3. Týždeň

JOIN 1
Agregačné funkcie, GROUP BY, HAVING

1. **Tri základné typy JOIN** http://dev.mysql.com/doc/refman/5.5/en/join.html http://dev.mysql.com/doc/refman/5.6/en/left-join-optimization.html
2. **(Ďalšie typy JOIN)**
3. **NULL hodnoty a OUTER JOIN.**
4. **Agregačné funkcie** http://dev.mysql.com/doc/refman/5.6/en/group-by-functions.html
5. **GROUP BY, HAVING** http://dev.mysql.com/doc/refman/5.5/en/group-by-functions-and-modifiers.html

1) Tri základné typy JOIN

JOIN slúži na získanie dát z dvoch alebo viac tabuliek na základe logických, prepojovacích vzťahov medzi stĺpcami tabuliek. Vieme, že hlavnými prostriedkami pre vertikálnu a horizontálnu filtráciu sú SELECT_zoznam a WHERE klauzula. Hlavnou úlohou JOIN príkazov je vertikálne spájanie atribútov z viacerých tabuliek, ale majú dopad aj na horizontálne spojenie tabuliek.

Typy

a) `CROSS` – `CROSS JOIN`
b) `INNER` – `[INNER] JOIN`
c) `OUTER` – `LEFT/RIGHT [OUTER] JOIN`

MySQL nepodporuje `FULL[OUTER]JOIN`.

Syntax:

```
... FROM T1 join_typ T2 [   ON (join_podmienka)] [WHERE ]
... FROM T1 join_typ T2 [USING (zoznam_stlpcov)] [WHERE ]
```

Celkový výsledok, počet vrátených riadkov JOIN dopytu predovšetkým závisí od:

- jeho typu
- podmienok v `ON`

ale aj napr. od

- podmienok vo `WHERE` klauzule.

Podmienka JOIN:

V podmienke JOIN sa najčastejšie používa porovnávací operátor = (ale je možné použiť aj iné porovnania).

- Porovnávacie podmienky sú najčastejšie založené na dvojici primárny a foreigen (cudzí) kľúč (**PK** a **FK**).
- Čitateľnosť kódu zvyšuje použitie aliasov pre tabuľky, z kadiaľ sú stĺpce.

Typ JOIN-u ovplyvňuje výsledok.

join_typ:

a)
```
... FROM T1 CROSS JOIN T2 ...
⇔ ... FROM T1, T2 ...
```

– CROSS JOIN vráti Karteziánsky súčin hodnôt dvoch stĺpcov - ku každému riadku jednej tabuľky pridá všetky riadky druhej.

b)
```
... FROM T1       JOIN T2 ON ...
... FROM T1 INNER JOIN T2 ON ...
```

– INNER JOIN vráti iba také riadky, ktorých zodpovedajúce stĺpcové hodnoty vyhovujú podmienke v ON klauzule, teda pre každý vrátený riadok platí, že k aktuálnemu riadku T1 existuje riadok T2, pre ktoré ON podmienka je splnená. Ostatné riadky eliminuje. Ak ON klauzula používa porovnávací operátor =, potom INNER JOIN vráti iba také riadky, v ktorých hodnoty atribútov ON klauzuly sa nachádzajú v oboch tabuľkách (horizontálny prienik dvoch tabuliek).

```
c) ... FROM T1 out_typ OUTER JOIN T2 ON ...

       out_typ:
       c1) LEFT
       c2) RIGHT
      [c3) FULL] s UNION
          ... FROM T1 LEFT  OUTER JOIN T2 ON ...
          ... FROM T1 LEFT  JOIN T2 ON ...
          ... FROM T2 RIGHT JOIN T1 ON ...
```

- T1 LEFT OUTER JOIN T2 určite vráti všetky riadky z tabuľky T1 (môže sa stať, že vráti všetky riadky aj z T2).

- Ak riadok z T2 nevyhovuje podmienke v ON klauzule, do jej stĺpcov sa zapíšu NULL hodnoty.

