Cursus Databanken

***Gewone* functies**

Aggregaat-functies

GROUP BY + HAVING

Docenten:

Damien Decorte
Wim Goedertier
Luc Vervoort
Tom Vande Wiele
Eric Juchtmans

HO GENT

Functies binnen SQL

In een SQL-instructie kan je allerlei **handige functies** gebruiken.

In verband met functies zijn er wel nogal **veel verschillen** tussen MySQL, Oracle, Microsoft SQL Server, PostgreSQL, etc…

Raadpleeg dus steeds de documentatie:

- MySQL Reference Manual → Functions and Operators
  (https://dev.mysql.com/doc/refman/8.4/en/functions.html)
- w3schools → MySQL Functions
  (https://www.w3schools.com/sql/sql_ref_mysql.asp)

Functies binnen SQL

De belangrijkste soorten zijn:

- Numerieke functies (numeric functions)
- Datum- en tijdfuncties (date and time functions)
- Stringfuncties (string functions)

Numerieke functies

```
ROUND(12.3456 , 2)
FLOOR(3.7) CEIL(3.2)
MOD( 7 , 3 )
7 % 3
POWER( 2 , 3 )
RAND()
PI()
SQRT(…), SIN(…), …
```

probeer dit uit met bvb:

```
SELECT ROUND(12.3456,2);
```

zoek meer info met:

```
\help ROUND
```
(in de Shell)

zoek in de Manual, zoek op W3Schools

Numerieke functies

`ROUND(12.3456 , 2)`	rond af, hier tot **2** cijfers na de komma
`FLOOR(3.7)  CEIL(3.2)`	rond af naar beneden, rond af naar boven
`MOD( 7 , 3 )`	modulo-functie : de ***rest*** van de deling 7 : 3
`7 % 3`	% is de modulo-***operator***: idem modulo-functie
`POWER( 2 , 3 )`	2 tot de 3^{de} macht = 2^3 = 2 * 2 * 2 = 8
`RAND()`	random getal tussen 0 en 1 (0 inbegrepen, 1 niet)
`PI()`	geeft als waarde 3.1415926…
`SQRT(…), SIN(…), …`	vierkantswortel, sinus, …

Dit is uitbreidingsleerstof (niet voor test/examen)

Numerieke functies

Opdrachten:

1.
Toon employee_id, last_name, salary en “new salary” van alle werknemers. Bereken “new salary” als “salary” + 2,3 % en rond af tot 2 cijfers na de komma.

2.
Toon alle info van de werknemers
die een employee_id hebben dat deelbaar is door 3

Numerieke functies

Oplossingen:

1. Toon employee_id, last_name, salary en “new salary” van alle werknemers.
Bereken “new salary” als “salary” + 2,3 % en rond af tot 2 cijfers na de komma.

```
SELECT employee_id, last_name,
       salary, ROUND(salary*1.023,2) AS 'new salary'
FROM employees;
```

2. Toon alle info van de werknemers
die een employee_id hebben dat deelbaar is door 3

```
SELECT * FROM employees
WHERE (employee_id % 3) = 0;
```

haakjes zijn handig, maar niet *nodig*

Numerieke functies

COUNT(), SUM(), AVG(), MIN(), MAX() zijn ook *numerieke* functies, maar werken over **meerder rijen**

→ het zijn **aggregerende functies** (aggregate functions). (zie verder)

Opmerking:

Functienamen zijn *niet* hoofdlettergevoelig

Op veel plaatsen zie je functies systematisch in hoofdletters (MySQL documentatie, MySQL Shell, W3Schools, ...)

Op andere plaatsen staan ze systematisch in kleine letters (DBeaver, ...)

