Databázové systémy

Množinové operácie a vnorené dopyty v SQL

2026 Lukáš Miňo

Motivácia

- **Limitácia cez JOIN:** Spojenie cez JOIN (**horizontálne spojenie**) vyžaduje, aby tabuľky mali explicitne definovaný vzťah (cudzí kľúč) a spájali sa „riadok k riadku“.
- Čo ak chceme spojiť výsledky dvoch dopytov nad rôznymi tabuľkami pod seba (**vertikálne**)?
- Čo ak potrebujeme filtrovať dáta na základe **dynamického výsledku**, ktorý v čase písania dopytu nepoznáme (napr. Všetky osoby, ktoré majú nadpriemernú výšku)?
- **Riešenie:** Využitie množinových operácií a vnorených dopytov (subqueries).

Množinové operácie (Set Operations)

Množinové operácie v SQL

- **Princíp:** Operácie spájajú (**vertikálne**) výsledky dvoch alebo viacerých samostatných SELECT dopytov do jednej výslednej tabuľky.
- Dopyty musia byť **typovo kompatibilné**:
  - Musia vracať rovnaký počet stĺpcov.
  - Stĺpce na rovnakých pozíciách musia mať kompatibilné dátové typy.
- **Základné operátory:** UNION, INTERSECT, EXCEPT / MINUS (Oracle).
- **Dva režimy spracovania:**
  - **Bez ALL (predvolené):** Pracuje s dátami ako s **množinami**. Automaticky **odstraňuje duplicitné riadky** z výsledku, čo vyžaduje dodatočné triedenie a zaťažuje RAM/CPU.
  - **S prívlastkom ALL: Zachováva** všetky **duplicitné riadky**. Je výrazne **rýchlejší**, nakoľko DBMS dáta iba mechanicky spojí alebo porovná bez triedenia.

