

Databáze – Přednáška 4: Selekce

Michal Krátký

Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

2026/2027

Obsah přednášky

1. Zrušení tabulek ve vazbách

2. Selekcce: porovnání hodnot atributů a logické spojky

Testování na rovnost hodnoty atributu: =

Testování na nerovnost hodnoty atributu: <>

Operátory >, >=, < a <=

Logické spojky v selekci

Další operátory pro porovnání hodnot atributů

Shrnující SQL dotaz

Schéma relační databáze vs relační databáze

- ▶ Nejprve další názvosloví:
 - ▶ **Schéma relační databáze** je definice tabulek, jejich atributů a integritních omezení.
 - ▶ **Relační databáze pro dané schéma** jsou pak samotné tabulky, tedy záznamy v tabulkách.
- ▶ Proto můžeme v našem příkladě mluvit o **databázi** uživatelů, oddělení a projektů.
- ▶ Zatímco relační datový model v ERD nebo skript s posloupností příkazů `create table` budeme nazývat **schéma relační databáze**, zkráceně **schéma databáze** nebo jen **schéma**.

Zrušení tabulek ve vazbách I

- ▶ Pokusme se nyní smazat / zrušit¹ tabulky z naší databáze uživatelů, oddělení a projektů.
- ▶ Začneme tabulkou Project:

```
drop table Project;
```

```
-- cannot drop table project because other objects depend on it
-- DETAIL: constraint user_project_idp_fkey on table
--   user_project depends on table project
-- HINT: Use DROP ... CASCADE to drop the dependent objects too.
```

- ▶ Nepodařilo se, zkusme zrušit tabulku User:

```
drop table "User";
```

```
-- cannot drop table "User" because other objects depend on ...
```

- ▶ Nepodařilo se, zkusme zrušit tabulku Department:

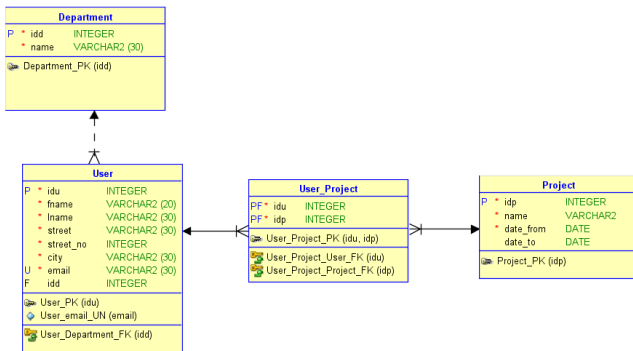
```
drop table Department;
```

```
-- cannot drop table Department because other objects depend ...
```

¹Smazání tabulky znamená zrušení tabulky, tedy smazání záznamů i schématu, nikoli pouze smazání záznamů tabulky příkazem `delete`.

Zrušení tabulek ve vazbách II

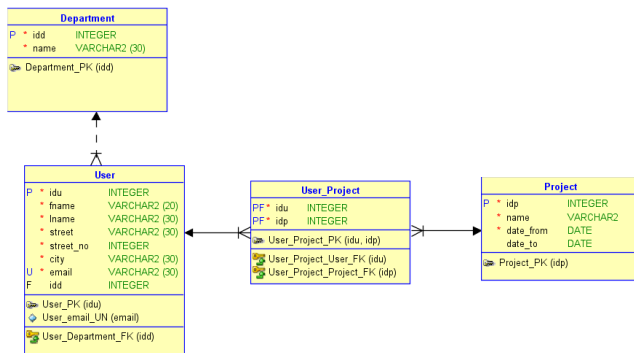
- ▶ Referenční integritní omezení, cizí klíče nám brání ve zrušení tabulek, na které se odkazuje atribut jiné tabulky.
- ▶ Připomeňme si relační datový model v ERD databáze uživatelů, oddělení a projektů:



- ▶ Tabulku Department **nemůžeme zrušit**, odkazuje se na ni, resp. její primární klíč `idd`, cizí klíč `idd` tabulky **User**.