Datum- en tijdfuncties

```
CURDATE(),CURTIME(),NOW()
MONTH('2022-12-31 23:59:41')
YEAR(…),DAY(…),HOUR(…), …
MONTHNAME('2022-8-16')
DAYNAME('2022-05-16')
DATEDIFF(…,…), ADDDATE(…,…)
WEEKDAY(…)
DATE_FORMAT(NOW(),'%a, %l %p')
STR_TO_DATE('May 10 2017','%M %d %Y');
MAKEDATE(…,…), MAKETIME(…,…,…)
```

probeer dit uit met bvb:

```
SELECT MONTH(NOW());
```

zoek meer info met bvb:

```
\help MONTH
```

online MySQL Manual

W3Schools

Datum- en tijdfuncties

`CURDATE(),CURTIME(),NOW()`	huidige datum, tijd, datum+tijd
`MONTH('2022-12-31 23:59:41')`	haalt maand uit datum, hier “12”
`YEAR(…),DAY(…),HOUR(…), …`	haalt jaar, dag, uur, ... uit datum
`MONTHNAME('2022-8-16')`	naam v/d maand, hier “August”
`DAYNAME('2022-05-16')`	naam v/d dag, hier “Monday”
`DATEDIFF(…,…), ADDDATE(…,…)`	rekenen met datums (zie manual)
`WEEKDAY(…)`	geeft getal: 0=ma, 1=di, …, 6=zo
`DATE_FORMAT(now(),'%a, %l %p')`	speciale formaten (zie manual)
`STR_TO_DATE('May 10 2017','%M %d %Y');`	
`MAKEDATE(…,…), MAKETIME(…,…,…)`	maakt datum, tijd (zie manual)

Dit is uitbreidingsleerstof (niet voor test/examen)

Datum- en tijdfuncties

Opdrachten:

1.

Toon alle info van alle werknemers, met een extra kolom met daarin het aantal dagen dat het geleden is dat ze zijn aangeworven (hired).

2.

Toon voornaam, naam en “hire date” van alle werknemers die in de maand juni (June) werden aangeworven.

Datum- en tijdfuncties

Oplossingen:

1. Toon alle info van alle werknemers, met een extra kolom met daarin het aantal dagen dat het geleden is dat ze zijn aangeworven (hired).

```
SELECT *,DATEDIFF(NOW(),hire_date)
FROM employees;
```

2. Toon voornaam, naam en “hire date”
van alle werknemers die in de maand juni (June) werden aangeworven.

```
SELECT first_name, last_name, hire_date
FROM employees WHERE MONTHNAME(hire_date)='June';
```

Stringfuncties

```
LENGTH('hallo')
CONCAT('abc',25,'def')
UPPER(…), LOWER(…)
LTRIM(…), RTRIM(…), TRIM(…)
REPLACE('abcdbc','bc','Q')
SUBSTR('abcde',2,3)
LOCATE('bc','abcd')
SUBSTRING_INDEX('a:bc:d',':',2);
SOUNDEX('hallo')
```

opletten met copy/paste !!

slides, online info, ... bevatten soms "smart"-quotes of *speciale* kopeltekens of verborgen karakters

→ geeft rare SYNTAX-errors

Stringfuncties

```
LENGTH('hallo')                -- geeft het C karakters (-> 5)
CONCAT('abc',25,'def')         -- plakt strings aan elkaar (+ converteert evt.) (-> 'abc25def')
UPPER(...), LOWER(...)         -- zet om in hoofdletters of kleine letters
LTRIM(...), RTRIM(...), TRIM(...) -- haalt spaties weg (vooraan, achteraan, voor én achter)
REPLACE('abcdbc','bc','Q')     -- vervangt elke string 'bc' door de string 'Q' (-> 'aQdQ')
SUBSTR('abcde',2,3)            -- start bij positie 2, selecteer 3 letters (-> 'bcd')
                               -- OPGELET: in C# start positie bij 0, bij MySQL bij 1
LOCATE('bc','abcd')            -- vindt de positie van 'bc' in 'abcd' (-> 2) – start ook bij 1
SUBSTRING_INDEX('a:bc:d',':',2);  -- toont eerste 2 velden met ':' als delimiter (-> 'a:bc')
SOUNDEX(string)                -- geeft code op basis van uitspraak
                               -- zelfde code = gelijkaardig uitgesproken
```

Dit is uitbreidingsleerstof (niet voor test/examen)

