizations II : Higher Normal Forms

13.1 Introduction

Nieuwe termen

In dit hoofdstuk maken we gebruik van MVDs (multi-valued dependency) als afgeleide van FDs en van JDs (join dependency) als afgeleide van MVDs.

13.2 Multi-valued dependencies and fourth normal form

Voorbeeld 13.1 (pagina 382 tem. 386)

Notities :

Dit voorbeeld maakt gebruik van figuren 13.1, 13.2 en 13.3 .

- Overtolligheidsprobleem veroorzaakt doordat t en x compleet onafhankelijk zijn van elkaar, oplossen door projecties te nemen.
- Er zijn ook slechte BCNFs, dus we introduceren MVDs om dit te verbeteren.

Definitie Multi-valued dependency

Laat R een relvar zijn en laat A, B en C subsets zijn van de attributen van R. We zeggen dat B multi-dependent is van A , $A \twoheadrightarrow B$, als en slechts als , in elke legale value van R, de set van B values, gematcht met een gegeven AC paar value, enkel afhangt van de A value en is onafhankelijk van de C value.
Het is gemakkelijk te tonen dat gegeven een relvar R $\{A, B, C\}$, de MVD $A \twoheadrightarrow B$ voldoet als en slechts als de MVD $A \twoheadrightarrow C$ ook voldoet. We schrijven dan : $A \twoheadrightarrow B|C$
MDV is een veralgemening van FD, dwz. elke FD is een MVD.
Een FD is een MVD in welke de set van afhankelijke values die matchen met een bepaalde determinant,altijd een singleton set is : als $A \rightarrow B, dan A \twoheadrightarrow B$.

Theoreme van Fagin

> Laat R $\{A, B, C\}$ een relvar zijn, waarbij A, B en C sets van attributen zijn. Dan is R gelijk aan de join van zijn projecties $\{A, B\}$ en $\{A, C\}$ als en slechts als R voldoet aan de MVDs $A \twoheadrightarrow B|C$.

Definitie fourth normal form

> Relvar R is in 4FN als en slechts als, telkens wanneer er subsets A en B van de attributen van R bestaan, zodat de nontriviale MVD $A \twoheadrightarrow B$ voldaan is, dan zijn alle attributen van R functioneel afhankelijk van A.
> We noemen $A \twoheadrightarrow B$ triviaal als ofwel A een superset is van B of A union B is de volledige heading.
> Dus de enige nontriviale FD/MDVs in R zijn van de vorm $K \rightarrow X$ met K superkey van een attribuut X.

Hoe RVAs elimineren?

> RVAs zijn relation-valued attributes. Het is beter de RVAs eerst te scheiden door projecties en daarna de overige RVAs te vervangen door hun scalaire attributen.

13.3 Join dependencies and fifth normal form

Definitie n-decomposable relvar

> Een relvar is n-decomposable wanneer deze non-decomposed kan worden in n projecties met n groter dan twee.

Voorbeeld 13.2 (pagina 386 tem. 391)

> **Notities :**
>
> Dit voorbeeld maakt gebruik van figuren 13.4 en 13.5 .
>
> - Noncyclisch nature van een constraint : Als s_1 gelinkt is tot p_1 en p_1 is gelinkt tot j_1 en j_1 op zijn beurt met s_1, dan moeten s_1 en p_1 en j_1 in dezelfde tuple voorkomen.

- Een relvar zal n-decomposable zijn voor een bepaalde n groter dan twee als en slechts als het voldoet aan een bepaalde cyclische constraint.

Definitie join dependency

Laat R een relvar zijn, en laat A, B, ... , Z subsets zijn van attributen van R. Dan zeggen we dat R voldoet aan
de JD * $\{A, B, ..., Z\}$, als en slechts als elke legale value van R gelijk is aan de join van zijn projecties op A,B,...,Z.
Een MDV is een speciaal geval van JD, of JDs zijn een veralgemening van MVDs. Formeel hebben we :
$A \rightarrow\rightarrow B|C = *\{AB, AC\}$.
JDs zijn de meest algemene vormen van afhankelijkheid.

Definitie fifth normal form

Een relvar is 5NF (ook projection-join normal form), als en slechts als elke non-triviale JD die voldaan is door de relvar R, geïmpliceerd is door de kandidaatsleutels van R, waarbij :

- de JD $*\{A, B, ..., Z\}$ op R triviaal is, als en slechts als minstens één van A,B,...,Z de set van al de attributen van R is.
- de JD $*\{A, B, ..., Z\}$ op R geïmpliceerd is door de kandidaatsleutel(s) van R als en slechts als elke A,B,...,Z een superkey is voor R.

5FN is de ultieme normal form. Een relvar in 5NF is vrij van onregelmatigheden bij het nemen van projecties.

13.4 The normalization procedure summarized

Techniek van nonloss decomposition

Gegeven een bepaalde relvar in 1NF en een set van FDs, MVDs en JDs dat toegepast zijn op R, dan kunnen we R reduceren tot een collectie van kleinere relvars, equivalent met R in een welgedefinieerde vorm. Elke stap van de reductie is een projectie, de constraints worden gebruikt om de juiste projecties te nemen.

13.5 A note on denormalization

Volledige normalisatie leidt tot verschillende gescheiden relvars, dit leidt opnieuw tot veel gescheiden bestanden, en dit leidt tot veel I/O. Denormalisatie moet gedaan worden op het niveau van opgeslagen bestanden, niet op het niveau van base relvars.