Zrušení tabulek ve vazbách III

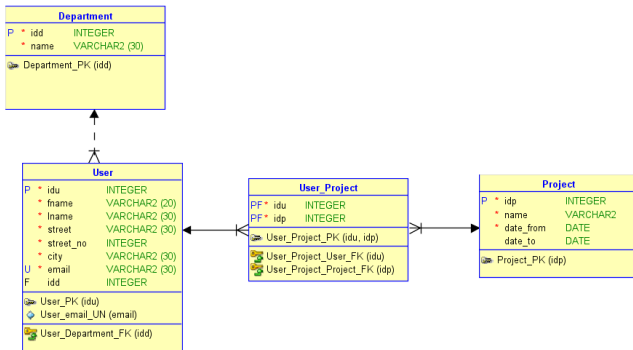
- Relační datový model v ERD databáze uživatelů a projektů:



- Podobně tabulku User **nemůžeme zrušit**, odkazuje se na ni, resp. její primární klíč idu, cizí klíč idu tabulky User_Project.
- A podobně tabulku Project **nemůžeme zrušit**, odkazuje se na ni, resp. její primární klíč idp, cizí klíč idp tabulky User_Project.

Zrušení tabulek ve vazbách IV

- Relační datový model v ERD databáze uživatelů a projektů:



- Jediná tabulka, na kterou se neodkazuje cizí klíč z jiné tabulky, je tabulka `User_Project`; musíme tedy začít touto tabulkou:

```
drop table User_Project;
-- DROP TABLE
```

Zrušení tabulek ve vazbách V

- Nebo můžeme použít variantu `drop table if exists` ze specifikace SQL99², která nebude vracet chybu, pokud tabulka neexistuje:

```
drop table User_Project;
-- table "user_project" does not exist
drop table if exists User_Project;
-- NOTICE: table "user_project" does not exist, skipping
```

- Nyní můžeme pokračovat zrušením tabulek User a Project, protože se na ně již neodkazuje cizí klíč z User_Project:

```
drop table "User";
drop table Project;
```

- A nakonec zrušíme tabulku Department, protože zrušením tabulky User (a s ní i cizího klíče idd odkazujícího primární klíč Department.idd) na Department již žádná tabulka neodkazuje:

```
drop table Department;
```

²Dříve implementováno v MySQL a PostgreSQL, v SQL Server od verze 2016, v Oracle od verze 23ai.

drop table cascade

- ▶ Příkaz `drop table ... cascade`³ náš problém řeší automaticky, jelikož zruší také všechna referenční integritní omezení, která se odkazují na rušenou tabulku:

```
drop table Project cascade;  
-- NOTICE: drop cascades to constraint user_project_idp_fkey  
--      on table user_project
```

- ▶ V některých případech pro nás ale může být takové automatické mazání integritních omezení nepřehledné.

³Specifikace SQL99.

Pojmenování referenčního integritního omezení

- ▶ V případě cyklických závislostí, kdy vhodným pořadím rušení tabulek **nemůžeme smazat** požadované tabulky databáze a chceme se vyhnout příkazu `drop table cascade, ...`
- ▶ ... musíme pojmenovat **referenční integritní omezení** a ta explicitně rušit příkazem jazyka pro definici dat `alter table` před zrušením samotné tabulky.

```
create table User_Project
(
    idu int,
    idp int,
    primary key(idu, idp),
    constraint fk_user_project_idu foreign key (idu)
        references "User"(idu),
    constraint fk_user_project_idp foreign key (idp)
        references Project(idp)
);

alter table User_Project drop constraint fk_user_project_idp;
drop table Project;
```

Selekce

- ▶ **Selekce** v SQL je operace:
 - ▶ která vybírá záznamy z tabulky, která byla sestavena v SQL příkazu konstrukcí `from` nebo `from – join`. Nejprve tedy dojde k sestavení tabulky, pak je teprve provedena **selekce** a následně, u dotazování, **projekce**.
 - ▶ je **booleovský výraz**, který se zapisuje za klauzuli `where`. Záznam z tabulky je zařazen do výsledku, pokud pro něj tento výraz nabývá hodnoty `true`.
 - ▶ která se používá v `select` pro výběr záznamů výsledku, v `update` pro výběr aktualizovaných záznamů a v `delete` pro výběr mazaných záznamů.
- ▶ V následujícím výkladu se zaměříme na selekci v dotazech:

```
select <projekce> from <table>
where <selekce>;
```

```
select <projekce> from <table>
join ...
where <selekce>;
```