Aggregerende functie: SUM()

Een "gewone" functie wordt op elke aparte rij toegepast:

```
SELECT round(salary,1) FROM employees;
```

Een "aggregerende" functie **combineert** gegevens
van **verschillende rijen** tot één resultaat

```
SELECT sum(salary) FROM employees;
```

	sum(salary)
▶	175500.00

Merk op: er mag geen spatie staan tussen functienaam en openend haakje:

```
SELECT sum (salary) FROM employees;
```
→ foutmelding

Opgelet: aggregerende functies NIET combineren met kolomnamen

Doe dit nooit!!

```
SELECT first_name , sum(salary)
FROM employees;
```

"first_name" zou voor elke rij iets anders geven,
maar "sum()" zou alle rijen moeten combineren tot één enkele rij.

Vroeger (vóór 2020) gaf MySQL hierbij GEEN foutmelding !!
Ondertussen gelukkig wel.

Dus: In een SELECT-clausule **NOOIT** aggrerende functies **COMBINEREN MET** kolomnamen (behalve bij GROUP BY (zie later))

Functies worden vaak gecombineerd met een kolomalias

```
SELECT SUM(salary) AS 'Totale loonkost'
FROM employees;
```

kolomalias
(met aanhalingstekens omwille van spatie)

```
SELECT SUM(salary)    'Totale loonkost'
FROM employees;
```

kolomalias zonder "AS"

OPDRACHT:

Bereken de totale loonkost van departement 80.

Antwoord: 30'100

Meer aggregerende functies: SUM() - AVG() - MIN() - MAX() - COUNT()

Pas volgende aggregerende functies toe op departement 80
en formuleer daarbij telkens een passende kolomalias:

- SUM(salary) zie vorige slide
- AVG(salary)
- MIN(salary)
- MAX(salary)
- COUNT(salary)

Zet ze allemaal naast elkaar in één SELECT-statement.

Aggregerende functies ***negeren*** NULL-waarden

Bvb: COUNT telt het aantal rijen met **niet-NULL** waarden:

```
SELECT salary, department_id, bonus FROM employees;

SELECT COUNT(salary), COUNT(department_id),
       COUNT(bonus)
FROM employees;
```

De combinatie met DISTINCT is ook handig

```
SELECT COUNT(DISTINCT department_id) FROM employees;
```

COUNT(*)

COUNT(*) is handig om het aantal rijen van een tabel of resultaat te tellen.

Je bent dan zeker dat er geen enkele rij geskipt wordt omwille van een NULL.

```
SELECT COUNT(*) FROM employees;
```

Alternatief: gebruik COUNT(1) (zelfde effect als COUNT(*))

```
SELECT COUNT(1) FROM employees;
```

Let op met het gebruik van 1 kolom. Rijen met een NULL worden geskipt!!

```
SELECT COUNT(employee_id) FROM employees;
SELECT COUNT(manager_id) FROM employees;
```

AVG() with NULL-values

Rijen met NULL-waarde in deze kolom

tellen niet mee voor de berekening van het gemiddelde.

```
SELECT AVG(commission_pct)
FROM employees;
```

Stemt dus overeen met:

```
SELECT SUM(commission_pct)/COUNT(commission_pct)
FROM employees;
```

NIET TOEGELATEN in WHERE-clausule

Aggregerende functies zijn NIET TOEGELATEN in WHERE-clausule

→ Geeft foutmelding

```
SELECT *
FROM employees
WHERE salary > avg(salary) ;
```

GROUP BY-clausule

Totale loonkost (alle medewerkers):

```
SELECT SUM(salary) FROM employees;
```

Loonkost voor departement 80:

```
SELECT SUM(salary) FROM employees
                   WHERE department_id = 60 ;
```

Loonkost ***per*** departement:

```
SELECT SUM(salary) FROM employees
                   GROUP BY department_id ;
```

GROUP BY steeds ***in combinatie met*** aggregerende functie

GROUP BY-clausule

Loonkost ***per*** departement:

```
SELECT SUM(salary) FROM employees
                   GROUP BY department_id ;
```

Handiger om ook **department_id** te tonen bij iedere loonkost:

```
SELECT department_id, SUM(salary)
                      FROM employees
                      GROUP BY department_id ;
```

OPGELET:

Bij aggregaatsfuncties, in de SELECT-clausule,
zijn ***enkel*** kolomnamen toegelaten die ook in de GROUP BY staan!!

