



ISO SQL: Structured Query Language

Prof. dr. Paul De Bra

Gebaseerd op:
Database System Concepts, 5th Ed.
©Silberschatz, Korth and Sudarshan



SQL: query taal “met woorden”

- doel: intuïtieve query taal
 - gebruikt Engelse woorden: *select*, *from*, *where*
 - is meer declaratief: je beschrijft de vraag en niet de berekening van het antwoord
 - beschikbaar in (bijna) alle database systemen
 - veel “toeters en bellen” die we in ISO niet bestuderen
 - het “data definitie” deel slaan we in ISO ook over
 - leunt aan bij wat computers goed kunnen (geen verzamelingen, beperkte data types zoals verschillende soorten getallen en strings)





voor andere datatypes: BLOBs

- wat we niet in SQL kunnen weergeven moeten we BLOBs gebruiken: **binary large objects** ze hebben verder ... geen betekenis



de SQL basis query structuur

- een typische SQL query ziet er als volgt uit:

```
select  $A_1, A_2, \dots, A_n$   
from  $r_1, r_2, \dots, r_m$   
where  $P$ 
```

- A_i is een attribuut
- R_i is een relatie (tabel)
- P is een logisch predicat.

- het resultaat betekent:

$$\prod_{A_1, A_2, \dots, A_n} (\sigma_P(r_1 \times r_2 \times \dots \times r_m))$$

- de query werkt op een instantie en produceert een nieuwe (virtuele) tabel-instantie



voorbeeld:

- geef het rekeningnummer en saldo van alle rekeningen bij het filiaal “Perryridge”

- in de algebra was dit:

$\Pi_{\text{account_number, balance}}(\sigma_{\text{branch_name}=\text{“Perryridge”}}(\text{account}))$

- in SQL wordt dit:

```
select account_number, balance
from account
where branch_name = “Perryridge”
```

- merk op dat dit bijna leest als Engels: “select the *account number* and *balance* of the *accounts* where the *branch name* is “Perryridge”



SQL: variaties op de *select* clause

- een sterretje geeft aan dat er geen projectie is (dat alle attributen behouden blijven):

```
select *
from loan
```

- in de **select** clause mag *gerekend* worden (+, −, *, /) met attributen en constanten

- de vraag:

```
select loan_number, branch_name, amount * 100
from loan
```

geeft een tabel die lijkt op *loan* maar met alle bedragen maal 100.





de **where** clause

- de **where** clause geeft een voorwaarde aan *per rij* uit de tabel (of cartesisch product)
 - komt helemaal overeen met het selectie-predicaat uit de relationele algebra.
 - we gebruiken **and**, **or**, **not** en haakjes, en we schrijven $<>$, $<=$ en $>=$ in plaats van \neq , \leq en \geq .
 - we mogen ook in de **where** clause rekenen:

```
select loan_number
from loan
where amount / 100 > 10
```



de **where** clause: bijzondere gevallen

- SQL kent een **between** vergelijking:
 - ```
select loan_number
from loan
where amount between 90000 and 100000
```
- SQL kent *string* operaties:
  - %: stelt eender welke string voor  
vb: **where** branch **like** '%idge%'
  - \_: stelt eender welk (enkel) teken voor  
vb: **where** customer\_name **like** '\_'. De Bra' selecteert De Bra met eender welke voorletter.
  - alleen exacte string vergelijking kan met "=".





