

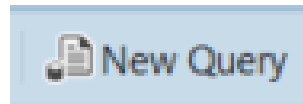
ÚVOD DO SQL

Lektor: **Ing. Michal Konvalinka**

Okno Object Explorer

- Slouží k procházení databází, tabulek a dalších objektů v databázi nebo na serveru
- Každý objekt nabízí specifické kontextové akce (pravé tlačítko myši)

Příkaz New Query otevře nové okno s textovým editorem pro tvorbu dotazů.

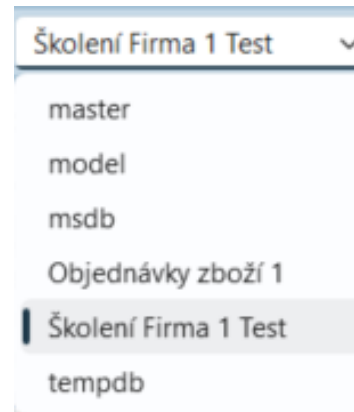


Vytvořit databázi

- CREATE DATABASE skoleni

Přepnout na databázi

- USE skoleni



Příkazy pro:

Definici databázových objektů

- DDL (Data Definition Language)
- příkazy CREATE, DROP, UPGRADE

Manipulaci s daty

- DML (Data Manipulation Language)
- příkazy SELECT, INSERT, UPDATE, DELETE

Řízení přístupů k datům

- DCL (Data Control Language)
- příkazy GRANT, REVOKE

Základní struktura SQL dotazu:

- SELECT

[seznam sloupců nebo * pro všechny sloupce]

- FROM

[tabulka nebo seznam tabulek se spojením]

```
SELECT  
    JMENO,  
    FCE  
FROM ZAM
```

```
SELECT * FROM ODDEL  
SELECT * FROM UKOLY  
SELECT * FROM ZAM
```

SELECT

SELECT definuje seznam atributů (sloupců), které dotaz vrátí.

- Lze použít * pro návrat všech atributů.
- Lze použít TOP (n) pro výběr prvních n řádků:
 - SELECT TOP(100) FROM tab

MySQL má pro výběr n řádků LIMIT:

- SELECT * FROM tab LIMIT 10

Jednotlivé atributy mohou mít přiřazen alias:

SELECT

atr AS [Roční průměr],
(hod1 + hod2) AS Součet,
(napocet * 10) AS Násobek

- AS je nepovinné.

```
SELECT TOP(10)
    JMENO,
    FCE
FROM ZAM
```

Klauzule FROM definuje objekty (tabulky, pohledy), které obsahují podkladová data.

Jednotlivé tabulky mohou mít přiřazen alias:

```
SELECT
```

```
  a.*
```

```
FROM
```

```
  tab a;
```

V SQL skriptu lze použít dva typy komentářů:

- řádkový – od `--` po konec řádku se vše ignoruje
- blokový – od `/*` po
 komentář
 po `*/` se vše ignoruje

`SELECT` -- takto vypadá řádkový komentář

`/*` tento blok se

 nevykoná `*/`

`FROM`

`tab1`

`WHERE`

`atr1 = 'A';`

Aritmetické výrazy:

<code>hod1 + hod2</code>	-- sčítání
<code>hod1 - hod2</code>	-- odčítání
<code>hod1 * hod2</code>	-- násobení
<code>hod1 / hod2</code>	-- dělení

Aritmetické výrazy se vyhodnocují zleva doprava. Pořadí vyhodnocení lze změnit pomocí závorek:

`hod1 * (hod2 + hod3)`

Spojování řetězců se provádí pomocí CONCAT:

`CONCAT('ABC', atr1)`

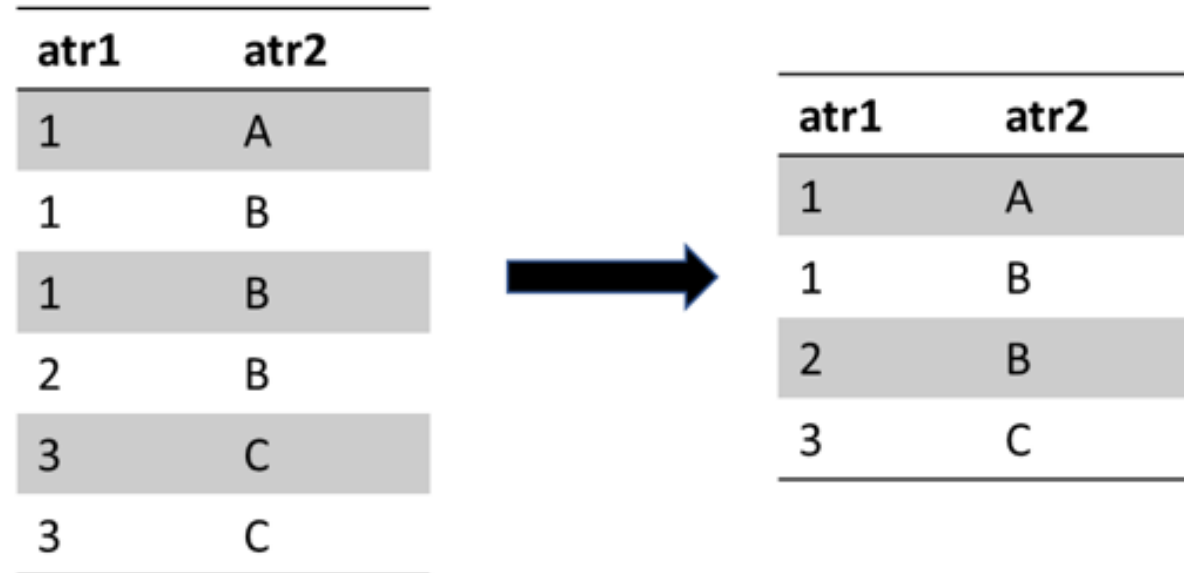
V některých DB systémech `||`, v MSSQL +