[FULL OUTER JOIN vráti všetky riadky oboch tabuliek – MS SQL Server]

Syntax ešte raz:

1)
```
SELECT * FROM T1 CROSS JOIN T2
```

2)
```
SELECT * FROM T1 JOIN T2 ON ...
SELECT * FROM T1 INNER JOIN T2 ON ...
```

3)
```
SELECT * FROM T1 LEFT JOIN T2 ON ...
SELECT * FROM T1 LEFT OUTER JOIN T2 ON ...
```

Ku každému riadku T1 môže pridať riadky z T2

```
SELECT * FROM T1 RIGHT JOIN T2 ON ...
SELECT * FROM T1 RIGHT OUTER JOIN T2 ON ...
```

T1			T2	
id1	**x**		**id2**	**a**
1	x		1	a
2	y		2	b
			3	c
			4	d

```
USE dbmaz;

DROP TABLE IF EXISTS T1;
CREATE TABLE IF NOT EXISTS T1( id1 INT, x CHAR(1) );
INSERT T1 VALUES(1, 'x');
INSERT T1 VALUES(2, 'x');  ### INSERT T1 VALUES(null, 'z');

DROP TABLE IF EXISTS T2;
CREATE TABLE IF NOT EXISTS T2( id2 INT, a CHAR(1) );
INSERT T2 VALUES(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd');
```

Neštandardné (1=2, 1=1 resp. false, true, 0,1 [v Microsoft SQL Server nie]) a štandardné použitie JOIN:

CROSS a INNER JOIN:

```
-- Ku kazdemu riadku T1 prida celu T2:

SELECT * FROM T1, T2;                     -- 8r
SELECT * FROM T1 CROSS JOIN T2;           -- 8r <=>
SELECT * FROM T1 STRAIGHT_JOIN T2;        -- 8r <=>
SELECT * FROM T1 INNER JOIN T2 ON (1=1); -- 8r <=>
SELECT * FROM T1 JOIN T2 ON (true);       -- 8r <=>
SELECT * FROM T1 JOIN T2;                 -- 8r <=>
SELECT * FROM t1 NATURAL JOIN t2;

SELECT * FROM T1 JOIN T2 ON (T2.a = 'a' OR T2.a = 'c');-- 4r
SELECT * FROM T1 JOIN T2 ON (T2.id2 = T1.id1);          -- 2r
       #WHERE (T2.a = 'a' OR T2.a = 'c');        -- 1r
```

OUTER JOIN:

```
SELECT * FROM T1 LEFT OUTER JOIN  T2 ON 1 = 2;
```

	id1	x	id2	a
1	1	x	NULL	NULL
2	2	y	NULL	NULL

```
SELECT * FROM T1 RIGHT JOIN T2 ON 0;
```

	id1	x	id2	a
1	NULL	NULL	1	a
2	NULL	NULL	2	b
3	NULL	NULL	3	c
4	NULL	NULL	4	d

FULL JOIN:

```
SELECT * FROM T1 LEFT OUTER JOIN  T2 ON 1 = 2
UNION
SELECT * FROM T1 RIGHT JOIN T2 ON 1 = 2;
```

	id1	x	id2	a
1	1	x	NULL	NULL
2	2	y	NULL	NULL
3	NULL	NULL	1	a
4	NULL	NULL	2	b
5	NULL	NULL	3	c
6	NULL	NULL	4	d

```
SELECT * FROM T1 LEFT JOIN T2
          ON (T2.a = 'a' OR T2.a = 'c');
```

id1	x	id2	a
1	x	1	a
2	y	1	a
1	x	3	c
2	y	3	c

```
SELECT * FROM T1 RIGHT JOIN T2
          ON (T2.a = 'a' OR T2.a = 'c');
```

id1	x	id2	a
1	x	1	a
1	x	3	c
2	y	1	a
2	y	3	c
NULL	NULL	2	b
NULL	NULL	4	d

Filtrácia

- Filtrácia riadkov: `ON, WHERE, HAVING`
- Filtrácia stĺpcov: `JOIN` vs. zoznam stĺpcov

Odporúča sa dať podmienku spájania tabuliek do `ON` a nie do `WHERE`.