GROUP BY-clausule

Opdracht:

Maak een resultatenlijst met het gemiddelde loon per job-functie (job_id).

Gebruik "gemiddelde loon" als kolomnaam.

job_id	gemiddeld loon
AC_ACCOUNT	8300.000000
AC_MGR	12000.000000
AD_ASST	4400.000000
AD_PRES	24000.000000
AD_VP	17000.000000
IT_PROG	6400.000000
MK_MAN	13000.000000
MK_REP	6000.000000
SA_MAN	10500.000000
SA_REP	8866.666667
ST_CLERK	2925.000000
ST_MAN	5800.000000

GROUP BY-clausule

Wat is hier fout?

```
SELECT department_id, avg(salary)
FROM employees
GROUP BY job_id ;
```

Dit mag niet, maar dit houdt ook geen steek!

GROUP BY-clausule

OPDRACHT:

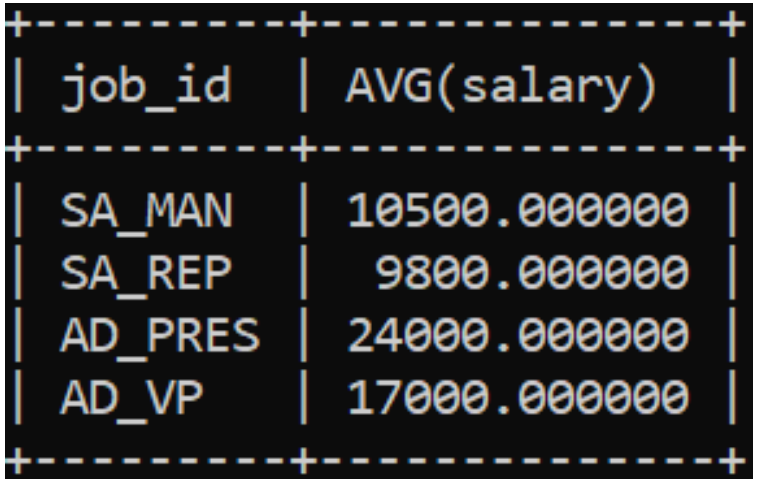
Maak een resultatenlijst met
het gemiddelde loon
binnen elke job-functie,
binnen elk departement

(antw: 13 rows returned)

department_id	job_id	Gemiddeld loon
90	AD_PRES	24000.000000
90	AD_VP	17000.000000
60	IT_PROG	6400.000000
50	ST_MAN	5800.000000
50	ST_CLERK	2925.000000
80	SA_MAN	10500.000000
80	SA_REP	9800.000000
NULL	SA_REP	7000.000000
10	AD_ASST	4400.000000
20	MK_MAN	13000.000000
20	MK_REP	6000.000000
110	AC_MGR	12000.000000
110	AC_ACCOUNT	8300.000000

Tip: “GROUP BY” kan meerdere kolommen bevatten,
gescheiden door komma’s (zoals bij ORDER BY)

GROUP BY-clausule



job_id	AVG(salary)
SA_MAN	10500.000000
SA_REP	9800.000000
AD_PRES	24000.000000
AD_VP	17000.000000

OPDRACHT:

Maak een resultatenlijst met het gemiddelde loon

per job-functie, maar enkel voor departement 80 en 90.

(antw: 4 rows returned)

Tip: Gebruik éérst de WHERE om de juiste departementen te selecteren,
pas daarna de GROUP BY op departement en job-functie

GROUP BY-clausule

Oplossing:

Maak een resultatenlijst met het gemiddelde loon
per job-functie, maar enkel voor departement 80 en 90.

```
SELECT job_id, AVG(salary)
FROM employees
WHERE department_id IN (80,90)
GROUP BY job_id;
```

HAVING: sub-clausule van GROUP BY

Met HAVING kan je, ***na*** de GROUP BY, nog verder filteren op een voorwaarde ***met een aggregaatsfunctie***.