What is denormalization?

Definitie normalisatie

> Een relvar normalizeren, betekent dat we R vervangen door een set van projecties , zodat R equivalent is met de join van die projecties. Het objectief hierbij is om overtolligheid te reduceren.

Definitie denormalisatie

> Laat $R_1, ..., R_n$ een set zijn van relvars. De denormalisatie van die relvars, betekent dat deze relvars worden vervangen door hun join R. Op een zodanige manier dat de projectie van R over bepaalde attributen , gegarandeerd is de gescheiden relvars te bekomen. Het objectief is om redundancy te verhogen, met R op een lager level dan $R_1, ..., R_n$.

Some problems

- waar stoppen met denormalisatie?
- overtolligheids- en update problemen.
- soms is denormalizatie goed voor de ene applicatie, maar slecht voor een andere.

13.6 Orthogonal design (a digression)

Further Observations

13.7 Other normal forms

De theorie van de normalizatie

Dependency theorie , en er zijn nog andere NFs...

Andere normal forms

- Domain key NF (DK/NF): deze vorm is niet gedefinieerd in termen van FDs, MVDs en JDs.
  Een relvar R is in DK/NF als en slechts als elke constraint op R een logische consequentie is van de domain constraints en key constraints die toepasbaar zijn op R.
  - domain constraint : beperking met het effect dat values van een gegeven attribuut genomen zijn van een voorgeschreven domein. (attribuut constraint)
  - key constraint : beperking met het effect dat een bepaalde set van attributen een kandidaatsleutel vormen.

  Als deze beperkingen in acht genomen zijn, zijn alle andere beperkingen automatisch vervuld.
- Restriction-union NF : De mogelijkheid om de originele relvar de ontleden via restricties en niet via projecties. Vb. union restrictie.
- Sixth NF : Fifth NF is de finale NF volgens de klassieke projectie en join. Er bestaat nog een nieuwe sixth NF door veralgemening van projectie en join, veralgemening van join afhankelijkheid,... Alle relvars in 6NF zijn in 5NF.

14 Chapter 14 : Semantic Modeling

Part 4 : TRANSACTION MANAGEMENT

15 Chapter 15 : Recovery

15.1 Introduction

Definitie recovery in een databanksysteem

> Recovery in een databank is het herstellen van de databank zelf : het herstellen van de databank tot een correcte toestand nadat een bepaalde fout een incorrecte toestand van de databank weergeeft, of op zijn minst vermoedt.
> Het onderliggende principe is redundancy(overtolligheid) op het fysieke niveau.
> In andere woorden, om de databank herstelbaar te maken, moeten we zorgen dat elke stuk informatie die deze databank bezit gereconstrueerd kan worden door andere opgeslagen informatie ergens anders in het systeem.

Veronderstellingen

> We veronderstellen dat het databanksysteem een large system is, dus shared en multi-user. Bij small systems moet de gebruiker zelf instaan voor de recovery aangezien dit bijna niet aanwezig is in het systeem. De gebruiker moet dan backups voorzien.

15.2 Transaction

Definitie transaction

> Een transactie is een logische eenheid van werk : het begint met een uitvoering van een BEGIN TRANSACTION operatie en eindigt met een uitvoering van een COMMIT of ROLLBACK operatie.

Figuur 15.1 : Een voorbeeld van een transactie (pseudocode) (pagina 447)

> Een databank kan incorrect zijn tussen de uitvoering van twee updates, het geeft geen ware toestand weer van de reële wereld.

Garantie

> Er kan geen garantie gegeven worden zodat alle updates altijd worden uitgevoerd, want een systeem kan bijvoorbeeld crashen tussen de twee updates of er kan zicht een overflow voordoen, ...
> Een systeem met een *transaction management* kan zulks garantie geven.

Definitie transaction management

Dit management garandeert dat, wanneer de transactie bepaalde updates uitvoert en wanneer er zich dan een fout voordoet vooraleer het bereiken van het geplande einde, die updates ongedaan zullen gemaakt worden. Dus de transactie zal ofwel helemaal worden uitgevoerd, ofwel wordt deze totaal geanulleerd.

Definitie transaction manager

De manager voorziet dit transaction management, en wordt ook nog TP monitor of transaction processing monitor genoemd. De COMMIT en ROLLBACK operaties zijn de sleutels tot de werking hiervan :

- De COMMIT operatie signaleert een succesvolle einde-van-transactie : het vertelt aan de transaction manager dat de transactie succesvol beëindigd is en dat de databank in een correctie toestand moet verkeren, alle updates mogen worden uitgevoerd.
- De ROLLBACK operatie signaleert een niet-succesvolle einde-van-transactie : het vertelt aan de transaction manager dat de transactie dat er iets misgelopen is, de databank kan in een niet-correctie toestand zijn, alle updates moeten ongedaan gemaakt worden.

Nog enkele belangrijke puntjes

- Implicit ROLLBACK : het systeem gaat uit van een impliciete ROLLBACK.
- Message handling : een transactie doet updates en geeft ook berichten terug om aan te geven wat er gebeurd is.
- Recovery log : hoe is het mogelijk een update ongedaan te maken? Dit kan omdat het systeem een log bijhoudt met alle details van de updates (op schijf). De log bestaat uit twee delen : een actief/online gedeelte en een archief/offline gedeelte.
- Statement atomicity : de statement uitvoeringen moeten atomic zijn. In relationele systemen zijn statements set-level, zodat deze opereren op verschillende tuples tegelijk, als er in dit geval een fout optreedt , moet de databank onveranderd blijven.