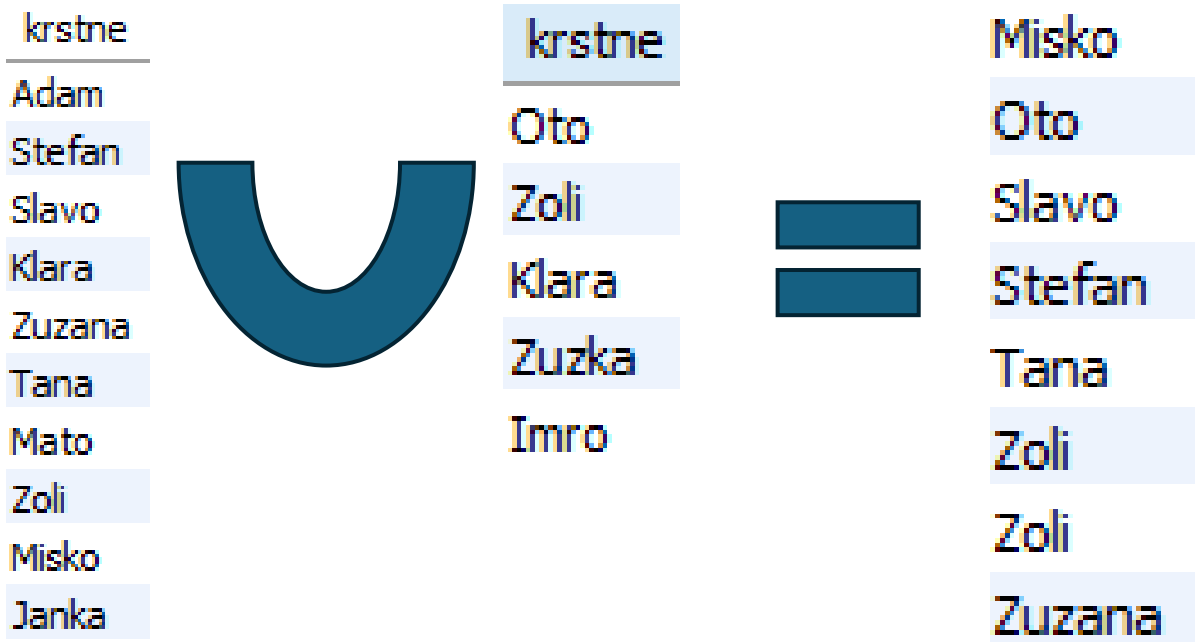
Zjednotenie: UNION vs. UNION ALL

- **UNION:** Spojí riadky oboch dopytov pod seba a **odstráni duplicitné riadky** (vykoná DISTINCT). Ak napr. meno figuruje v oboch výberoch, vo výsledku bude len raz.
- **UNION ALL:** Spojí riadky oboch dopytov pod seba a **vrátane duplicít**. Je výrazne **rýchlejší**, nakoľko neuskutočňuje žiadne čistenie od duplicít.

Zjednotenie: UNION vs. UNION ALL

- Zobrazte všetky mená pacientov a lekárov. Mená sa môžu opakovať. Výsledok zoraďte lexikograficky.

```
-- 15 riadkov
SELECT krstne FROM pacienti
UNION ALL
SELECT krstne FROM lekari
ORDER BY krstne;
```



Zjednotenie: UNION vs. UNION ALL

- Zobrazte všetky mená pacientov a lekárov tak, aby sa mená neopakovali. Výsledok zoraďte lexikograficky.

```
-- 13 riadkov
SELECT krstne FROM pacienti
UNION
SELECT krstne FROM lekari
ORDER BY krstne;
```

krstne
Adam
Imro
Janka
Klara
Mato
Misko
Oto
Slavo
Stefan
Tana
Zoli
Zuzana
Zuzka

Prienik: INTERSECT vs. INTERSECT ALL

- **INTERSECT:** Vracia iba riadky, ktoré sa nachádzajú v prvom aj druhom dopyte. Ak sa hodnota nachádza v prvej tabuľke 3-krát a v druhej 2-krát, vo výsledku bude **iba 1-krát**.
- **INTERSECT ALL:** Zachováva násobnosti duplicít z oboch strán. Ak je hodnota v tabuľke A *X-krát* a v tabuľke B *Y-krát*, vo výsledku sa objaví presne **MIN(X, Y)-krát**.

Prienik: INTERSECT vs. INTERSECT ALL

- Zobrazte rovnaké mená pacientov a lekárov. Výsledok zoraďte lexikograficky.

```
-- 2 riadky
SELECT krstne FROM pacienti
INTERSECT
SELECT krstne FROM lekari
ORDER BY krstne;
```

krstne
Klara
Zoli

Rozdiel: EXCEPT vs. EXCEPT ALL

- **EXCEPT (v Oracle MINUS):** Vráti riadky z prvého dopytu, ktoré kompletne chýbajú v druhom dopyte. Odstráni všetky duplicity.
- **EXCEPT ALL:** Zohľadňuje počty riadkov. Ak sa riadok nachádza v tabuľke A *X-krát* a v tabuľke B *Y-krát*, vo výsledku zostane presne **X - Y-krát** (ak X > Y, inak sa nezobrazí).

Prienik: INTERSECT vs. INTERSECT ALL

- Zobrazte len mená lekárov, ktorých mená neobsahujú mená pacientov. Výsledok zoraďte lexikograficky.

```
-- 3 riadky
SELECT krstne FROM lekari
EXCEPT
SELECT krstne FROM pacienti
ORDER BY krstne;
```

krstne
Imro
Oto
Zuzka

Vnorené dopyty (Subqueries)

Vnorené dopyty

- **Definícia:** Vnorený dopyt (Subquery / Inner Query) je SELECT príkaz umiestnený vnútri iného SQL príkazu (Outer Query).
- **Kategorizácia podľa návratovej hodnoty:**
  - **Skalárne:** vracajú presne 1 hodnotu, t.j. 1 riadok a 1 stĺpec.
  - **Stĺpcové:** vracajú zoznam hodnôt z jedného stĺpca.
  - **Tabuľkové:** vracajú tabuľku s viacerými stĺpcami a riadkami.