`INNER JOIN` je lepší ako `CROSS JOIN` – optimalizátor implementačne sa sústreďuje na `INNER JOIN`.

3) (Ďalšie typy JOIN) http://msdn.microsoft.com/en-us/library/ms173815.aspx http://msdn.microsoft.com/en-us/library/ms191426.aspx

```
- STRAIGHT_JOIN
- NATURAL JOIN
- INNER LOOP JOIN (Tab1 je veľmi malá)
- INNER MERGE JOIN (Tab1 je usporiadaná)
- LEFT OUTER HASH JOIN
```

4) NULL hodnoty a OUTER JOIN - Príklad

Predikát

Predikát je Boolean-hodnotová funkcia P: X → {true, false}.

Predikát je výraz s názvami **atribútov**, **stĺpcov**, ktorý pre danú n-ticu / riadok vráti

TRUE, FALSE alebo NULL/UNKNOWN.

Predikáty sú používané vo vyhľadávacích podmienkach WHERE a HAVING klauzúl (častí) dopytu, v JOIN ON podmienkach FROM klauzuly a na ďalších miestach, kde sa očakávajú boolové premenné.

Výsledok porovnávacieho predikátu, ktorý obsahuje NULL je NULL a preto sa INNER JOIN-om riadky s NULL hodnotami v príslušných stĺpcoch ON klauzuly sa nevrátia. Avšak ako sme videli, takéto riadky, teda riadky s NULL hodnotou, je možné vrátiť s OUTER JOIN-mi, čo sa v aplikáciach využíva.

```
SELECT * FROM T1 JOIN T2 ON (T2.id2 = T1.id1); -- 2r
INSERT T1 VALUES(null,'z');
SELECT * FROM T1 JOIN T2 ON (T2.id2 = T1.id1); -- 2r
```

Príklady

Do výsledku dopytu vypíšte aj chýbajúce dátumy z tabuľky pomocou NULL hodnôt

```
USE DBMAZ;
DROP TABLE IF EXISTS Dni;
CREATE TABLE IF NOT EXISTS Dni(id INT, den
DATETIME);
INSERT Dni VALUES(1, '2008-09-01');
INSERT Dni VALUES(2, '2008-09-03');
INSERT Dni VALUES(3, '2008-09-06');
INSERT Dni VALUES(4, '2008-09-09');
SELECT * from Dni;
```

	id	den
1	1	2008-09-01 00:00:00.000
2	2	NULL
3	3	2008-09-03 00:00:00.000
4	4	NULL
5	5	NULL
6	6	2008-09-06 00:00:00.000
7	7	NULL
8	8	NULL
9	9	2008-09-09 00:00:00.000

```
DROP TABLE IF EXISTS DniPocitadlo;
CREATE TABLE IF NOT EXISTS DniPocitadlo(id INT); #NOT NULL PRIMARY KEY);
SELECT * from DniPocitadlo;
```

Potrebujeme ešte dve premenné s hodnotami **2008-08-31** a **2008-09-09**.

```
SET @den2 = (SELECT MAX(d.den) FROM Dni d);
SELECT @den2;
SET @den1 = (SELECT DATE_ADD(MIN(d.den), INTERVAL -1 DAY) FROM Dni d);

SELECT @den1, @den2;
SELECT DATEDIFF(@den2, @den1);
```