`CONCAT_WS (oddělovač, text, text, text)` vloží mezi hodnoty oddělovač

DISTINCT – UNIKÁTNÍ ŘÁDKY

Pro návrat pouze unikátních řádků lze použít klauzuli DISTINCT:

```
SELECT  
  DISTINCT atr1, atr2  
FROM  
  tab1;
```



The diagram illustrates the effect of the DISTINCT clause. On the left, a table with columns 'atr1' and 'atr2' contains six rows, including two identical rows (1, B) and (3, C). A thick black arrow points to the right, where the resulting table is shown. This table contains only the unique rows: (1, A), (1, B), (2, B), and (3, C).

atr1	atr2
1	A
1	B
1	B
2	B
3	C
3	C

atr1	atr2
1	A
1	B
2	B
3	C

```
SELECT DISTINCT TITUL  
FROM ZAM
```

WHERE

Klauzule WHERE definuje logické podmínky, které filtrují řádky, jež dotaz vrátí.

SELECT

*

FROM

tab1

WHERE

atr1 = 'A'; -- vrátí jen řádky, kde je atr1 roven A

```
SELECT DISTINCT CIS_ODD  
FROM ZAM  
WHERE TITUL = 'ING'
```

`atr1 = 'A'` -- rovnost
`atr1 <> 'A'` -- nerovnost
`atr1 > 100` -- větší než
`atr1 < atr2` -- menší než
`atr1 >= 100` -- větší nebo rovno než
`atr1 <= atr2` -- menší nebo rovno než
`atr1 BETWEEN '2014-08-22' AND '2014-08-30'` -- hodnoty ze
zadaného intervalu (včetně hraničních hodnot)
`atr1 NOT BETWEEN 100 AND 200`
`atr1 IN (1, 2, 3)` -- hodnota leží ve výčtu
`atr1 NOT IN (1, 2, 3)` -- hodnota neleží ve výčtu

Operátor LIKE slouží k porovnání řetězců:

- `atr1 LIKE 'A'` -- `atr1` musí být roven `A`

Znak `%` reprezentuje libovolný řetězec:

- `atr1 LIKE 'A%'` -- `atr1` musí začínat na `A`

Znak `_` reprezentuje jeden libovolný znak:

- `atr1 LIKE 'A_'` -- `atr1` musí začínat na `A` a má přesně dva znaky

Negace operátoru LIKE:

- `atr1 NOT LIKE '%Seznam.cz%'`
-- `atr1` nesmí obsahovat text `seznam.cz`

ZÁSTUPNÉ ZNAKY (WILDCARDS) - MSSQL

Znak % reprezentuje libovolný řetězec:

atr1 LIKE 'A%'

Znak _ reprezentuje jeden libovolný znak:

atr1 LIKE 'A_'

Znak [] reprezentuje jakýkoliv jednotlivý znak uvnitř závorek:

atr1 LIKE 'M[ai]rek' může být Marek i Mirek

Znak [^] reprezentuje jakýkoliv znak, který není v závorkách:

atr1 LIKE 'M[^aeou]rek'

Znak - reprezentuje jakýkoliv znak v rozsahu:

atr1 LIKE 'p[e-g]s'

NULL reprezentuje prázdnou/chybějící hodnotu.

- Aritmetické operace, ve kterých je jeden z operandů NULL, dávají NULL i na výsledek:

```
SELECT 6 + NULL
```

Logické výrazy s NULL (atr1 = NULL) vracejí hodnotu UNKNOWN.

Pro testování, zda je atribut NULL, se používá:

```
atr1 IS NULL
```

```
atr1 IS NOT NULL
```

Více podmínek lze kombinovat s operátory **AND** a **OR** a s použitím závorek.

Při připojování více podmínek pomocí **AND** musí být splněny všechny podmínky.

Při použití operátoru **OR** může být pro sadu výsledků pravdivá jedna nebo obě podmínky.

Operátory **AND** se zpracovávají před operátory **OR**, pokud nejsou použity závorky.

WHERE (atr1 = 2 **OR** atr1 = 3) **AND** atr2 < 100

Základní struktura SQL dotazu:

SELECT **co**
[seznam sloupců nebo * pro všechny sloupce]
FROM **z čeho**
[tabulka nebo seznam tabulek se spojením]
WHERE **kde (platí-li podmínka)**
[filtry nad řádky]
ORDER BY
[řazení výstupu]

Klauzule **WHERE** a **ORDER BY** jsou nepovinné.