Kde všade môžeme písať vnorené dopyty?

1. **Za klauzulou SELECT (Projekcia):** Vnorený dopyt vystupuje ako vypočítaný stĺpec. Musí byť striktne **skalárny** (vrátiť 1 hodnotu pre daný riadok).
2. **Za klauzulou FROM (Derivovaná tabuľka):** Vnorený dopyt vytvára dočasnú virtuálnu tabuľku v pamäti, nad ktorou beží hlavný dopyt. **Vyžaduje alias**.
3. **Za klauzulou WHERE (Selekcia):** Najčastejšie použitie. Slúži na dynamické filtrovanie riadkov pomocou **porovnávacích operátorov (=, >, <, >=, <=, <>, !=)**, **IN, EXISTS, ANY/SOME** alebo **ALL**.
4. **Za klauzulou HAVING (Filtrácia skupín):** Slúži na filtrovanie už zoskupených dát (GROUP BY) oproti agregovanej hodnote.
5. **V DML príkazoch (INSERT, UPDATE, DELETE):** Dynamická modifikácia dát na základe podmienok z iných tabuliek.

1. Za klauzulou **SELECT** (Skalárny výpočet stĺpca)

- **Pravidlo:** Vnútorný dopyt sa vyhodnocuje pre každý riadok zvlášť a **nesmie** vrátiť viac ako 1 riadok a 1 stĺpec.

- Pridajte stĺpec poradie k pacientom, kde tento stĺpec bude reprezentovať lexikografické poradie krstných mien v abecede.

```
SELECT
  (SELECT COUNT(*) + 1
   FROM pacienti x
   WHERE x.krstne < p.krstne
  ) poradie,
  p.*
FROM pacienti p;
```

poradie	idP	krstne	mesPrijem
1	1	Adam	10000
7	2	Stefan	9500
6	3	Slavo	8500
3	4	Klara	9000
10	5	Zuzana	35000
8	6	Tana	20000
4	7	Mato	28000
9	8	Zoli	32000
5	9	Misko	NULL
2	10	Janka	NULL

2. Za klauzulou **FROM** (Derivovaná tabuľka)

- **Pravidlo:** Vytvára virtuálnu tabuľku pre ďalšiu agregáciu. Každý takýto vnútorný dopyt **musí mať priradený alias** (napr. **AS moja_tabulka**).

- Zobrazte rôzne krstné pacientov, ktorí navštívili daného lekára viac ako raz.

```
SELECT DISTINCT p.krstne
FROM
(
  SELECT idP, idL, COUNT(*) pocet
  FROM navstevy
  GROUP BY idP, idL
) AS T -- musi byt alias tabulky
JOIN pacienti p ON p.idP = T.idP
WHERE T.pocet > 1;
```

krstne
Stefan
Mato

3. Za klauzulou **WHERE** (Filtrovanie riadkov)

- **Pravidlo:** Porovnáva atribút konkrétneho riadku s výsledkom vnútorného dopytu ešte **pred zoskupením**.

- Nájdite údaje o pacientoch, ktorí majú najmenší mesačný príjem.

```
SELECT p.*
FROM pacienti p
WHERE
p.mesPrijem =
( SELECT
      MIN(p1.mesPrijem)
  FROM pacienti p1
  WHERE p1.mesPrijem IS NOT NULL
);
```

idP	krstne	mesPrijem
3	Slavo	8500

4. Za klauzulou **HAVING** (Filtrovanie zoskupených dát)

- **Pravidlo:** Vyhodnocuje sa **až po klauzule** GROUP BY. Porovnáva agregovanú hodnotu skupiny s iným vnútorným dopytom.
- Nájdite priezviská, ktorých priemerná výška je vyššia ako celkový priemer.

```
SELECT priezvisko, AVG(vyska) AS priemerna_vyska
FROM osoba
GROUP BY priezvisko
HAVING AVG(vyska) > (SELECT AVG(vyska) FROM osoba)
ORDER BY priemerna_vyska DESC;
```

priezvisko	priemerna_vyska
Ľavá	195.50000
Novák	182.50000
Prvý	180.00000
Horváth	176.25000
Múdry	175.50000
Silný	175.50000
Kováč	173.25000
Urbanová	172.50000
Pravá	170.50000