Uložená procedúra (stored procedure - SP) v MySQL je obklúčená z dvoch strán s oddelovačom, delimiterom. Nasledujúca SP vytvorí zoznam celých čísel od 1 po 9 do pomocnej tabuľky `DATEDIFF`, ktoré zodpovedajú dňom medzi dvomi dátumami `den1` a `@den2`.

```
drop procedure if exists xx1;
delimiter #
create procedure xx1(den1 DATE, den2 DATE)
begin
    declare i int;
    declare ii int;
    set i=1;
    set ii= ( SELECT DATEDIFF(@den2, @den1) );
    WHILE i <= ii do
            INSERT DniPocitadlo VALUES (i);
            SET i = i + 1;
            END while;
END #
delimiter ;

CALL xx1(@den1, @den2);

=>
### 1a) Dni zadane:
SELECT id, den FROM     dbmaz.Dni;
### 1b) Pomocna tabulka:
SELECT id FROM dbmaz.dnipocitadlo;
```

	id	den
1	1	2008-09-01 00:00:00.000
2	2	2008-09-03 00:00:00.000
3	3	2008-09-06 00:00:00.000
4	4	2008-09-09 00:00:00.000

	id
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

```
### 1c) Dni vsetky - aj chybajuce:
SELECT T1.id, T2.den FROM
     DniPocitadlo T1 LEFT OUTER JOIN
              Dni T2 ON
                  T2.den = DATE_ADD(@den1, INTERVAL T1.id DAY)
      ORDER BY T1.id;
```

	id	den
1	1	2008-09-01 00:00:00.000
2	2	NULL
3	3	2008-09-03 00:00:00.000
4	4	NULL
5	5	NULL
6	6	2008-09-06 00:00:00.000
7	7	NULL
8	8	NULL
9	9	2008-09-09 00:00:00.000

```
### 1d_a) Iba dni chybajuce:
SELECT T1.id, T2.id FROM
  DniPocitadlo T1 LEFT OUTER JOIN Dni T2
            ON T2.den = DATE_ADD(@den1, INTERVAL T1.id DAY)
WHERE T2.den IS NULL;

### 1d_b) Iba dni-datumy chybajuce:
SELECT T1.id, DATE_ADD(@den1, INTERVAL T1.id DAY) chyba FROM
 DniPocitadlo T1 LEFT OUTER JOIN Dni T2
           ON T2.den = DATE_ADD(@den1, INTERVAL T1.id DAY)
WHERE T2.den IS NULL;
```

	id	id
1	2	NULL
2	4	NULL
3	5	NULL
4	7	NULL
5	8	NULL

	id	chyba
1	2	2008-09-02 00:00:00.000
2	4	2008-09-04 00:00:00.000
3	5	2008-09-05 00:00:00.000
4	7	2008-09-07 00:00:00.000
5	8	2008-09-08 00:00:00.000

http://blogs.technet.com/b/bpaulblog/archive/2010/02/28/simplifying-sql-server-joins-query-and-internals.aspx

5 Agregačné funkcie

Agregačné funkcie vracajú jednu jedinú hodnotu (skalárnu veličinu) na základe množiny hodnôt.

Najzákladnejšie agregačné funkcie sú:

```
MIN, MAX, COUNT, SUM, AVG , VAR_POP, STDEV_POP, ...
```

- Agregačné funkcie majú jeden vstupný argument (z názvov stĺpcov), ale argumentom COUNT môže byť aj *. V tom prípade COUNT vráti počet všetkých riadkov, ináč iba počet nenulových hodnôt (napr. stĺpca)

  `COUNT(*)` vs. `COUNT(vek)`

- Pred argument môžeme doplniť aj služobné slovo DISTINCT.
- Agregačné funkcie okrem COUNT ignorujú NULL (preskočia ich).
- COUNT v konkrétnych stĺpcoch ignoruje NULL ale pre celú tabuľku nie.
- Často sú použité spolu s GROUP BY.
- Funkcie AVG a SUM pracujú iba s numerickými stĺpcami.
- Funkcie MIN a MAX pracujú s numerickými, znakovými (character) a dátumovými stĺpcami.