ORDER BY definuje pořadí řádků ve výsledku dotazu.
Databáze sama od sebe žádné řazení negarantuje.

```
SELECT *  
FROM tab1  
ORDER BY atr1;
```

Je možné použít i poziční určení atributů:

```
SELECT atr1, atr2, hod1  
FROM tab1  
ORDER BY 1, 2;
```

Standardně se řadí vzestupně (ASC), sestupné řazení je nutné určit pomocí DESC:

```
SELECT atr1, atr2, hod1  
FROM tab1  
ORDER BY atr1 DESC, atr2 ASC;
```

V ORDER BY se lze odkazovat na alias sloupce ze SELECTu – vyhodnocuje se jako poslední.

Pokud je TOP použito bez ORDER BY, má nedeterministické výsledky.

Stejný problém nastane, pokud řadíme podle nejedinečného sloupce.

Pokud je nutné vrátit data, která se nacházejí ve více tabulkách, musíme databázovému stroji sdělit, jak se tyto tabulky mají napojit. Spojení se většinou děje přes PK = FK skrz klauzuli ON.

Rozlišujeme tři základní typy spojení:

- vnitřní spojení (INNER JOIN – lze psát pouze JOIN)
- vnější spojení (LEFT/RIGHT/FULL OUTER JOIN – slovo OUTER lze vynechat)
- kartézský součin (CROSS JOIN)

```
SELECT *  
FROM tab1  
JOIN tab2 ON tab1.atr1 = tab2.atr1;
```

SPOJOVÁNÍ TABULEK – SYNONYMA PRO TABULKY

Při spojování více tabulek je výhodné přiřadit tabulkám synonyma (aliasy) a následně jejich atributy kvantifikovat tímto aliasem.

Tím se zamezí nejednoznačností.

Pokud jsou definovány aliasy, nelze již použít názvy tabulek.

```
SELECT
```

```
  t1.*,
```

```
  t2.atr3
```

```
FROM
```

```
  tab1 t1
```

```
JOIN tab2 t2 ON t1.atr1 = t2.atr1;
```

Klauzule FROM definuje objekty (tabulky, pohledy), které obsahují podkladová data.

Jednotlivé tabulky mohou mít přiřazen alias:

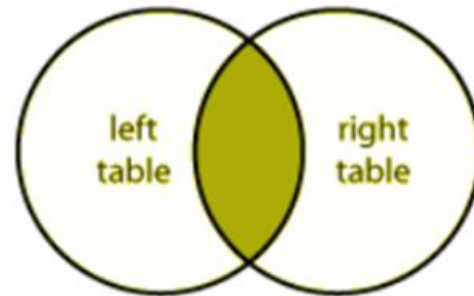
```
SELECT  
  a.*  
FROM  
  tab1 a;
```

Klauzule může obsahovat i jiný SQL dotaz – ten musí mít alias, pokud se na něj chceme odkazovat:

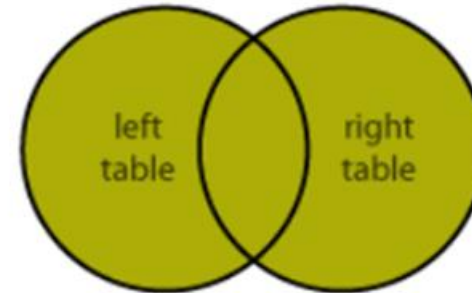
```
SELECT t.* FROM (SELECT atr1 FROM tab1) t;
```

SPOJOVÁNÍ TABULEK

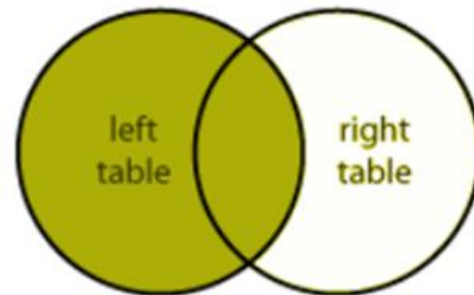
INNER JOIN



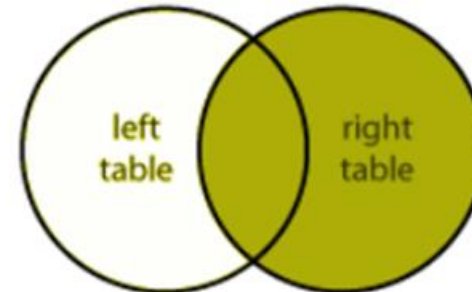
FULL JOIN



LEFT JOIN



RIGHT JOIN



SPOJOVÁNÍ TABULEK — INNER JOIN

INNER JOIN – na výstup se dostanou jen ty řádky z obou vstupních tabulek, které splňují danou spojovací podmínku.