• 5. V **DML** príkazoch (**INSERT, UPDATE, DELETE**)

- Vnorené dopyty nie sú obmedzené len na čítanie dát. V praxi sú kľúčové pri hromadných úpravách a mazaní záznamov.
- **Špecifikum MySQL:** MySQL nedovoľuje modifikovať (UPDATE/DELETE) tú istú tabuľku, z ktorej priamo čítate vo vnútornom dopyte WHERE. Riešením je dvojité vnorenie cez FROM.

- Vymažte vzťahy osôb, ktoré už nežijú (majú vyplnený dat_smrti).

```
DELETE FROM vztah
WHERE id_on IN (SELECT id FROM osoba WHERE dat_smrti IS NOT NULL);
```

• 5. V **DML** príkazoch (**INSERT, UPDATE, DELETE**)

- Nastavte dátum ukončenia vzťahu na skorší z dátumov úmrtia partnerov.

```
UPDATE vztah v
SET do = (
   SELECT MIN(o.dat_smrti)
   FROM osoba o
   WHERE o.id IN (v.id_on, v.id_ona)
)
WHERE v.do IS NULL
  AND EXISTS (
     SELECT 1
     FROM osoba o
     WHERE o.id IN (v.id_on, v.id_ona) AND o.dat_smrti IS NOT NULL
  );
```

- 5. V **DML** príkazoch (**INSERT, UPDATE, DELETE**)

- Vložte zosnulé osoby z tabuľky Osoba do archívu iba ak tam ešte neexistujú.

```
INSERT INTO archiv_zosnulych (id, meno, priezvisko, dat_nar, dat_smrti)
SELECT o.id, o.meno, o.priezvisko, o.dat_nar, o.dat_smrti
FROM osoba o
WHERE o.dat_smrti IS NOT NULL
  AND NOT EXISTS (
     SELECT 1
     FROM archiv_zosnulych a
     WHERE a.id = o.id
  );
```

Kategorizácia podľa návratovej hodnoty

- **Kategorizácia podľa návratovej hodnoty:**
  - **Skalárne:** vracajú presne 1 hodnotu – 1 riadok, 1 stĺpec.
  - **Stĺpcové:** vracajú zoznam hodnôt z jedného stĺpca.
  - **Tabuľkové:** vracajú tabuľku s viacerými stĺpcami a riadkami.

Skalárne vnorené dopyty

- **Pravidlo použitia:** Ak vnútorný dopyt vracia iba jednu jedinú hodnotu, môžeme použiť klasické porovnávacie operátory (=, >, <, >=, <=, <>, !=).
- **Riziko zlyhania:** Ak vnorený dopyt vráti **viac ako jeden riadok**, vonkajší dopyt **zlyhá s chybou**.

```
-- Výber všetkých osôb, ktoré sú vyššie ako je priemerná výška v databáze
SELECT meno, priezvisko, vyska
FROM osoba
WHERE vyska > (SELECT AVG(vyska) FROM osoba); -- Vráti 1 číselnú hodnotu

-- Zobrazenie osoby a váhového rozdielu oproti maximu
SELECT meno, priezvisko, vaha,
    (SELECT MAX(vaha) FROM osoba) - vaha AS odchylka_od_najtazsieho
FROM osoba
WHERE vaha IS NOT NULL;
```

Viachodnotové vnorené dopyty (IN, ANY/SOME, ALL)