- Program execution is a sequence of transactions : COMMIT en ROLLBACK beëindigen de transactie, niet het programma. Eén programma kan uit veel verschillende lopende transacties bestaan.
  Figuur 15.2 : Uitvoering van een programma als een sequentie van transacties (pagina 449)
- No nested transactions : (voorlopig) kunnen we geen transactie beginnen, als er nog een in proces is.
- Correctness : we veronderstellen dat wanneer T een transactie is en deze toestand D_1 transformeert naar D_2, en wanneer D_1 correct is, dan is D_2 ook correct.
- Multiple assignment : we negeren multiple assignments, deze worden nog niet gesupporteerd, alhoewel met het zou kunnen zien als atomic.

15.3 Transaction recovery

Definitie commit point

COMMIT stelt een commit point vast (ook synchpoint genoemd). Een commut point correspondeert tot het succesvolle einde van de transactie, dus tot het punt dat de databank in correctie toestand verkeert. ROLLBACK gaat terug naar de toestand zoals het was bij BEGIN TRANSACTION, wat betekent terug naar het vorige commit point.

Wat gebeurt er bij het ontstaan van een commit point?

- alle databank updates gemaakt sinds het vorige commit point zijn uitgevoerd, dus permantent. Eens uitgevoerd, kan een update nooit ongedaan gemaakt worden.
- alle databank positionering is verloren en alle tuple locks zijn vrijgegeven. Databank positionering is de idee dat een uitvoerbaar programma adressen heeft voor tuples (cursors). Dit is verloren bij het commit point.(Dit kan ook bij ROLLBACK)

Sommige systemen ondersteunen dat databank positionering niet verloren is en dat tuple locks kunnen behouden worden.

transacties als unit van recovery

Dit kan door de write-ahead log rule : de log moet fysisch geschreven worden voor het COMMIT proces kan worden beëindigd. Voorbeeld wanneer een COMMIT is uitgevoerd en wanneer het systeem crashed vooraleer de updates fysisch worden geschreven.
Meer precies :

- de log record van een gegeven databankupdate moet fysisch geschreven worden in de log vooraleer die update fysisch geschreven is in de databank.
- alle andere log records moeten fysisch geschreven worden in de log vooraleer de COMMIT log record fysisch geschreven is in de log.
- COMMIT moet niet beëindigd worden vooraleer de COMMIT log record fysisch geschreven is in de log.

The ACID properties

Transacties moeten de ACID eigenschappen bezitten :

- Atomicity : transacties zijn atomic , alles of niets.
- Correctness : transacties transformeren een correctie toestand in een andere correctie toestand.
- Isolation : stel A en B verschillende transacties, A kan B zijn updates zien en omgekeerd, maar niet tegelijkertijd.
- Durability : eenmaal de transactie uitgevoerd wordt (COMMIT), dan blijven de updates permanent in de databank, ookal is er een systeem crash.

15.4 System recovery

Soorten failure

- lokaal : heeft enkel effect op de transactie in welke de faling zich heeft voorgedaan.

- globaal : heeft effect op alle transacties die in progres zijn op het tijdstip van de faling.

Twee soorten global failure

- System failures (soft crash): (vb stroomonderbreking) dit heeft effect op alle transacties in progres, maar brengt geen fysische schade toe aan de databank. De precieze toestand van een transactie dat in progres was, is niet meer gekend, en kan niet meer succesvol worden uitgevoerd, en moet dus ongedaan gemaakt worden en later weer herdaan worden bij het opnieuw opstarten van het systeem.
- Media failures (hard crash) : (vb schijf crash) dit brengt wel schade toe aan de databank of aan een deel ervan.

Hoe weet het systeem welke transacties moeten worden herdoen of ongedaan gemaakt?

Op bepaalde intervallen neemt het systeem een checkpoint. Het checkpoint record bevat een lijst van alle transacties dat in progres waren op het moment dat het checkpoint genomen was.

Figuur 15.3 : Vijf transactie categoriën (pagina 454)

Voorbeeld 15.1 (pagina 454 tem. 455)
Er wordt een procedure beschreven met alle stappen die het systeem onderneemt bij restart time.
Het systeem werkt achterwaarts door de log (undo) backward recovery, dan weer voorwaarts (redo) forward recovery.
Het systeem aanvaardt nieuw werk enkel wanneer de recovery gedaan is.

ARIES

ARIES is Algorithms for Recovery and Isolation Exploiting Semantics en doet redo voor undo in drie fasen :

- analysis : bouw de REDO en UNDO lijst

- redo : start van een plaats in de log bepaalt door de analysis fase en zet de databank terug in de staat dat het was op de tijd van de crash (repeating history).
- undo : undo de effecten van de transacties die niet konden worden uitgevoerd.

15.5 Media recovery

Definitie media failure

> Media failure is een fout bij welke een deel van de databank fysisch wordt vernietigd. Dit kan enkel worden hersteld door een backup copy te herladen, om elke transactie te herdoen die nog niet gedaan waren bij het maken van de backup (forward recovery).

15.6 Two-phase commit

Wanneer is two-phase commit handig?