<u>Voorbeeld</u>:

Toon per job-functie het gemiddelde loon

job_id	AVG(salary)
AC_ACCOUNT	8300.000000
AC_MGR	12000.000000
AD_ASST	4400.000000
AD_PRES	24000.000000
AD_VP	17000.000000
IT_PROG	6400.000000
MK_MAN	13000.000000
MK_REP	6000.000000
SA_MAN	10500.000000

maar beperk je tot de job-functies met een gemiddeld loon > 10000

job_id	AVG(salary)
AC_MGR	12000.000000
AD_PRES	24000.000000
AD_VP	17000.000000
MK_MAN	13000.000000
SA_MAN	10500.000000

HAVING: sub-clausule van GROUP BY

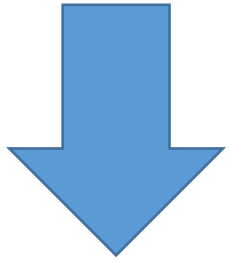
Met HAVING kan je, ***na*** de GROUP BY, nog verder filteren op een voorwaarde ***met een aggregaatsfunctie***.

<u>Voorbeeld</u>:

Toon per job-functie het gemiddelde loon maar beperk je tot de job-functies met een gemiddeld loon > 10000

```
SELECT job_id, AVG(salary
FROM employees
GROUP BY job_id
   HAVING AVG(salary) > 10000;
```

job_id	AVG(salary)
AC_ACCOUNT	8300.000000
AC_MGR	12000.000000
AD_ASST	4400.000000
AD_PRES	24000.000000
AD_VP	17000.000000
IT_PROG	6400.000000
MK_MAN	13000.000000
MK_REP	6000.000000
SA_MAN	10500.000000



job_id	AVG(salary)
AC_MGR	12000.000000
AD_PRES	24000.000000
AD_VP	17000.000000
MK_MAN	13000.000000
SA_MAN	10500.000000

HAVING: sub-clausule van GROUP BY

HAVING kan alleen als onderdeel van een GROUP BY

HAVING is alleen zinvol met een voorwaarde
met een aggregaatsfunctie

Een voorwaarde ***met een aggregaatsfunctie***
is ***verboden*** in een WHERE-clausule

```
SELECT job_id,AVG(salary) FROM employees
WHERE AVG(salary) > 9000
GROUP BY job_id;
```

→ Geeft syntax-error

WHERE : voorwaarde op ***record***-niveau
HAVING : voorwaarde op ***groep***-niveau

```
SELECT job_id , avg(salary) FROM employees
WHERE department_id = 110
GROUP BY job_id;
```

→ “department_id=110” is een voorwaarde op record-niveau

```
SELECT job_id , avg(salary) FROM employees
GROUP BY job_id
    HAVING avg(salary) >= 9000 ;
```

→ “avg(salary) >= 9000” is
een voorwaarde op groep-niveau

Combinatie kan ook:

```
SELECT job_id , avg(salary)
FROM employees
WHERE department_id = 110
GROUP BY job_id
    HAVING avg(salary) >= 9000;
```

HAVING

-- OPDRACHT:

-- maak een resultatenlijst met

-- per departement het aantal medewerkers en de som van alle lonen,

-- toon enkel de departementen waarbij de som meer dan 10.000 bedraagt

-- 6 rows returned

20	2	19000.00
50	5	17500.00
60	3	19200.00
80	3	30100.00
90	3	58000.00
110	2	20300.00

HAVING

-- OPDRACHT:
-- maak een resultatenlijst met
-- het aantal medewerkers per departement,
-- van elk departement met minstens 3 medewerkers
-- met uitzondering van departement 60,
-- gerangschikt volgens het aantal medewerkers

-- 3 rows returned

80	3
90	3
50	5