- Ak vnútorný dopyt vracia stĺpec hodnôt (viac riadkov), musíme nahradiť porovnávacie znamienka operátormi:
  - **IN:** Hodnota sa **musí nachádzať** v zozname.
  - **ANY / SOME:** Podmienka je splnená, ak platí **aspoň pre jeden prvok** zoznamu.
  - **ALL:** Podmienka musí platiť **pre všetky prvky** zoznamu súčasne.

```
-- Nájdite deti, ktorých matka je Eva alebo Mária
SELECT meno, priezvisko
FROM osoba
WHERE matka IN (SELECT id FROM osoba WHERE meno IN ('Eva', 'Mária'));

-- Nájdite osoby, ktoré sú vyššie ako VŠETKY ženy v databáze
SELECT meno, priezvisko, vyska
FROM osoba
WHERE vyska > ALL (SELECT vyska FROM osoba WHERE pohlavie = 'z' AND vyska IS NOT NULL);
```

Korelované vnorené dopyty (Correlated Subqueries)

- **Nekorelovaný dopyt:** Beží samostatne. Databáza ho vykoná iba raz a výsledok odovzdá ďalej.
- **Korelovaný dopyt:** Vnútri vnútorného dopytu sa nachádza **referencia na stĺpec z vonkajšieho dopytu**.
- **Algoritmická náročnosť:** Databáza spúšťa vnútorný dopyt **pre každý jeden riadok** vonkajšieho dopytu (analógia s cyklom v cykle $O(N^2)$), čo môže spôsobiť závažné výkonnostné problémy na veľkých tabuľkách.

Korelované vnorené dopyty (Correlated Subqueries)

```
-- Nájdite osoby, ktoré sú vyššie ako priemer ich vlastného pohlavia
SELECT o1.meno, o1.priezvisko, o1.pohlavie, o1.vyska
FROM osoba o1
WHERE o1.vyska > (
   SELECT AVG(o2.vyska)
   FROM osoba o2
   WHERE o2.pohlavie = o1.pohlavie -- Prepojenie (korelácia) s o1
);
```

Optimalizácia cez EXISTS a NOT EXISTS

- **Princíp:** Nevracia žiadne konkrétne dáta, iba logické **TRUE/FALSE**.
- **Výkonnostná výhoda:** Databázový **Query Optimizer** zastaví vyhľadávanie vo vnútornom dopyte v momente, keď narazí na **prvú zhodu** (Short-circuit evaluation). Nemusí prechádzať celú tabuľku do konca.

```
-- Vyberte len tie osoby, ktoré reálne majú v tabuľke nejaké zapísané deti
SELECT o.id, o.meno, o.priezvisko
FROM osoba o
WHERE EXISTS (
   -- používa sa SELECT 1,
   -- nakoľko na vybraných stĺpcoch nezáleží, hľadá sa iba riadok
   SELECT 1 FROM osoba AS rodic WHERE o.id IN (rodic.otec, rodic.matka)
);
```

Viacnásobné vnorenie vnútorných dopytov (Multi-level Subqueries)

- **Princíp vykonávania (Bottom-up):** Databázový engine vyhodnocuje dopyty **od najvnútornejšieho smerom von**. Výsledok najhlbšej úrovne slúži ako vstupná množina pre nadradenú úroveň.
- **Viditeľnosť stĺpcov (Scope):** Vonkajší dopyt **nevidí** stĺpce z tabuliek, ktoré sú použité iba vo vnútri vnútorného dopytu.

- Vypište mená a priezviská vnúčat po otcovskej vetve Adama Prvého.

```
-- 3. úroveň (Najvonkajší dopyt): Vyberieme mená vnúčat
SELECT meno, priezvisko FROM osoba
WHERE otec IN (
   -- 2. úroveň: Nájdeme synov osôb z 1. úrovne
   SELECT id FROM osoba
   WHERE otec IN (
      -- 1. úroveň (Najvnútornejší dopyt): Nájdeme ID zakladateľa rodu
      SELECT id FROM osoba WHERE meno = 'Adam' AND priezvisko = 'Prvý'
   )
);
```

meno	priezvisko
DÁŠa	Nováková
Viera	Silná

Viacnásobné vnorenie vnútorných dopytov (Multi-level Subqueries)

- Ak vnorenie presiahne viac úrovni:
  - **Zlyhanie Query Optimizera:** Optimalizátor prestáva efektívne prepisovať vnútorné dopyty na JOIN operácie, čo vedie k pomalému exekučnému plánu.
  - **Nečitateľnosť kódu:** Ak vnorenie presiahne viac úrovni, kód sa stáva nečitateľným.