> Dit is belangrijk wanneer een gegeven transactie kan inwerken op verschillende onafhankelijke recourse managers.

Voorbeeld 15.2 (pagina 356 tem. 357)

> De transactie gaat uit van een system-wide COMMIT of ROLLBACK : deze COMMIT of ROLLBACK wordt behandeld door een systeem component, die coordinator wordt genoemd.
>
> De coordinator moet garanderen dat beide resource managers alletwee de updates committen of rollbacken waarvoor zij verantwoordelijk zijn. (ook als het systeem faalt in het midden van het proces).
> Het two-phase commit protocol enables de coordinator om dit te garanderen.
>
> Gegeven dat de transactie zijn databank proces succesvol heeft beëindigd, zodat de system-wide instructie COMMIT is. De coordinator gaat verder met 2 fases :

- PREPARE : Elke resource manager wordt opgelegd om alle log records te schrijven naar zijn eigen fysische log. Als dit succesvol is, antwoorden de betrokkenen met OK aan de coordinator, anders NOT OK.
- COMMIT : Wanneer de coordinator antwoorden gekregen heeft van alle recourse managers, dan schrijft hij eerst zijn eigen fysische log, afhankelijk van zijn beslissing(alles Ok , dan Commit, minstens 1 NOT OK : rollback). Daarna informeert de coordinator alle managers met zijn beslissing, daarna moet elke manager de lokale transactie uitvoeren. Elke manager moet doen wat de coordinator zegt. Bij faling kijkt het systeem bij het heropstarten naar de beslissings record log, als die er is, kan er verder gegaan worden, anders wordt er vanuit gegaan dat de beslissing ROLLBACK was.

15.7 Savepoints (a digression)

15.8 SQL facilities

16 Chapter 16 : Concurrency

16.1 Introduction

Definitie concurrency

> Deze term refereert naar het feit dat de DBMS vele transacties toestaat om toegang te hebben tot dezelfde databank op hetzelfde moment. Er is dus een mechanisme nodig om te zorgen dat deze transacties niet in botsing komen met elkaar.

Ideën van concurrency

> Deze ideën zijn onafhankelijk van het ondeliggende systeem (relationeel of anders) en het is tevens een zeer groot onderwerp.

16.2 Three concurrency problems

Drie problemen

> We merken op dat elke beschouwde transactie op zichzelf correct is, maar door de interactie met andere transacties duiken er problemen op. De drie problemen worden besproken in volgende drie paragrafen.

The lost update problem

Figuur 16.1 : Transactie A verliest een update op tijdstip t_4 want B schrijft het over (pagina 467)

> Beide transacties doen updates onafhankelijk van elkaar en weten dus niet wat er ondertussen gebeurt.

The uncommitted dependency problem

Wanneer komt dit probleem voor?

Dit komt voor wanneer een transactie toegestaan is om een tuple te retrieven of te updaten, die upgedated is door een andere transactie, maar die nog niet gecommitted was door deze tweede transactie. Dus zolang het niet gecommitted wordt, kan deze tweede transactie nog gerollbacked worden, zodanig dat de eerste transactie gegevens zal gezien hebben die eigenlijk niet bestaan.

Figuur 16.2 : Transactie A wordt afhankelijk van een nog niet gecommittede verandering op tijdstip t_2 (pagina 468)

Transactie A ziet een nog niet gecomittede update op tijdstip t_2, deze update wordt ongedaan gemaakt op tijdstip t_3, dus transactie A is aan het opereren op een vals vermoeden, namelijk het vermoeden dat tuple t de value heeft gezien op tijdstip t_2, en t heeft eigenlijk de value van op tijdstip t_1.

Figuur 16.3 : Transactie A updates een nog niet committede verandering op tijdstip t_2, en verliest die update op tijdstip t_3 (pagina 468)

Transactie A wordt afhankelijk van een nog niet gecommiteerde verandering op tijdstip t_2, maar verliest ook een update op tijdstip t_3, omdat de rollback op tijdstip t_3 veroorzaakt dat tuple t moet hersteld worden naar de value die hij had op tijdstip t_1. (Dit is een andere versie van het lost update probleem.)

The inconsistent analysis problem

Figuur 16.4 : Transactie A verricht een inconsistente analyse (pagina 469)

Transactie A heeft een inconsistente toestand van de databank gezien en heeft daardoor een inconsistente (incorrecte) analyse doorgevoerd.

A closer look

We beschouwen dat er enkel problemen optreden bij databank retrievals en databank updates. We bekijken deze operaties als read en write. Het is duidelijk dat als A en B concurrente transacties zijn, dan kunnen er problemen optreden wanneer A en B willen readen of writen naar het zelfde databank object, vb. tuple t. Er zijn vier mogelijkheden :

- RR : A en B willen beide t readen. Reads kunnen niet in botsing komen met elkaar, dus hier geen probleem.
- RW : A reads t en dan wil B t writen. Als B toegestaan is te writen, dan kan het inconsistente analyse probleem voorkomen. Dus laatsgenoemd probleem wordt veroorzaakt door een RW conflict.
  Opmerking : wanneer B de write op t uitvoert en A reads opnieuw, dan zal deze een value zien, verschillend van wat hij daarvoor gezien had. Dit noemt men nonrepeatable reads en deze worden dus ook veroorzaakt door RW conflicten.
- WR : A writes t en dan wil B t readen. Als B toegestaan is te readen, dan komt het uncommited dependency probleem aan bod. Dus laatsgenoemd probleem wordt veroorzaakt door een WR conflict.
  Opmerking : Als de read van B toegestaan is, spreken we van een dirty read.
- WW : A writes t en dan wil B t writen. Als B toegestaan is om te writen, dan kan het lost update probleem voorkomen. Dus lost updates worden veroorzaakt door WW conflicten.
  Opmerking : Als B toegestaan is te writen, spreken we van een dirty write.