Customer	
id	name
1	Jana
2	Pavel
3	Marta
4	Lenka

JOIN

Account	
id	account
1	a
2	b
2	c
3	d



id	name	account
1	Jana	a
2	Pavel	b
2	Pavel	c
3	Marta	d

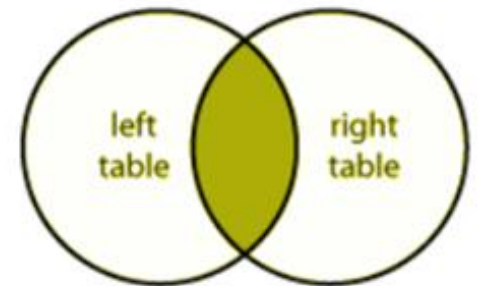
SELECT

Customer.*,

Account.account

FROM Customer

JOIN Account ON Customer.id = Account.id



SPOJOVÁNÍ TABULEK — LEFT OUTER JOIN

LEFT JOIN – na výstup se dostanou všechny řádky z první vstupní tabulky a ty záznamy ze zbývající tabulky, které splňují spojovací podmínku.

Customer	
id	name
1	Jana
2	Pavel
3	Marta
4	Lenka

LEFT JOIN

Account	
id	account
1	a
2	b
2	c
3	d



id	name	account
1	Jana	a
2	Pavel	b
2	Pavel	c
3	Marta	d
4	Lenka	Null

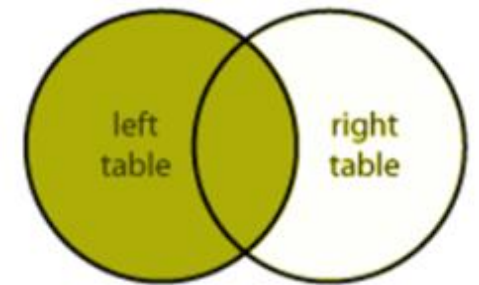
SELECT

Customer. ,

Account.account

FROM Customer

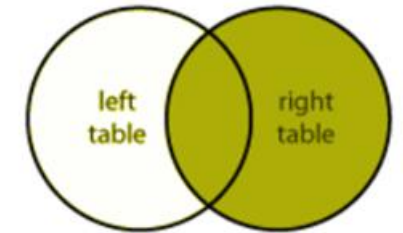
LEFT JOIN Account ON Customer.id = Account.id



SPOJOVÁNÍ TABULEK — RIGHT OUTER JOIN

RIGHT JOIN – na výstup se dostanou všechny řádky z druhé vstupní tabulky a ty záznamy z první tabulky, které splňují spojovací podmínku.

```
SELECT c.*, a.*  
FROM Customer c  
RIGHT JOIN Account a ON c.id = a.id
```



Customer	
id	name
1	Jana
2	Pavel
3	Marta

RIGHT JOIN

Account	
id	account
1	a
2	b
2	c
4	d



c.id	name	a.id	account
1	Jana	1	a
2	Pavel	2	b
2	Pavel	2	c
null	null	4	d

SPOJOVÁNÍ TABULEK — FULL OUTER JOIN

FULL JOIN – na výstup se dostanou všechny řádky z první i druhé tabulky; pokud splňují spojovací podmínku, jsou spojeny.

Customer	
id	name
1	Jana
2	Pavel
3	Marta

FULL JOIN

Account	
id	account
1	a
2	b
2	c
4	d



id	name	account
1	Jana	a
2	Pavel	b
2	Pavel	c
3	Marta	null
null	null	d

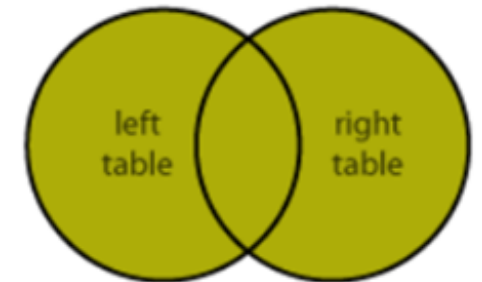
SELECT

Customer.*,

Account.account

FROM Customer c

FULL JOIN Account a ON c.id = a.id



SPOJOVÁNÍ TABULEK — CROSS JOIN

CROSS JOIN – každý řádek první tabulky je spojen s každým řádkem druhé tabulky.

```
SELECT t1.*, t2.atr3  
FROM tab1 t1  
JOIN tab2 t2 ON 1 = 1;
```

```
SELECT t1.*, t2.atr3  
FROM tab1 t1  
CROSS JOIN tab2 t2;
```