- **Riešenie:** V letnom semestri

Nájdite údaje o pacientoch, ktorí majú druhý najmenší mesačný príjem.

```
-- Pomocou MIN
SELECT
  p.*
FROM
  pacienti p
WHERE
p.mesPrijem =
(
  SELECT
    MIN(p2.mesPrijem)
  FROM
    pacienti p2
  WHERE
    p2.mesPrijem >
    (
      SELECT
        MIN(p1.mesPrijem)
      FROM
        pacienti p1
      WHERE
        p1.mesPrijem IS NOT NULL
    )
);
```

idP	krstne	mesPrijem
4	Klara	9000

MIN(p2.mesPrijem)
9000

MIN(p1.mesPrijem)
8500

```
-- Pomocou ALL
SELECT
 T.*
FROM
(
 SELECT p3.* FROM pacienti p3
 WHERE p3.mesPrijem >
 (
  SELECT p2.mesPrijem FROM pacienti p2
  WHERE p2.mesPrijem <= ALL
  (
   SELECT p1.mesPrijem FROM pacienti p1
   WHERE p1.mesPrijem IS NOT NULL
  )
 )
) T
WHERE
T.mesPrijem <= ALL
(
 SELECT p3.mesPrijem FROM pacienti p3 -- 7
 WHERE p3.mesPrijem >
 (
  SELECT p2.mesPrijem FROM pacienti p2 -- 1
  WHERE p2.mesPrijem <= ALL
  (
   SELECT p1.mesPrijem FROM pacienti p1 -- 8
   WHERE p1.mesPrijem IS NOT NULL
  )
 )
);
```

idP	krstne	mesPrijem
4	Klara	9000

idP	krstne	mesPrijem
1	Adam	10000
2	Stefan	9500
4	Klara	9000
5	Zuzana	35000
6	Tana	20000
7	Mato	28000
8	Zoli	32000

mesPrijem
10000
9500
8500
9000
35000
20000
28000
32000

mesPrijem
8500

mesPrijem
10000
9500
9000
35000
20000
28000
32000

Viacnásobné vnorenie vnútorných dopytov (Multi-level Subqueries)

- Nájdite údaje o pacientoch, ktorí májú tretí najmenší mesačný príjem.

-- v ramci domacej pripravy

Prepis množinových operácií

Prepis množinových operácií

- Prepísanie množinovej operácie (**UNION, INTERSECT, EXCEPT**) na logický operátor **OR** alebo vnorený dopyt (**IN, EXISTS**) sa oplatí najmä z dôvodu **vyššieho výkonu (menej prechodov tabuľkou)**, **úspory pamäte pri odstraňovaní duplicít** a **kompatibility so staršími databázovými systémami**.

1. Eliminácia viacnásobného čítania tabuľky (I/O optimalizácia)
2. Úspora pamäte a procesora (Odstránenie triedenia)
3. Kompatibilita a prenositeľnosť kódu medzi DBMS
4. Vyššia flexibilita pri výbere stĺpcov a práci s **NULL**

Kedy sa prepis neoplatí?

- **Rôzne nesúvisiace tabuľky:** Ak potrebujete spojiť krstné mená z tabuľky Lekari a Pacienti pod seba do jedného stĺpca, operátor **OR** ani **EXISTS** použiť nemôžete a **UNION / UNION ALL je jedinou správnou voľbou**.
- **Využitie viacerých indexov naraz:** Ak v tabuľke hľadáte záznamy podľa dvoch rôznych stĺpcov (**WHERE stlpec_A = 1 OR stlpec_B = 2**) a každý stĺpec má vlastný index, niektoré databázové optimalizátory nedokážu použiť oba indexy pri OR naraz. Vtedy je paradoxne rýchlejšie prepísať OR na dva samostatné dopyty spojené cez **UNION ALL**.