16.3 Locking

Definitie locking

> De problemen die we zonet besproken hebben kunnen door middel van locking worden opgelost. De idee is eigenlijk : wanneer een transactie A de zekerheid wil dat een zeker object (waarin het geïntresseerd is), niet zal veranderen wanneer deze transactie even niet kijkt, dan kan deze een lock voorzien op dat object, deze lock voorkomt dat het object veranderd wordt. Het object blijft dus in dezelfde toestand zolang dat transactie A dat wilt.

Gedetailleerde beschrijving van locking

- Er zijn twee soorten locks : exclusieve lock (X) en gedeelde (shared) lock (S), ook write en read locks genoemd. Enkel tuples kunnen gelocked worden.
- Wanneer A een X lock houdt op tuple t, dan kan een aanvraag van een transactie B voor een lock van elk type op t niet direct toegestaan worden.

- Wanneer A een S lock houdt op tuple t, dan :
  - kan een aanvraag van transactie B voor een X lock op tuple t niet direct toegestaan worden.
  - kan een aanvraag van transactie B voor een S lock wel toegestaan worden (dus B kan ook een S lock houden op tuple t).

Deze regels kunnen samengevat worden adhv een lock type compatibiliteitsmatrix :
Figuur 16.5 : Compatibiliteitsmatrix voor lock types X en S (pagina 471)
Bij deze matrix is N een conflict en Y een compatibiliteit. De matrix is symmetrisch.

Data acces protocol of locking protocol

Dit protocol maakt gebruik van X en S lock om te garanderen dat de drie voorgaande problemen niet kunnen voorkomen.

- Een transactie die een tuple wil retrieven moet eerst een S lock verwerven.
- Een transactie die een tuple wil updaten moet eerst een X lock verwerven. Als deze transactie al een S lock heeft op die tuple, moet hij deze lock upgraden naar een X lock.
- Een transactie B gaat in de *wait state*, wanneer een lock request van B niet direct kan worden toegestaan, omdat het in conflict is met een lock die reeds gehouden wordt door transactie A. B moet minstens wachten tot transactie A de lock heeft vrijgegeven. Het systeem moet *livelock of starvation* voorkomen (=wait state forever). Dit kan voorkomen worden door het principe van FCFS aanvragen.
- X locks worden vrijgegeven aan het einde van een transactie. Bij S locks geldt hetzelfde. Dit protocol noemt het *strict two-phase locking*.

16.4 The three concurrency problems revisited

The lost update problem

Figuur 16.6 : Geen updates gaan verloren, maar er ontstaat een deadlock op tijdstip t_4 (pagina 472)

> In deze figuur maken we gebruik van het strict two-phase locking protocol. Er onstaat deadlock, omdat beide transacties in een wait state zijn.

The uncommitted dependency problem

Figuur 16.7 en 16.8 : Transactie A wordt verhinderd om een nog niet gecommiteerde verandering te zien op tijdstip t_2 en transactie A wordt verhinderd om een nog niet gecommiteerde verandering op tijdstip t_2 up te daten (pagina 473)

> In deze figuur maken we gebruik van het strict two-phase locking protocol. Het originele probleem wordt hier opgelost, want transactie A is niet langer afhankelijk van een nog niet gecommiteerde verandering.

The inconsistent analysis problem

Figuur 16.9 : Inconsistente analyse is voorkomen, maar een deadlock ontstaat opt tijdstip t_7 (pagina 474)

> In deze figuur maken we gebruik van het strict two-phase locking protocol. Er ontstaat een deadlock.

16.5 Deadlock

Definitie deadlock

> We maken gebruik van *Figuur 16.10 : Voorbeeld van deadlock (pagina 457).*
> Sommige systemen werken met timers. Deadlock is een situatie in welke twee of meer transacties in simultane wait state zijn, elk van hen wachtend voor de andere om hun lock vrij te geven, vooraleer zij verder kunnen gaan. Als deadlock voorkomt, moet het systeem dit detecteren, dit kan door de cyclus in de Wait-For Graaf te detecteren. Om de deadlock te breken, moeten we een slachtoffer zoeken tussen de transacties en deze te roll backen.

Deadlock avoidance

In plaats van toe te staan dat deadlocks voorkomen en uit te zoeken wat er mee te doen, is het best mogelijk deadlock te voorkomen door het locking protocol aan te passen. We beschouwen twee versies : Wait-Die en Wound-Wait.

- Elke transactie is timestamped met zijn starttijd. (uniek)
- Wanneer transactie A een lock voor een tuple aanvraagt dat al gelocked is door transactie B, dan :
  - WAIT-DIE : A wacht als het ouder is dan B,(vroeger tijdstip) anders it DIES = A wordt gerollbacked en wordt herstart.
  - WOUND-WAIT : A wacht als het jonger is dan B, anders it WOUNDS B = B is rollbacked en wordt herstart.
- Als een transactie moet herstart worden, dan krijgt het zijn originele timestamp.