COUNT, MIN, MAX

```
CREATE TABLE tt(id int, x float);
INSERT tt VALUES
(1,10),(2, null),(3,30),(null, null),(1,20);


SELECT COUNT(*) FROM tt;           # 5
SELECT COUNT(id) FROM tt;          # 4
SELECT COUNT(DISTINCT id) FROM tt;# 3


SELECT MIN(x) FROM tt;             # 10
#SELECT MIN(DISTINCT x) FROM tt; # 10
```

id	x
1	10
2	NULL
3	30
NULL	NULL
1	20

Suma a priemer – SUM, AVG

Pre použitie agregačných funkcií SUM a AVG s DATETIME hodnotami treba sa opierať o pomocné funkcie `SEC_TO_TIME, TIME_TO_SEC a FROM_DAYS, TO_DAYS.`

```
CREATE TABLE ttt(t time, d date);
INSERT ttt VALUES('01:00:00', '2015:10:06'),
                ('02:00:00', '2015:10:07'),
                ('03:00:00', '2015:10:08');
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(t))) FROM ttt; # '06:00:00'
```

Vyskúšajte do time zapísať aj minúty či sekundy.

```
SELECT FROM_DAYS(AVG(TO_DAYS(d))) FROM ttt;   # '2015-10-07'
SELECT MAX(t), MIN(t), MAX(d) FROM ttt;
```

MAX(t)	MIN(t)	MAX(d)
03:00:00	01:00:00	2015-10-08

Výberová disperzia – VAR_POP

```
VAR_POP = VARIANCE ~ VAR_SAMP*(n-1)/n
```

Výberová odchýlka – STDDEV_POP

```
STDDEV_POP() = SQRT( VAR_POP() )  (= STD() = STDDEV())
```

```
### S_n
SELECT sqrt(var_pop(x)), '=', stddev_pop(x)  FROM tt;
# '8.16496580927726', '=', '8.16496580927726'

### S^2_n
SELECT var_pop(x), '=', variance(x), '=',
        var_samp(x)*(count(x)-1)/count(x) FROM tt;
    # '66.66666666666667', '=', '66.66666666666667', '=', '66.66666666666667'
```

Výroková logika vs. predikátová logika prvého rádu

Výroková logika sa v matematickej logike skladá z
- atomických výrokov (teda z tvrdení, ako výrokových premenných), ktoré sú buď T alebo F a
- logických operátorov.

Výroková logika sa na rozdiel od predikátovej logiky prvého rádu nezaoberá
- nelogickými objektami,
- predikátmi o nich a
- kvantifikátormi.

Výroková logika tvorí základ predikátovej logiky prvého rádu.
Pr.
Sneží.
Ak sneží, potom je zamračené.

Predikátová logika prvého rádu na rozdiel od výrokovej logiky môže pracovať aj kvantifikovanými premennými.
Predikátová logika prvého rádu sa skladá z
- nelogických objektov, individuí,
- predikátov (vlastnosti alebo vzťahy) o nich a
- kvantifikátorov (relačný kalkulus).

Pr.
... FROM student WHERE vek = 20 AND znamka > 2 ...
Predikát vek = 20 sa stane T, F alebo null pre každý riadok tabuľky student.

6 GROUP BY, HAVING

a) GROUP BY a sumarizácia

Výsledky dopytu sú zoskupené na základe stĺpcov vymenovaných v klauzule GROUP BY.

1. **Štandard SQL** vyžaduje, aby všetky stĺpce klauzuly SELECT použité v neagregačných výrazoch, boli vymenované v GROUP BY klauzule. Teda, názvy stĺpcov v SELECTe, ktoré nie sú súčasťou agregačného výrazu, musia byť aj v GROUP BY klauzule.
   - Ak nechcete zoskupovať podľa nejakého stĺpca, nedajte ho do zoznamu stĺpcov SELECTu mimo agregácie.
   - Stĺpec môže byť v GROUP BY ale nemusí byť v SELECTe
   - Stĺpec alebo výraz v GROUP BY môže byť aj v ORDER BY
2. Pozor! MySQL *rozširuje* použitie GROUP BY tak, že povoľuje SELECTovať stĺpce, ktoré nie sú uvedené v klauzule GROUP BY. http://dev.mysql.com/doc/refman/5.5/en/group-by-handling.html