atr1	atr2
1	A
2	B

CROSS
JOIN

atr1	atr3
2	b
2	c
3	d



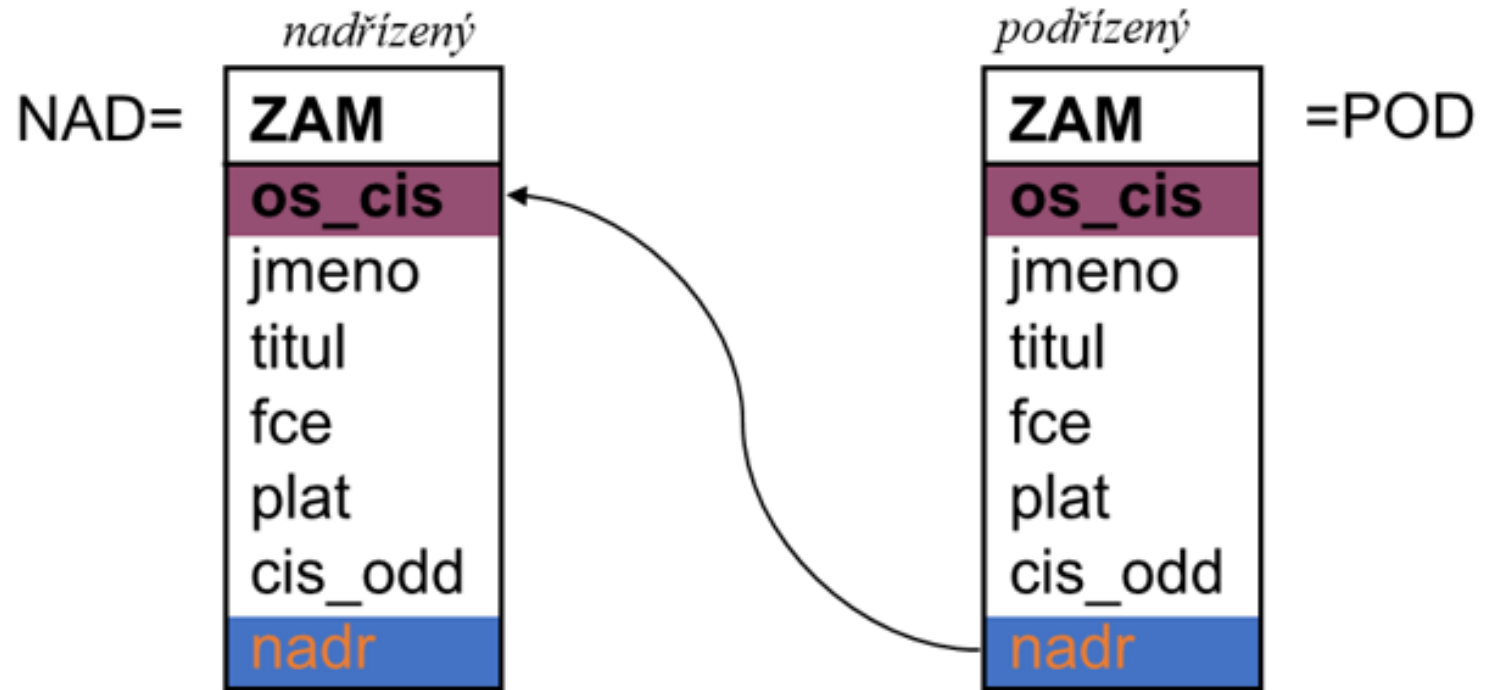
atr1	atr2	atr3
1	A	b
1	A	c
1	A	d
2	B	b
2	B	c
2	B	d

SELF JOIN

Self join – tabulka je spojena sama se sebou.

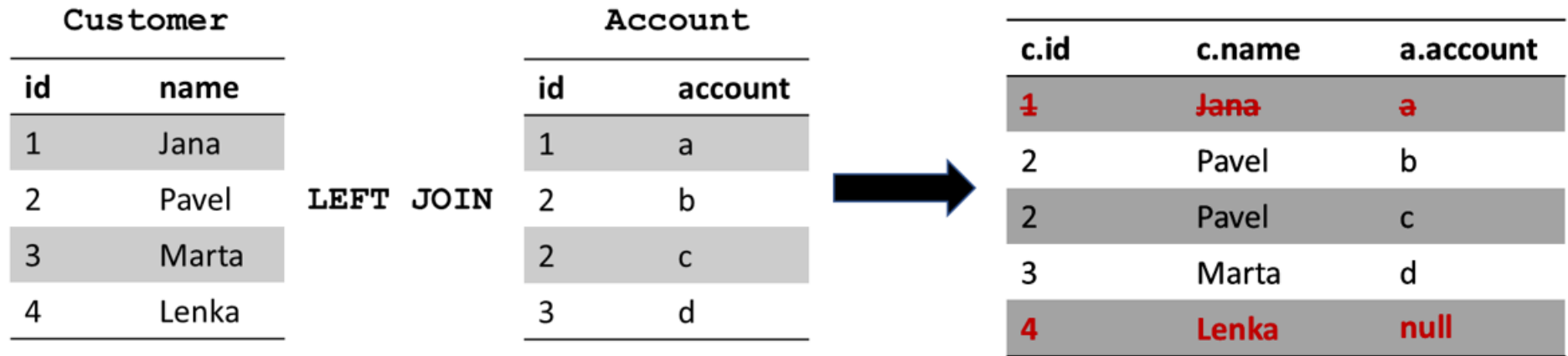
Příklad bezprostředně nadřízených a podřízených:

```
SELECT
  nad.os_cis,
  nad.jmeno,
  nad.titul,
  nad.fce,
  pod.os_cis,
  pod.jmeno
FROM zam nad
LEFT JOIN zam pod
  ON nad.os_cis = pod.nadr
ORDER BY 1
```