## de from clause

- de **from** clause bevat de relaties waarvan het cartesisch product genomen wordt.
  - vb: geef klant-naam, leningnummer en leningbedrag van alle klanten met leningen bij het filiaal Perryridge:

```
select customer_name, borrower.loan_number, amount
from borrower, loan
where borrower.loan_number = loan.loan_number
and branch_name = 'Perryridge'
```

- merk op dat we dezelfde impliciete hernoeming gebruiken als bij de relationele algebra



## expliciete hernoeming

- naast impliciete hernoeming kent SQL ook expliciete hernoeming van tabellen en van attributen: de **as** clause:

*old-name as new-name*

- geef klant-naam, leningnummer en leningbedrag van alle klanten met leningen bij het filiaal Perryridge en noem het *loan\_number* om tot *loan\_id*:

```
select customer_name, b.loan_number as loan_id,
 amount
from borrower as b, loan as l
where b.loan_number = l.loan_number
```

- we noemen een hernoemde tabel ook *tupel variabele*





## hernoeming (vervolg)

- voorbeeld met nodige hernoeming:  
geef de namen van alle filialen met grotere  
“assets” dan een filiaal in Brooklyn:

```
select T.branch_name
from branch as T, branch as S
where T.assets > S.assets
and S.branch_city = 'Brooklyn'
```

- “groter... dan een filiaal in Brooklyn” betekent niet hetzelfde als “groter... dan elk filiaal in Brooklyn”.



## verzamelings-operaties

- de operaties  $\cup$ ,  $\cap$ ,  $-$  uit de relationele algebra heten in SQL **union**, **intersect**, en **except**.
  - de queries die moeten gecombineerd worden moeten tussen haakjes staan
  - hoewel SQL in het algemeen een “probleem” heeft met het elimineren van dubbele tupels gebeurt dat elimineren bij union, intersect en except altijd automatisch (zodat er geen vergissingen ontstaan)





## verzamelings-operaties

- geef alle klanten met een rekening of een lening (of allebei):

```
(select customer_name from depositor)
union
(select customer_name from borrower)
```

- geef alle klanten met een rekening en een lening:

```
(select customer_name from depositor)
intersect
(select customer_name from borrower)
```

- geef alle klanten met een rekening maar geen lening:

```
(select customer_name from depositor)
except
(select customer_name from borrower)
```



## opgaven

- we stellen dezelfde queries als bij de relationele algebra, op de database van opgave 2.1:

*employee* (person\_name, street, city)

*works* (person\_name, company\_name, salary)

*company* (company\_name, city)

*manages* (person\_name, manager\_name)

de sleutels hebben hierbij belang!





## opgaven

- Stel volgende vragen in SQL:
  1. geef de namen van alle bedienden die wonen in Eindhoven
  2. geef de namen van alle bedienden die niet in Eindhoven wonen
  3. geef de namen van alle bedienden die zichzelf als manager hebben
  4. geef de naam van de managers met een salaris van meer dan 100.000
  5. geef de naam van de bedienden met een manager met een salaris van meer dan 100.000



## opgaven

- Stel volgende vragen in SQL:
  6. geef de namen van de bedienden die meer verdienen dan hun manager
  7. geef de naam van de bedrijven die gevestigd zijn in een stad waar nog een ander bedrijf gevestigd is
  8. geef de namen van alle bedienden die wonen in de stad waar ze werken
  9. geef de namen van alle bedienden die wonen in een andere stad dan hun manager
  10. geef de naam van de bedrijven die gevestigd zijn in een stad waar geen enkele bediende van dat bedrijf woont







## opgaven

- Stel volgende vragen in SQL:
  11. geef de naam van bedienden wiens manager voor een ander bedrijf werkt dan zij zelf
  12. geef de namen van de bedrijven die werknemers hebben die in Eindhoven wonen
  13. geef de namen van de bedrijven die geen werknemers hebben die in Eindhoven wonen
  14. geef de naam van de bediende met het hoogste salaris
  15. geef de naam van de manager met het hoogste salaris



## terugvertaling naar het Nederlands

- **select** e.person\_name  
**from** employee **as** e, manages **as** m employee **as** me  
**where** e.person\_name = m.person\_name  
    **and** m.manager\_name = me.person\_name  
    **and** e.city = me.city
  
- ( **select** c.company\_name **from** company **as** c )  
**except**  
( **select** c.company\_name  
**from** company **as** c, works **as** w, employee **as** e  
**where** e.person\_name = w.person\_name  
    **and** e.city = c.city )