We zien dat elke transactie zijn doel bereikt en livelock niet kan voorkomen en geen enkele transactie zal te veel worden herstart. Het doet wel te veel rollbacks.

16.6 Serializability

Definitie serializability

> Serializability is het criterium voor correctheid voor de interleaved (passen in elkaar) uitvoering van een set van transacties. Zo een uitvoering is correct als en slecht als ze serializable is. En een gegeven uitvoering van een gegeven set van transacties is serializable (en daarom correct) als en slechts als deze uivoering equivalent is (gegarandeerd om hetzelfde resultaat te geven als) aan een bepaalde serieële uitvoering van dezelfde transacties. Een seriële uitvoering is er één in welke de transacties één voor één worden uitgevoerd in een bepaalde sequentie.

Verdere verklaring van de definitie

- Individuele transacties worden verondersteld correct te zijn (van de ene correcte toestand naar de andere).

- Transacties in een bepaalde sequentie uitvoeren is dus ook correct.
- Een interleaved uitvoering is dus correct als en slechts als het equivalent is met een serieële uitvoering (als en slechts als ze serializable is).

Definitie shedule

Gegeven een set van transacties, elke uitvoering (al dan niet interleaved) van deze transacties wordt een shedule genoemd.

Definitie serial shedule $<--->$ interleaved shedule

Uitvoeren van de transacties elk op een ander tijdstip, met geen interleaving, geeft een serial shedule. Een interleaved shedule is een shedule die niet serial is.

Definitie equivalente shedules

Twee shedules worden equivalent genoemd als en slechts als ongeacht de initiële staat van de databank, deze shedules gegarandeerd zijn om hetzelfde resultaat te produceren als de andere. Dus een shedule is serializable (en correct) als en slechts als deze equivalent is met een serial shedule.

Opmerking

Twee serial shedules of twee interleaved shedules met deze dezelfde transacties kunnen verschillende resultaten produceren en toch nog correct zijn. Shedules die equivalent zijn aan deze shedules zijn dus ook correct.

Two-phase locking theorema

Als alle transacties gehoorzamen aan het two-phase locking protocol, dan zijn alle interleaved shedules serializable.
Het two-phase locking protocol luidt :

- Vooraleer op een object te opereren, moet een transactie een lock op dat object verkrijgen. (growing fase)
- Na het releasen van een lock, mag een transactie nooit meer locks verkrijgen. (shrinking fase)

Definitie S is een serialization van I

> Laat I een interleaved shedule zijn met betrekking tot een set van transacties $T_1, ..., T_n$. Als I serializable is, dan bestaat er minstens één serial shedule S met transacties $T_1, ..., T_n$ zodat I equivalent is met S. S is een serialization van I.

16.7 Recovery revisited

Figuur 16.11 : Een unrecoverable shedule (painga 479)

Voorwaarde voor recoverable shedule

> Als A één van de updates van B ziet, dan mag A niet committen voordat B eindigt. Nood aan cascade rollbacks?

Figuur 16.12 : Een shedule met een cascaded rollback (pagina 480)

Wanneer de ene transactie rollbackt en de andere kan doordoor ook rollbacken (cascade), dan is er een gevaar voor cascade-kettingen.
Voorwaarde voor cascade-free shedule

> Als A één update van B ziet, dan mag A niet updaten voordat B eindigt. Strikte two-phase locking garandeert dat alle shedules cascade-free zijn. Als een sheduele cascade-free is dan is deze ook recoverable.

16.8 Isolation levels

Definitie isolatie level

> Vooreerst merken we op dat serializability isolatie uit ACID garandeert, de programmeur moet zich geen zorgen maken als er twee transacties gelijktijdig worden uigevoerd. Maar protocollen om serializability te garanderen kunnen de graad van concurrency sterk verminderen.
> Het isolatie level dat toegepast is op een gegeven transactie mag gedefinieerd worden als de graad van tussenkomst die de transactie bereid is nog te tolereren van andere transacties.

Verschillende levels

Er kunnen minstens vijf levels gedefinieerd worden, hoe hoger het level, hoe kleiner de graad van tussenkomst. We beschouwen hier twee levels : Repeatable read (RR) en Cursor stability (CS).

Repeatbale read is het maximum level, als alle transacties opereren op dit level, dan zijn alle shedules serializable.
Cursor stability : Als transactie A

- addreseerbaarheid verkrijgt over een tuple (pointer naar de tuple) en dus
- een lock over t verkrijgt, en dan
- zijn addresseerbaarheid vrijgeeft, zonder te te updaten, en zo
- de X lock niet promoot, dan
- kan de lock vrijgegeven worden zonder te wachten op een einde-van-transactie.

Maar een andere transactie kan nu wel t updaten en committen. Zo kan er inconsistentie van de databank ontstaan.

Enkele opmerkingen

- Een transactie onder CS voldoet niet aan het two-phase locking protocol en garandeert dus geen serializability. CS geeft meer concurrency dan RR.
- Als transactie T wordt uitgevoerd op een level dat niet het maximum level is , dan kan een correcte toestand van de databank niet gegarandeerd worden.
- Er bestaat een LOCK TABLE statement.
- Als je lager dan het maximum level opereert, dan kan het probleem van *phantoms (spoken)* ontstaan.