Testování na rovnost hodnoty atributu: =

- ▶ Pro porovnání hodnoty (nejčastěji atributu) v selekci používáme operátor =, výsledkem jsou hodnoty true / false.
- ▶ Operátor = můžeme použít pro čísla, řetězce i hodnoty datumových datových typů.
- ▶ Mějme **dotaz** nad tabulkou se třemi uživateli s idu 1, 2 a 3: vrať uživatele s idu = 1:

```
select * from "User" where idu = 1;
```

- ▶ Booleovský výraz v selekci idu = 1 nabývá hodnoty true pouze pro záznam, kde idu je 1, u ostatních záznamů nabývá hodnoty false.
- ▶ Jelikož idu je primární klíč, ve výsledku bude pouze jeden záznam:

```
-- idu  fname  lname  street  street_no  city  email  idd
-- 1     Jan    Sobota  17. listopadu  15  Olomouc  jan.sobota01@upol.cz  4
```

Selektce v update a delete

- ▶ Podobně selekci použijeme v příkazu jazyka pro modifikaci dat update pro aktualizaci záznamů:

```
update "User" set idd=null where idu = 3;
```

- ▶ Nebo příkazu delete pro mazání záznamů:

```
delete from "User" where idu = 3;
```

- ▶ V obou případech selektce za klauzulí where vybraly jeden záznam, protože idu je primární klíč a záznam s hodnotou 3 atributu idu je v tabulce.

Testování na rovnost hodnoty atributu II

- **Dotaz:** Vrať všechny uživatele z Olomouce:

```
select * from "User" where city = 'Olomouc';
```

```
-- idu  fname  lname  street  street_no  city  ...
-- 1     Jan    Sobota  17. listopadu  15    Olomouc ...
-- 2     Yveta   Švecová Hněvotínská   215    Olomouc ...
-- 3     Petr    Švec    Hněvotínská   217    Olomouc ...
```

- **Dotaz:** Vrať všechny projekty, které začaly 1. ledna 2020:

```
select * from Project where date_from = '2020-01-01';
```

```
-- idp  name      date_from      date_to
-- 1     IS banky      2020-01-01      2021-12-31
```

Selekce po from - join, dotaz 1a

- ▶ Selekce se **provádí až po sestavení tabulky** konstrukcí from nebo from - join. Projekce se provádí jako poslední.
- ▶ **Dotaz:** Vrať id projektů s názvem 'Spisová služba' a id, jméno a příjmení jejich uživatelů:
- ▶ Nejprve sestavíme tabulku pro **dotaz:** Vrať projekty a jejich uživatele:

```
select * from Project p
join User_Project up on p.idp = up.idp
join "User" u on up.idu = u.idu;
```

- ▶ **Výsledek:**

idp	name	...	idu	idp	idu	fname	lname	...	idd
1	IS banky	...	1	1	1	Jan	Sobota	...	4
2	IS firmy	...	1	2	1	Jan	Sobota	...	4
3	IS úřadu	...	1	3	1	Jan	Sobota	...	4
4	Spisová služba	...	1	4	1	Jan	Sobota	...	4
1	IS banky	...	2	1	2	Yveta	Švecová	...	3
2	IS firmy	...	2	2	2	Yveta	Švecová	...	3

Selekce po from - join, dotaz 1b

- **Dotaz:** Vrať id projektů s názvem 'Spisová služba' a id, jméno a příjmení jejich uživatelů:
- Pak přidáme selekci pouze na projekty s názvem 'Spisová služba':

```
select * from Project p
join User_Project up on p.idp = up.idp
join "User" u on up.idu = u.idu
where p.name = 'Spisová služba';
```

- **Výsledek:**

idp	name	...	idu	idp	idu	fname	lname	...	idd
4	Spisová služba	...	1	4	1	Jan	Sobota	...	4

Selektce po from - join, dotaz 1c

- ▶ **Dotaz:** Vrať id projektů s názvem 'Spisová služba' a id, jméno a příjmení jejich uživatelů:
- ▶ A nakonec zařadíme projekci na atributy Project.idp, User.idu, User.fname a User.lname:

```
select p.idp, u.idu, u.fname, u.lname
from Project p
join User_Project up on p.idp = up.idp
join "User" u on up.idu = u.idu
where p.name = 'Spisová služba';
```