## geneste of ingenestelde queries

- in SQL kunnen queries binnen andere queries worden gebruikt
- *subqueries* worden o.a. gebruikt om  $\in$  en  $\notin$  uit te drukken (met **in** en **not in**):
  - geef de klanten met een rekening maar geen lening:

```
select customer_name (select
from depositor customer_name
where customer_name from depositor)
not in except
(select customer_name (select
from borrower) customer_name
 from borrower)
```



## waarom geneste queries?

- bij een geneste query kunnen in de subquery attributen uit hoofd- en subquery worden gebruikt:
  - geef de klanten met een rekening waarop het saldo hoger is dan het bedrag van elk van hun leningen:

```
select customer_name
from depositor as d, account as a
where d.account_number = a.account_number
and customer_name not in
(select customer_name
 from borrower as b, loan as l
 where b.loan_number = l.loan_number
 and l.amount >= a.balance)
```





## meer mogelijkheden...

- dit lijkt op dezelfde vraag...
- maar is dit echt dezelfde query?

```
select customer_name
from depositor as d
where customer_name not in
(select b.customer_name
 from borrower as b, loan as l , account as a
 where d.account_number = a.account_number
 and b.loan_number = l.loan_number
 and l.amount >= a.balance)
```



## meer mogelijkheden, meer fouten...

- dit lijkt opnieuw op dezelfde vraag...
- maar wat betekent deze query?

```
select customer_name
from depositor as d
where customer_name not in
(select b.customer_name
 from borrower as b, loan as l ,
 depositor as dd, account as a
 where d.customer_name = dd.customer_name
 and dd.account_number = a.account_number
 and b.loan_number = l.loan_number
 and l.amount >= a.balance)
```





## scoping regels

- in de subquery mag worden verwezen naar de hoofdquery, maar niet omgekeerd
  - wat is er fout in de volgende query?

```
select customer_name
from depositor as d, account as a
where d.account_number = a.account_number
and d.customer_name = b.customer_name
and customer_name not in
(select customer_name
from borrower as b, loan as l
where b.loan_number = l.loan_number
and l.amount >= a.balance)
```



## verzamelingen vergelijken: some

- we kunnen =, <>, >, >=, <, <= met **some** gebruiken (**in** is hetzelfde als = **some**):
  - geef de (namen van) klanten met een rekening waarvan het saldo groter is dan het bedrag van een van hun leningen:

```
select customer_name
from depositor as d, account as a
where d.account_number = a.account_number
and balance > some
(select amount
from borrower as b, loan as l
where b.customer_name = d.customer_name
and b.loan_number = l.loan_number)
```
  - verschil (in Engels) tussen “some” en “any” !!





## some : beperkt gebruik

- je mag alleen een attribuutwaarde met een subquery vergelijken.

- dit is OK:

```
select account_number
from account
where balance > some (select amount
 from loan)
```

- dit is **niet OK**:

```
select customer_name
from depositor as d, account as a
where d.account_number = a.account_number
and some balance > (select amount
 from loan)
```



## exacte betekenis van some

- $F \langle \text{comp} \rangle \text{some } r \Leftrightarrow \exists t \in r \text{ zodat } (F \langle \text{comp} \rangle t)$   
waarbij  $\langle \text{comp} \rangle <, \leq, >, \geq, =, \text{ of } \neq$  is

(5 < **some**

|   |
|---|
| 0 |
| 5 |
| 6 |

) = waar (lees: 5 < sommige waarden in de tabel)

(5 < **some**

|   |
|---|
| 0 |
| 5 |

) = onwaar

(5 = **some**

|   |
|---|
| 0 |
| 5 |

) = waar

(5  $\neq$  **some**

|   |
|---|
| 0 |
| 5 |

) = waar (want  $0 \neq 5$ )