Phantoms

Voorbeeld 16.1 (pagina 482)

16.9 Intent locking

Definitie locking granularity

De eenheid voor locking kan niet enkel gaan over tuples, maar ook over relvars , of een component in een tuple ,...
Er is een trade-off : Hoe fijner de granularity, hoe groter de concurrency, hoe ruwer de granularity, hoe kleiner de overhead.

Conflict

Stel dat een transactie T een X lock vraagt voor een relvar R, het systeem moet dan kunnen vertellen of een andere transactie al een lock heeft op een tuple van R. Want in dat geval krijgt T de X lock niet. We introduceren het intent locking protocol.

Definitie intent locking protocol

Geen enkele transactie is toegestaan een lock te verkrijgen op een tuple, vooraleer hij een intent lock heeft verkregen op de relvar die de tuple bevat.

Definities intent lock

X en S locks gelden voor zowel tuples als relvars, maar intent lock gelden enkel voor relvars. We veronderstellen dat transactie T een lock heeft aangevraagd voor relvar R. Er zijn verschillende soorten :

- Intent shared (IS) : T wil S locks zetten op tuples van R, om stabiliteit te garanderen van die tuples wanneer in proces.
- Intent exclusive (IX) : Zelfde als IS + T kan willen updaten op tuples in R en wil dus een X lock op die tuples.
- Shared (S) : T kan concurrente readers tolereren, maar geen concurrente updaters in R. T zal zelf ook geen tuples in R updaten.
- Shared intent exclusive (SIX) : Combinatie van S en IX, T kan readers tolereren, maar geen updaters in R + T mag individuele tuples in R updaten en zal daarvoor X locks plaatsen op die tuples in R.

- Exclusive (X) : T kan geen enkele concurrente toegang tot R tolereren. T mag zelf ook niet updaten.
- *Figuur 16.13 : Compatibiliteitsmatrix om intent locks in te voegen (pagina 484)*

Definitie intent locking protocol

- Vooraleer een gegeven transactie een S lock kan krijgen op een tuple, moet het eerst een IS of sterkere lock op de revlar verkrijgen, die de tuple bevat.
- Vooraleer een gegeven transactie een X lock kan krijgen op een tuple, moet het eerst een IX of sterkere lock verkrijgen op de relvar, die de tuple bevet.
- *Figuur 16.14 Lock type prioriteit graaf (pagina 485)*

Definitie lock escalation

Om de conflicterende vereisten van hoge concurrency en lage lock management overhead te balanceren.

16.10 Dropping ACID

Immediate constraint checking

Correctness

Isolation

Durability

Atomicity

Concluding remarks

16.11 SQL facilities

To remember

Recovery(herstellen) en concurrency(gelijktijdig uitvoeren) zijn begaan met bescherming van data : de data beschermen tegen verlies of schade.
Beide zijn begaan met volgende problemen :

- Het systeem kan crashen in het midden van de uitvoering van een programma, daarbij de databank in een ongekende toestand achterlaten.
- Twee programma's die op hetzelfde moment worden uitgevoerd, kunnen in botsing komen met elkaar en incorrecte resultaten produceren, ofwel in de databank ofwel daarbuiten.

Part 5 : FURTHER TOPCIS

17 Chapter 17 : Security

17.1 Introduction

Definitie veiligheid en volledigheid

> Security betekent data bescherming tegen unauthorized gebruikers, er op toezien dat gebruikers zijn toegestaan om dingen te doen die ze bezig zijn. Integrity beschermt de data tegen authorized gebruikers, er op toezien dat de dingen dat de gebruikers doen, correct is. In beide gevallen moet het systeem er op toezien dat er niet wordt ingegaan op zekere constraints (behouden in catalog).

Verschillende aspecten van veiligheid

- Legale, sociale en ethische aspecten
- Fysieke controle
- Policy vragen
- Operationele problemen
- Hardware controle
- OS ondersteuning
- Onderwerpen die specifiek van belang zijn voor databanksystemen zelf.

Twee soorten databeveiliging

- Discretionary control (willekeurige controle) : Een gebruiker zal verschillende privileges (toegangsrechten) hebben op verschillende objecten, welke gebruikers kunnen welke rechten hebben
- Mandatory control (verplichte controle) : Elk dataobject is gelabeld met een zeker classificatielevel en elke gebruiker heeft een clearance level. Een gegeven dataobject kan enkel kan enkel toegang bieden tot gebruikers met een angepaste clearance.

De DBMS kan enkel de genomen beslissingen bekrachtigen

- Deze beslissingen moeten bekend gemaakt worden aan het systeem door security constraints en moeten worden bijgehouden door het syteem bij de catalog.
- Het DBMS authorizatie subsysteem zorgt voor het checken van een gegeven toegangsaanvraag.
- Het systeem moet de bron van de aanvraag kunnen herkennen. Daarvoor moeten gebruiker beschikken over een ID en een paswoord. De authenticatie is het eigenlijke controleren van ID en paswoord. Verschillende gebruikers kunnen een zelfde ID delen, dit noemen we gebruikersgroepen (roles). Gebruikersgroepen kunnen genest worden.