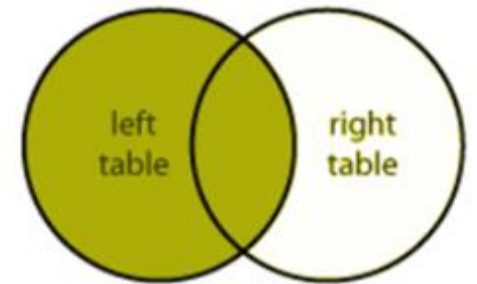
LEFT OUTER JOIN S PODMÍNKOU WHERE

V případě podmínky v klauzuli WHERE se řádky odebírají až po připojení tabulky.



```
SELECT c.*, a.account  
FROM customer c  
LEFT JOIN account a  
ON c.id = a.id
```

WHERE account NOT IN ('a')

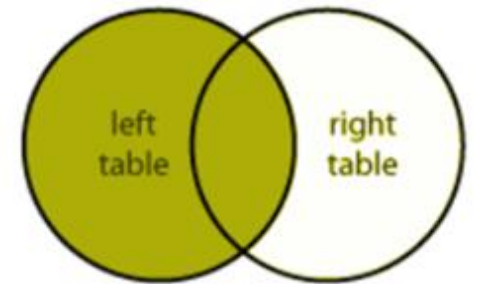


LEFT OUTER JOIN S PODMÍNKOU AND V ON

V případě podmínky v rámci klauzule ON je připojovaná tabulka omezena již na vstupu a připojuje se už vyfiltrovaná tabulka ke všem řádkům levé tabulky.

Customer		LEFT JOIN	Account			Result		
id	name		id	account		c.id	c.name	a.account
1	Jana		~~1~~	~~a~~	→	1	Jana	null
2	Pavel		2	b		2	Pavel	b
3	Marta		2	c		2	Pavel	c
4	Lenka		3	d		3	Marta	d
						4	Lenka	null

```
SELECT c.*, a.account  
FROM customer c  
LEFT JOIN account a  
  ON c.id = a.id  
  AND account NOT IN ('a');
```



Často je potřeba data z dotazu rozdělit do skupin a pro každou skupinu spočítat agregovanou hodnotu.

Agregační funkce vracejí jednu hodnotu pro celou skupinu řádků (suma, průměr, počet...).

Agregační funkce ignorují hodnotu NULL.

atr1	hod1		atr1	Průměr(hod1)	Suma(hod1)
A	50	→	A	125	500
A	100		B	500	1000
A	150		C	1000	1000
A	200				
B	400				
B	600				
C	1000				

SUM - součet hodnot ve skupině

AVG - spočítá průměr z daného výrazu pro každou skupinu

MAX - maximální hodnota

MIN - minimální hodnota ve skupině

počty:

- COUNT (výraz) - spočítá počet NOT NULL hodnot daného výrazu ve skupině
- COUNT (*) - spočítá počet řádků ve skupině
- COUNT (DISTINCT hod1) - spočítá počet unikátních hodnot atributu hod1

AGREGAČNÍ FUNKCE — SESKUPOVÁNÍ (GROUP BY)

Počet řádků v tabulce, průměr atributu hod1 a počet NOT NULL hodnot atributu hod1:

```
SELECT COUNT (*), AVG(hod1), COUNT (hod1)  
FROM tab1;
```

Suma přes skupinu:

```
SELECT atr1, SUM(hod1)  
FROM tabl  
GROUP BY atr1;
```

Dotaz vytvoří skupinu z hodnot atributů atr1 a atr2

```
SELECT atr1, atr2, SUM(hod1)  
FROM tabl  
GROUP BY atr1, atr2;
```

Skupinu v klauzuli GROUP BY nelze definovat podle pořadí atributů v dotazu (na rozdíl od ORDER BY)

AGREGACE DAT — SYNTAXE DOTAZU

SELECT	[seznam sloupců]
FROM	[tabulky]
WHERE	[filtry]
GROUP BY	[sloupce definující skupinu]
HAVING	[filtrace nad skupinami];

- GROUP BY definuje sloupce, jejichž jedinečné kombinace hodnot tvoří jednotlivé skupiny, pro které budou počítány agregované hodnoty
- Pokud je GROUP BY vynechána, pak jsou agregované hodnoty počítány z celé tabulky a SELECT musí obsahovat pouze agregační funkce
- Nepovinná klauzule HAVING slouží k vyfiltrování skupin (například na ty, jejichž průměr hodnot je > 1000)
- Při agregaci musí všechny sloupce v SELECT, HAVING a ORDER BY být sloupce, přes které se agreguje (jsou v GROUP BY) nebo být vstupem do agregační funkce
- Pokud dotaz používá GROUP BY, všechny následující fáze dotazu pracují se skupinami, nikoliv s originálními řádky

AGREGAČNÍ FUNKCE - FILTROVÁNÍ SKUPIN (HAVING)

K filtrování skupin slouží klauzule HAVING (jedná se o jakousi obdobu klauzule WHERE, ale pro agregace)

Do klauzule HAVING se zapisují logické výrazy, které pracují s agregačními funkcemi

Příklad - výpis skupin, kde průměr je větší než 500:

```
SELECT
  atr1,
  AVG (hod1)
FROM tabl
GROUP BY atr1
HAVING AVG(hod1) > 500;
```