(= **some**)  $\equiv$  in maar, ( $\neq$  **some**)  $\neq$  not in





## verzamelingen vergelijken: *all*

- we kunnen =, <>, >, >=, <, <= met **all** gebruiken (**not in** is hetzelfde als <> **all**):
  - geef de (namen van) klanten met een rekening waarvan het saldo groter is dan de bedragen van al hun leningen:  

```
select customer_name
from depositor as d, account as a
where d.account_number = a.account_number
and balance > all
(select amount
 from borrower as b, loan as l
 where b.customer_name = d.customer_name
 and b.loan_number = l.loan_number)
```
  - vergelijk (in Engels) “all” en “any” !!



## exacte betekenis van *all*

- $F <comp> \mathbf{all} r \Leftrightarrow \forall t \in r (F <comp> t)$

$$(5 < \mathbf{all} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{onwaar}$$

$$(5 < \mathbf{all} \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{waar}$$

$$(5 = \mathbf{all} \begin{array}{|c|} \hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{onwaar}$$

$$(5 \neq \mathbf{all} \begin{array}{|c|} \hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{waar (want } 5 \neq 4 \text{ en } 5 \neq 6)$$

( $\neq \mathbf{all}$ )  $\equiv$  **not in**  
 maar, ( $= \mathbf{all}$ )  $\neq$  **in**





## test of een subquery-resultaat leeg is

- een **exists** clause geeft de waarde **waar** terug als de subquery een niet-leeg resultaat oplevert.
- **exists**  $r \Leftrightarrow r \neq \emptyset$
- **not exists**  $r \Leftrightarrow r = \emptyset$
- **select** *customer\_name*  
**from** *borrower* **as** *b*  
**where not exists**  
( **select** \*  
**from** *depositor* **as** *d*  
**where** *d.customer\_name* = *b.customer\_name* )



## Opgaven

- Stel volgende vragen in SQL:
  1. geef de namen van klanten die een rekening hebben die ze niet gemeenschappelijk hebben met iemand anders
  2. geef de namen van klanten die geen rekening gemeenschappelijk hebben met iemand anders
- Wat betekent de volgende query:
  3. **select** *balance*  
**from** *account*  
**where** *balance* > all  
( **select** *amount*  
**from** *loan*, *borrower*, *customer*  
**where** *loan.loan\_number* = *borrower.loan\_number*  
**and** *borrower.customer\_name* = *customer.customer\_name*  
**and** *customer\_city* = 'Eindhoven' )





## aggregatie-functies in SQL

- je kunt een “berekening” uitvoeren over de waarden van een attribuut uit verschillende tupels
  - **avg** berekent het gemiddelde (van getallen)
  - **sum** berekent de som (van getallen)
  - **min** neemt het minimum (van getallen, of alfabetisch eerste bij strings)
  - **max** neemt het maximum (van getallen, of alfabetisch laatste bij strings)
  - **count** telt het aantal elementen in de tabel of het “groepje”
- de berekening werkt op een lijst, niet een verzameling (dus dubbels blijven behouden)



## voorbeeld van aggregatie

- geef de som van de saldi van alle rekeningen bij filialen uit Eindhoven:
  - **select** *sum(balance)*  
**from** *account as a, branch as b*  
**where** *a.branch\_name = b.branch\_name*  
**and** *b.branch\_city = 'Eindhoven'*
- geef de naam van de klant(en) met de hoogste lening:
  - **select** *customer\_name*  
**from** *borrower as b, loan as l*  
**where** *b.loan\_number = l.loan\_number*  
**and** *l.amount in*  
**( select** *max(amount)*  
**from** *loan )*







## aggregatie kan ook in een subquery

- geef de klanten die meer dan 10000 aan saldo hebben op al hun rekeningen samen
  - **select** *d.customer\_name*  
**from** *depositor as d*  
**where** 10000 < **some**  
( **select** **sum**(*a.balance*)  
**from** *depositor as dd, account as a*  
**where** *dd.account\_number = a.account\_number*  
**and** *dd.customer\_name = d.customer\_name* )