Zjednotenie: UNION

- **Princíp zjednotenia:** Spojí riadky oboch dopytov pod seba.
- Kedy sa dá operácia **UNION** nahradiť operátorom **OR**?
  - Ak oba vnútorné dopyty vyberajú **dáta z tej istej tabuľky** (alebo zhodného spojenia tabuliek). Vtedy ide o spojenie dvoch filtrovacích podmienok v klauzule **WHERE** namiesto vertikálneho spájania dvoch samostatných dopytov.
- **Lepší výkon:** Databáza prečíta tabuľku iba **raz** (jeden priechod), zatiaľ čo **UNION** núti systém vykonať dva samostatné dopyty a výsledky spájať.
- **Pozor na duplicity:** Operátor **OR** prirodzene vráti každý riadok tabuľky maximálne raz. Ak chceme nahradiť **UNION** na stĺpcoch s opakujúcimi sa hodnotami (napr. priezviská), musíme pridať **DISTINCT**.

Zjednotenie: UNION

```sql
-- 1. Množinový zápis (2 prechody tabuľkou + odstraňovanie duplicít)
SELECT id, meno, priezvisko FROM osoba WHERE vyska > 190.0
UNION
SELECT id, meno, priezvisko FROM osoba WHERE YEAR(dat_nar) < 1925;

-- 2. Optimalizovaný prepis cez OR (1 prechod tabuľkou, rovnaký výsledok)
SELECT id, meno, priezvisko
FROM osoba
WHERE vyska > 190.0 OR YEAR(dat_nar) < 1925;
```

Zjednotenie: UNION

- Kedy sa **nedá** operácia **UNION** nahradiť operátorom **OR**?
  - **Rôzne tabuľky:** Ak spájate dáta z dvoch odlišných entít
    - napr. zoznam dátumov narodenia z tabuľky *Osoba* a dátumov začiatku vzťahov z tabuľky *Vztah*).
  - **Rôzne stĺpce z tej istej tabuľky pod seba:** Ak chcete spojiť hodnoty stĺpca *otec* a stĺpca *matka* do **jedného jediného výsledného stĺpca** pod seba. Operátor **OR** by ich nechal vedľa seba.

Prienik: INTERSECT

- **Princíp prieniku:** Hľadáme prvky z prvej množiny, ktoré zároveň existujú v druhej množine.
- **Pozor na duplicity:** Na rozdiel od čistého INTERSECT musíme pri prepise použiť DISTINCT, aby sme predišli opakovaniu riadkov.

```sql
-- Prienik pomocou IN
SELECT krstne
FROM pacienti
WHERE krstne IN (SELECT krstne FROM lekari)
ORDER BY krstne;

-- Prienik pomocou ANY
SELECT Krstne
FROM pacienti
WHERE krstne = ANY (SELECT krstne FROM lekari)
ORDER BY krstne;

-- Prienik pomocou SOME
SELECT krstne
FROM pacienti
WHERE krstne = SOME (SELECT krstne FROM
lekari)
ORDER BY krstne;

-- Prienik pomocou EXISTS
SELECT krstne
FROM pacienti
WHERE EXISTS (SELECT krstne FROM lekari
              WHERE lekari.krstne = pacienti.krstne)
ORDER BY krstne;
```

```sql
-- Prienik pomocou INTERSECT (od verzie 8.0.31)
SELECT krstne FROM pacienti
INTERSECT
SELECT krstne FROM lekari
ORDER BY krstne;

-- Prienik pomocou CROSS JOIN
SELECT p.krstne
FROM pacienti p, lekari l
WHERE p.krstne = l.krstne
ORDER BY p.krstne;

-- Prienik pomocou CROSS JOIN
SELECT p.krstne
FROM pacienti p
CROSS JOIN lekari l
WHERE p.krstne = l.krstne
ORDER BY p.krstne;

-- Prienik pomocou JOIN
SELECT p.krstne
FROM pacienti p
JOIN lekari l ON p.krstne = l.krstne
ORDER BY p.krstne;
```