Pomocou GROUP BY môžeme získať jednorozmernú tabuľku početností, ale nie len početností, ale aj iných sumárnych veličín. PIVOT je dvojrozmerným rozšírením GROUP BY.

b) HAVING

HAVING spolu s WHERE používame na filtráciu.
HAVING je súčasťou GROUP BY a filtruje výsledok z GROUP BY.
Having vráti riadky, v ktorých agregačná hodnota spĺňa podmienku.
V HAVING môže vystupovať agregácia ľubovoľného stĺpca, ale bez agregácie iba stĺpce, ktoré sú v SELECT zozname.
Agregačné výsledky treba filtrovať pomocou HAVING a nie WHERE.

c) Príklady a 'ONLY_FULL_GROUP_BY'

```
### A) Group By
###
DROP TABLE IF EXISTS T;
CREATE TABLE T(i INT, j INT, k INT);
INSERT INTO T VALUES
(1, 10, 5),
(2, NULL, 4),
(3, 50, 3),
(4, 50, 2),
(5, 10, 1),
(6, 50, 2);
SELECT * FROM T;
```

i	j	k
1	10	5
2	NULL	4
3	50	3
4	50	2
5	10	1
6	50	2

Zoznam všetkých rôznych hodnôt v stĺpci j:

```
SELECT j FROM T GROUP BY j ORDER BY j;
```

j
NULL
10
50

```
SELECT SUM(i) FROM T GROUP BY j ORDER BY j;
```

SUM(i)
2
6
13

```
-- Lepšie takto:
SELECT SUM(i),j FROM T GROUP BY j ORDER BY j;
```

SUM(i)	j
2	NULL
6	10
13	50

```
-- Snáď ešte lepšie takto:
SELECT j, SUM(i) FROM T GROUP BY j ORDER BY j;
```

j	SUM(i)
NULL	2
10	6
50	13

Zdôrazňujeme, že SELECT zoznam, ORDER zoznam alebo HAVING podmienka nemôže sa odvolávať na neagregovaný stĺpec, ktorý nie je vymenovaný v GROUP BY klauzule. MySQL túto podmienku SQL štandardu môže ignorovať kvôli niektorým zriedkavým prípadom.

Správne nastavenie SQL štandardu:

```
SET SESSION sql_mode = 'ONLY_FULL_GROUP_BY';
```

alebo

```
# SET GLOBAL sql_mode = 'ONLY_FULL_GROUP_BY';
SELECT @@sql_mode;
```

správne hlásenie dvoch chýb:

```
SELECT j, SUM(i) FROM T; # nesprávne použitie j
SELECT j, k, SUM(i) FROM T GROUP BY j; # nesprávne použitie k
```

Pozor! **Zrušenie správneho nastavenia:**

```
SET SESSION sql_mode=(SELECT
            REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY',''));
SELECT @@sql_mode;
SELECT j, SUM(i) FROM T;
SELECT j, k, SUM(i) FROM T GROUP BY j;
```

```
### B) Group By  +  Having
###
SELECT j, SUM(i) FROM T GROUP BY j
                                  HAVING j > 10;
```

j	SUM(i)
50	13

```
SELECT j, SUM(i) FROM T GROUP BY j

                                  HAVING SUM(i) > 5;
```

j	SUM(i)
10	6
50	13

```
USE OsobaVztah; # resp. OsobaDB
SELECT avg(vyska), pohlavie FROM osoba
GROUP BY pohlavie;
  -- HAVING avg(vyska)>170;
```

avg(vyska)	pohlavie
175.02000	m
164.19231	z

avg(vyska)	pohlavie
175.02000	m