17.2 Discretionary acces control

Authoroties

Het is meestal handiger voor een systeem om te zeggen wat er toegestaan is , dan wat er niet toegestaan is, daarom moet het mogelijk zijn in een bepaalde taal authoroties te definiëren. Authorizaties zijn allemaal ge-ORed aan elkaar. Dus een aanvraag is toegestaan als en slechts als minstens één authoroty het toelaat.
Voorbeeld 17.1 (pagina 507)
We kunnen ook authoroties droppen.

Voorbeeld 17.2 (pagina 507 tem. 508)

Request modification

Audit trails

We mogen niet aannemen dat het veiligheidssysteem perfect is. Daarom moeten we een audit trail voorzien. Dit is een speciale file of databank in welke het systeem alle operaties bijhoudt door gebruikers op gewone data.

17.3 Mandatory acces control

Basisidee

De basisidee is dat elk dataobject een classificatielevel heeft (vb. top secret, secret,...) en dat elke gebruiker een clearance level heeft. De levels moeten een strikte ordening hebben.

Enkele simpele regels

- Eenvoudige veiligheidseigenschap : Gebruiker i kan het object j enkel opvragen enkel wanneer het clearance level van gebruiker i groter of gelijk is aan het classificatielevel van het object j.
- Stereigenschap : Gebruiker i kan het object j enkel updaten wanneer het clearance level van gebruiker i gelijk is aan het classificatielevel van object j.

De tweede regel wil ook zeggen dat alles wat geschreven wordt door gebruiker i, automatisch een classificatielevel vereist gelijk aan de gebruiker i zijn clearance level.
Enkel voor INSERT kunnen we zeggen dat het clearance level van i kleiner of gelijk moet zijn aan het classificatieleven van j. Hieruit volgt dat gebruikers dingen kunnen schrijven die ze niet kunnen lezen.

Ontstaan mandatory controls

Ontstaan in de vroege jaren 90. Deze controls werden gedocumenteerd in twee Department of Defense prublicaties, gekend als het Orange book en de Lavender Book. De Orange book definieert een set van veiligheidsvereisten. De Lavender book definieert een interpretatie van de vereisten gespecifieerd in het Orange book, speciaal voor databank systemen.

Mandatory controls als deel van een veiligheids classificatie systeem

Er zijn vier veiligheidsklassen gedefinieerd : A, B, C en D. Klasse D is het minst veilige.

- Discretionary protection : Klasse C is opgedeeld in twee subklassen $C_1 en C_2$ en C_1 is minder veilig dan C_2. Elk van de subklassen ondersteunt discretionary controls, bedoeld dat de toegang het onderwerp is van de discretie van de data eigenaar.

  - Klasse C_1 maakt een onderscheid tussen eigenaar en toegang, het ondersteunt dus het concept van gedeelde data, door ondertussen toe te staan dat gebruikers private data hebben voor hun eigen.

- Mandatory protection : Klasse B is de klasse die zoch bezighoudt met mandatory controls. Deze is ingedeeld in drie subklassen $B_1, B_2 en B_3$. Met B_1 de minst veilige.
  - Klasse B_1 vereist gelabelde veiligheidsbescherming, het vereist dat elk dataobject gelabeld is met zijn eigen classificatielevel. Het vereist ook een informeel statement van de veiligheids policy.
  - Klasse B_2 vereist een formeel statement van hetzelfde ding. Het vereist ook dat covert kanalen worden geïdentificeerd en verwijderd.
  - Klasse B_3 vereist speciaal audit en recovery ondersteuning, en ook een aangeduide veiligheids administrator.
- Verified protection : Klasse A, de meest veilige, vereist een wiskundig bewijs dat het veiligheids mechanisme consistent en adequaat is om de veiligheids policy te ondersteunen.

Multi-level security

Voorbeeld 17.3 (pagina 512 tem. 513)

17.4 Statistical databases

Definitie statistische databanken

> Een statistische databank, is een databank die queries toestaat die handelen over aggregate informatie (som, gemiddelde) en die geen queries toestaat die handelen over individuele informatie. Maar het probleem onstaat dat we individuele informatie kunnen opvragen door gebruik te maken van legale queries.

Gedetailleerd voorbeeld 17.4 (pagina 514 tem. 518)

> Dit voorbeeld maakt gebruik van figuren 17.2, 17.3 en 17.4.
> **Notities :**

- Definitie individuele tracker : een booleaanse expressie die de gebruiker toelaat individuele informatie te verkrijgen.
  Als de gebruiker een booleaanse expressie kent BE dat een specifiek individu I identificeert, en als BE kan uitgedrukt worden in de vorm van BE1 AND BE2, dan is de booleaanse expressie BE1 AND NOT BE2 een tracker voor I. En BE1 en BE1 AND NOT BE2 identificeren beide een resultatenset met kardinaliteit c in het gebied $b \leq c \leq n - b$. De reden is dat BE identiek is aan het verschil tussen BE1 en BE1 AND NOT BE2.
- Definitie algemene tracker : een booleaanse expressie die kan gebruikt worden om een antwoord te vinden op een algemene vraag die gegevens geeft over een individu. Elke expressie met een resultaat kardinaliteit c in het gebied $2b \leq c \leq n - 2b$ is een algemene tracker. Een algemene tracker bestaat meestal altijd! n = aantal tuples in de relatie, b = minimum aantal tuples in het resultaat en c = totaal aantal tuples in het resultaat.

Statistische databanken zijn dus een echt probleem.

17.5 Data encryption

The data encryption standard

Public-key encryption

17.6 SQL facilities

18 Vage databanken