Rozdiel: EXCEPT / MINUS

- **Princíp rozdielu:** Hľadáme prvky z prvej množiny, ktoré sa nenachádzajú v druhej množine.
- **Pozor na NOT IN:** Ak vnorený dopyt vráti čo i len jednu jedinú hodnotu NULL, celý operátor NOT IN vyhodnotí podmienku ako UNKNOWN a vráti **prázdnu tabuľku**.
- **Pravidlo:** Na prepis EXCEPT vždy preferujte korelovaný dopyt s **NOT EXISTS**, ktorý hodnoty NULL ignoruje bezpečne.

```sql
-- Rozdiel pomocou NOT IN
SELECT krstne
FROM lekari
WHERE krstne NOT IN (
        SELECT krstne FROM pacienti)
ORDER BY krstne;

-- Rozdiel pomocou ALL
SELECT krstne
FROM lekari
WHERE krstne <> ALL (
        SELECT krstne FROM pacienti)
ORDER BY krstne;

-- Rozdiel pomocou NOT EXISTS
SELECT krstne
FROM lekari
WHERE NOT EXISTS (
        SELECT krstne FROM pacienti
        WHERE pacienti.krstne = lekari.krstne)
ORDER BY krstne;
```

```sql
-- Prienik pomocou EXCEPT (od verzie 8.0.31)
SELECT krstne FROM lekari
EXCEPT
SELECT krstne FROM pacienti
ORDER BY krstne;

-- Prienik pomocou LEFT OUTER JOIN
SELECT l.krstne
FROM lekari l
LEFT OUTER JOIN pacienti p ON p.krstne = l.krstne
WHERE p.idP IS NULL
ORDER BY l.krstne;
```

Zhrnutie a rady

Kategória / Konštrukcia	Hlavný účel	Výkon a optimalizácia	Kritické riziká a Best Practices
UNION vs. UNION ALL	Vertikálne spájanie výsledkov dvoch dopytov pod seba.	UNION ALL je výrazne rýchlejší (netriedi dáta).	Dopyty musia mať **rovnaký** počet a typ stĺpcov. Preferujte **UNION ALL**, ak nepotrebujete čistiť duplicity.
INTERSECT / EXCEPT	Hľadanie prieniku a rozdielu dvoch výsledných množín.	Vyžaduje triedenie / hashovanie oboch množín v pamäti.	V **starších verziách MySQL chýba**, vtedy sa nahrádza pomocou IN alebo EXISTS.
Skalárne vnútorné dopyty (=, >, <, >=, <=, <>, !=)	Dynamický výpočet jednej konkrétnej hraničnej hodnoty.	Vykoná sa iba raz pred spustením hlavného dopytu.	Vnútorný dopyt **nesmie** vrátiť viac ako 1 riadok a 1 stĺpec, inak dopyt zlyhá s runtime chybou.
Viachodnotové vnútorné dopyty (IN, ANY/SOME, ALL)	Filtrovanie riadkov oproti zoznamu vrátených hodnôt.	Dobrý výkon pri malých a stredných zoznamoch.	**Pozor na NULL**: Zápisy NOT IN a <> ALL vrátia **nula riadkov**, ak vnútorný dopyt obsahuje aspoň jednu hodnotu **NULL**.
Korelované vnútorné dopyty (EXISTS, NOT EXISTS)	Overenie existencie väzby pre každý riadok vonkajšej tabuľky.	Short-circuit: Zastaví hľadanie hneď pri prvej zhode.	**Najbezpečnejšia voľba pre negáciu** (ignoruje NULL). Pozor na **O(N^2)** zložitosť pri zložitejších výpočtoch.
Viacnásobné vnorenie (2+ úrovní)	Postupné filtrovanie dát cez viacero krokov (zdola nahor).	Optimalizátor stráca prehľad pri viac ako 3 úrovniach.	Držte vnorenie maximálne na 1-2 úrovniach kvôli čitateľnosti a optimalizácii.