ROZDÍL MEZI HAVING A WHERE

WHERE filtruje data před tím, než jsou seskupena

- Je možné filtrovat všechny sloupce zdrojových tabulek
- Není možné pracovat s výsledky agregačních funkcí

HAVING filtruje data až poté, co jsou spočítány skupiny

- Je možné pracovat napřímo pouze se sloupci specifikovanými v GROUP BY
- Je možné filtrovat skupiny na základě výsledků agregačních funkcí

POŘADÍ VYHODNOCOVÁNÍ SQL DOTAZU

1. SELECT — co
2. FROM — z čeho
3. JOIN — napojení další tabulky
4. WHERE — kde platí (podmínka)
5. GROUP BY — seskupit dle
6. HAVING — kde platí podmínka pro skupinu
7. ORDER BY — seřazeno dle

POŘADÍ VYHODNOCOVÁNÍ SQL DOTAZU

Pořadí, v jakém je dotaz zapsán, není pořadí, v jakém je vyhodnocován SQL Serverem

7 SELECT 8 DISTINCT 10 TOP <select list>

1 FROM <left table>

3 <join type> JOIN <right table>

2 ON <join condition>

4 WHERE <where condition>

5 GROUP BY <group by list>

6 HAVING <having condition>

9 ORDER BY <order by list>

Množinové operace pracují nad výsledky dvou SQL dotazů (dotazy musejí mít stejnou strukturu – počet atributů, jejich pořadí a kompatibilní datové typy – aplikuje se implicitní konverze)

Množinové operace:

- UNION (sjednocení)
- UNION ALL (sjednocení se zachovanými duplicitami)
- EXCEPT (rozdíl) - v Oracle MINUS
- INTERSECT (průnik)
- kartézky součin (cross join)

```
SELECT * FROM tabl, tab2
```

Číselné

- INTEGER (celé číslo)
- DECIMAL, NUMERIC (n,m) (desetinné číslo)
- FLOAT (n) (desetinné číslo)

Textové (potřeba uvozovat apostrofy)

- CHAR (n) (textový řetězec s pevnou délkou)
- VARCHAR (n) (textový řetězec s variabilní délkou)

Datum a čas

- DATE- funkce SYSDATETIME () či GETDATE () vrátí systémové datum
- TIMESTAMP

CAST (atribut AS datový typ) - ANSI standard

```
SELECT
```

```
  CAST (atribut AS INTEGER)
```

```
FROM tabulka
```

CONVERT (datový typ, atribut)

```
SELECT
```

```
  CONVERT (varchar (30) ,atribut)
```

```
FROM tabulka
```

OPERACE S TEXTOVÝMI ŘETĚZCI

UPPER / LOWER

UPPER (text) LOWER (text)

- Převeďte text na velká/malá písmena

SUBSTRING

SUBSTRING (text, start, délka)

- Vybere část textu od určité pozice a určité délky

TRIM/LTRIM/RTRIM

TRIM(text) TRIM(znak FROM text)

- Odstraňuje mezery (z obou stran/zleva/zprava)

LEFT/RIGHT

LEFT(text, počet znaků)

- Odebere počet znaků zleva/zprava

LEN

LEN(text)

- Vrátil délku textového řetězce

REPLACE

REPLACE (text, původní text, nový text)

- Nahradí část textu novým textem

STUFF

STUFF (text, start, délka, nový text)

- Nahradí část textu jiným textem od určité pozice (start) a určité délky

TRANSLATE

TRANSLATE (text, znaky, nové znaky)

- Nahradí znaky v textu novými znaky

CONCAT/CONCAT

WS CONCAT WS (oddělovač, text, text, text)

- Spojí textové řetězce/přidá oddělovač

DIFFERENCE

DIFFERENCE (text1, text2)

- Porovnává podobnost textu (0— 4) 59

ABS **ABS(hodnota)**

- Absolutní hodnota

CEILING **CEILING(hodnota)**

- Zaokrouhlení na celé číslo směrem nahoru

FLOOR **FLOOR(hodnota)**

- Zaokrouhlení na celé číslo směrem dolů

ROUND **ROUND(hodnota, počet míst)**

- Zaokrouhlení na určitý počet desetinných míst (zbylá nahradí 0)

RAND **RAND()** **RAND(5)**

- Vygeneruje náhodné číslo mezi 0 a 1, hodnota v závorce udává kolikrát se má zopakovat stejné číslo

MOD **MOD(dělenec, dělitel)**

- Vráť zbytek dělení

DATEADD(interval, počet, datum)

- přidá počet roků/měsíců/dní k datu.

DATEDIFF(interval, datum1, datum2)

- rozdíl mezi daty.

DATEFROMPARTS(rok, měsíc, den)

- spojí části data v celé datum.

DATEPART(interval, datum)

- vrátí specifickou část data.

DATENAME(interval, datum)

- vrátí specifickou část data.

DAY(datum) / MONTH(datum) / YEAR(datum)

- vrátí část data.

GETDATE()

- vrátí systémové datum a čas.

CURRENT_TIMESTAMP

- vrátí současné datum a čas.

COALESCE(hod1, hod2, hod3, ...)