## opletten met scoping regels!

- geef de klanten, samen met het totaal van de saldi op hun rekeningen
  - **select** *d.customer\_name, sum(a.balance)*  
**from** *depositor as d*  
**where** *a.account\_number in*  
( **select** *a.account\_number*  
**from** *depositor as dd, account as a*  
**where** *dd.account\_number = a.account\_number*  
**and** *dd.customer\_name = d.customer\_name* )
- dit is dus fout: *a* wordt gebruikt buiten de scope van de subquery!





## groepjes vormen met *group by*

- geef de klanten, samen met het totaal van de saldi op hun rekeningen
  - **select** *d.customer\_name*, **sum**(*a.balance*)  
**from** *depositor* **as** *d*, *account* **as** *a*  
**where** *d.account\_number* = *a.account\_number*  
**group by** *d.customer\_name*
  - we zetten (zie **from** en **where**) klanten en *hun* rekeningen eerst naast elkaar
  - we maken dan groepjes van rijen die bij eenzelfde klant horen
  - we tonen dan de klant en de som van de saldi, berekend per groepje



## groepjes selecteren met *having*

- geef de klanten, samen met het totaal van de saldi op hun rekeningen, als dat totaal groter is dan 10000.
  - **select** *d.customer\_name*, **sum**(*a.balance*)  
**from** *depositor* **as** *d*, *account* **as** *a*  
**where** *d.account\_number* = *a.account\_number*  
**group by** *d.customer\_name*  
**having** **sum**(*a.balance*) > 10000
  - **where** werkt op afzonderlijke tupels
  - **having** werkt op groepjes van tupels





## opgaven

- Stel de volgende vragen in SQL:
  1. geef de klanten met twee of meer rekeningen (doe dit met en zonder **group by**)
  2. geef klanten met meer leningen dan rekeningen
- Wat betekent volgende query:
  3. **select sum(a.balance)**  
**from account as a, depositor as d, customer as c**  
**where a.account\_number = d.account\_number**  
**and d.customer\_name = c.customer\_name**  
**and c.customer\_city = 'Eindhoven'**



## opgaven (employee database, opg. 2.1)

16. geef (de namen van) de bedrijven waar geen enkele manager werkt.
17. geef de steden waar geen enkele manager woont.
18. geef (de namen van) de bedienden die meer verdienen dan elke manager.
19. geef (de namen van) de bedienden die meer verdienen dan elke manager die bij het bedrijf van die bediende werkt.
20. geef (de namen van) de bedrijven die alleen maar bedienden uit de vestigingsplaats van het bedrijf hebben.





## opgaven (employee database, opg. 2.1)

21. geef (de naam van) de manager met het hoogste salaris.
22. geef (de naam van) de bedrijven waar bedienden werken met een manager met het hoogste salaris.
23. geef (de naam van) de bedrijven waar de managers met het hoogste salaris werken.
24. maak een lijst van bedrijfsnamen met per bedrijf het gemiddelde salaris bij dat bedrijf.
25. maak een lijst van (de namen van) de bedrijven waar het gemiddelde salaris hoger is dan 20.000, en geef per bedrijf dat gemiddelde salaris.



## opgaven (employee database, opg. 2.1)

26. maak een lijst van (de namen van) de bedrijven waar het gemiddelde salaris hoger is dan 20.000.
27. maak een lijst van bedrijfsnamen met per bedrijf de totale salarislust van dat bedrijf.
28. maak een lijst van bedrijven met per bedrijf het aantal personeelsleden.
29. geef de stad waar de meeste managers wonen.
30. maak een lijst van steden met per stad het aantal personeelsleden bij bedrijven die in die stad gevestigd zijn.
31. maak een lijst van steden met per stad het aantal personeelsleden (van bedrijven uit de database) dat in die stad woont.