- vrátí první nenullovou hodnotu v seznamu.

ISNULL(výraz, hodnota)

- nahradí NULL jinou hodnotou.

IIF(podmínka, pravda, nepravda)

- testování podmínky.

STRING_AGG(hod1, ',')

- spojení textových hodnot ve skupině.

CASE – testování podmínky

PODMÍNKA CASE

CASE slouží k porovnávání podmínek - podmínky se vyhodnocují postupně.

CASE

WHEN age < 18 THEN infant

WHEN age > 65 THEN senior

ELSE adult

END AS age_category

SELECT CustomerName, City, Country

FROM Customers

ORDER BY

CASE

WHEN City IS NULL THEN Country

ELSE City

END;

DERIVOVANÁ TABULKA

Derivovaná tabulka je virtuální tabulka definovaná v rámci dotazu, která je vyhodnocena jako první a následně slouží jako zdroj dat pro vnější dotaz.

Není uložena v databázi, nelze se na ni v dotazu odkázat opakovaně.

Je možné ji dále zanořovat.

Lze v nich použít proměnné a parametry platné pro vnější dotaz.

Musí mít definovaný alias tabulky.

Všechny sloupce musí mít jméno (definované aliasy sloupců, nebo pomocí aliasu tabulky s doplněnými názvy sloupců).

Musí vracet množinu – tedy nesmí používat ORDER BY bez TOP nebo FETCH/OFFSET.

```
SELECT OrderYear,  
       COUNT(DISTINCT CustomerId) AS CustomerCount  
FROM ( SELECT CustomerId,  
            YEAR(OrderDate) AS OrderYear  
      FROM Orders) AS OrderWithYear  
GROUP BY OrderYear;
```

SUBDOTAZY (SUBQUERY)

SELECT zanořený uvnitř jiného dotazu typu SELECT, INSERT, UPDATE, DELETE, aj.

Jednoduché – Samostatné subdotazy (Self-contained Subqueries)

- Nejprve je vyhodnocen vnitřní dotaz jako celek a jeho hodnoty jsou pak dosazeny do vnějšího dotazu

Souvztažné – Korelované subdotazy (Correlated Subqueries)

- Vnitřní SELECT je vyhodnocován na základě hodnot z vnějšího příkazu SELECT

Subdotaz může vrátit:

- Skalární hodnotu (jedna hodnota) — (SELECT TOP 1 CustomerID FROM Customers)
- Množinu hodnot (jeden sloupec, více řádků) — (SELECT CustomerID FROM Customers)
- Množinu řádků (více sloupců, více řádků) — (SELECT CustomerID, CustomerName FROM Customers)

Výhody:

- Alternativní způsob dotazování na data, který by vyžadoval složitá spojení a sjednocení.
- Složitější dotazy jsou čitelnější
- Lze izolovat každou část složitého dotazu

Lze je spustit samostatně

Subdotazy ve FROM klauzuli:

- Vrací množinu řádků
- Je to vlastně derivovaná tabulka

`SELECT t.* FROM (SELECT atr1 FROM tab1) t`

Subdotazy v podmínce WHERE nebo HAVING:

- Vracející množinu hodnot (IN, ALL, ANY, SOME)
 - Pokud se hodnota pro aktuálně zpracovávaný řádek nachází v množině hodnot vrácených subdotazem, predikát IN vrací hodnotu PRAVDA
- Vracející maximálně jednu hodnotu (<, >, =, LIKE)
 - Pokud vnitřní dotaz nevrátí žádný řádek, návratová hodnota je zkonvertována na NULL
- Fungující jako test existence (pomocí EXISTS)
- `SELECT atr1, atr2 FROM tab1 WHERE atr2 > (SELECT AVG(atr3) FROM tab2)`

Subdotazy v SELECT klauzuli

`SELECT tab1.atr1, (SELECT AVG(atr2) FROM tab2), tab1.atr3 FROM tab1`

ZÁVISLÉ SUBDOTAZY (SOUVZTAŽNÉ)

Vnitřní příkaz SELECT je vyhodnocován na základě hodnot z vnějšího příkazu SELECT

Vnější dotaz dodává hodnotu z každého zpracovávaného řádku relace do vnitřního dotazu, který je pro každou z těchto hodnot opakovaně vyhodnocován.

V podmínce WHERE: (produkty, jejichž cena je vyšší než průměrná cena za kategorii, do které produkt patří)

```
SELECT      product_id,  
            product_name,  
            list_price  
FROM products p  
WHERE list_price > (SELECT AVG(list_price)  
                  FROM products  
                  WHERE category_id = p.category_id);
```

V klauzuli SELECT: (ke každému produktu vypíše průměrnou cenu za kategorii, do které produkt patří)

```
SELECT      product_id,  
            product_name,  
            standard_cost,  
            (SELECT AVG(standard_cost)  
              FROM products  
              WHERE category_id = p.category_id) avg_standard_cost  
FROM products p;
```

DĚKUJI VÁM ZA POZORNOST

Lektor: Ing. Michal Konvalinka

Web: exceltown.com

Kontakt: Konvalinka.Michal@gmail.com
774 322 245

ExcelTown.com 