- ▶ **Výsledek:**

idp	idu	fname	lname
4	1	Jan	Sobota

Selekce po from - join, dotaz 2

- Pozor na **zbytečné použití operace join**.
- **Původní zadání dotazu** „Vrať id projektů s názvem 'Spisová služba' a id, jméno a příjmení jejich uživatelů“ **nepatrně změníme na:** „Vrať id projektů s názvem 'Spisová služba' a id jejich uživatelů.“
- V tomto případě do výsledku nechceme zařadit hodnoty atributů z tabulky User kromě primárního klíče idu, který je ale také ve vazební tabulce User_Project. Spojení s tabulkou User tedy můžeme vynechat, jen v projekci uvedeme User_Project.idu namísto User.idu:

```
select p.idp, up.idu
from Project p
join User_Project up on p.idp = up.idp
where p.name = 'Spisová služba';
```

- **Výsledek:**

idp	idu
4	1

Testování na hodnotu null: is null

- ▶ Jak víme z předchozích přednášek, výsledkem porovnání s hodnotou null nejsou hodnoty true / false, ale hodnota unknown.
- ▶ Pro testování na hodnotu null musíme použít operátor is null.
- ▶ **Dotaz:** Vrať uživatele, kteří nejsou na žádném oddělení (případně kteří nemají zadáno žádné oddělení).

```
select * from "User" where idd is null;
```

- ▶ **Výsledek:**

```
-- idu  fname  lname  street  street_no  city  email  idd
-- 3      Petr   Švec   Hněvotínská  217      Olomouc  petr.svec02@upol.cz  NULL
```

Testování na nerovnost hodnoty atributu: <>

- ▶ Pro testování na **nerovnost hodnoty atributu** v selekci můžeme použít:
 - ▶ booleovský výraz s negací not a testováním na rovnost =
 - ▶ nebo přímo operátor <>.
- ▶ Výsledkem je true, pokud se hodnoty nerovnají, a opačně.
- ▶ Operátor <> můžeme použít pro čísla, řetězce i hodnoty datumových datových typů.

Testování na nerovnost hodnoty atributu: <>

- Mějme dotaz nad tabulkou se třemi uživateli s idu 1, 2 a 3: vrať uživatele, kteří nemají idu = 1:

```
select * from "User" where not idu = 1;
```

```
select * from "User" where idu <> 1;
```

- Booleovský výraz v selekci idu <> 1 nabývá hodnoty false pouze pro záznam, kde idu je 1, u ostatních záznamů nabývá hodnoty true.
- Jelikož idu je primární klíč a záznamy s idu=1 je v tabulce, ve výsledku budou všechny záznamy z tabulky kromě jednoho záznamu:

-- idu	fname	lname	street	street_no	city	email	idd
-- 2	Yveta	Švecová	Hněvotínská	215	Olomouc	yveta.svecova01@upol.cz	3
-- 3	Petr	Švec	Hněvotínská	217	Olomouc	petr.svec02@upol.cz	NULL

Testování na nerovnost hodnoty null: is not null

- ▶ Při testování na nerovnost hodnoty null můžeme použít buď negaci not výsledku operátoru is null, nebo přímo operátor is not null, který vrací true, pokud hodnota na levé straně (nejčastěji atributu) není null.
- ▶ **Dotaz:** Vrať uživatele, kteří mají přiřazeno nějaké oddělení.

```
select * from "User" where not idd is null;
```

```
select * from "User" where idd is not null;
```

- ▶ **Výsledek:**

idu	fname	lname	street	street_no	city	email	idd
1	Jan	Sobota	17. listopadu	15	Olomouc	jan.sobota01@upol.cz	4
2	Yveta	Švecová	Hněvotínská	215	Olomouc	yveta.svecova01@upol.cz	3

Příklady, které nemůžeme řešit testováním na nerovnost I

- ▶ Pozor na, na první pohled správná, **chybná** řešení.
- ▶ **Dotaz:** Vrať id uživatelů, kteří nepracují na projektu s id 1.
- ▶ Na první pohled se zdá, že správné řešení je triviálně jednoduché:

```
select idu from User_Project where idp <> 1;
```

- ▶ **Výsledek:**

```
idu  
1  
1  
1  
2
```

Příklady, které nemůžeme řešit testováním na nerovnost II

- ▶ Ještě odstraníme duplicitní záznamy výsledku pomocí `distinct`:

```
select distinct idu from User_Project
where idp <> 1;
```

- ▶ Výsledek:

```
idu
2
1
```

- ▶ A zjistíme, že výsledek je **chybný**, přestože jsme pro opačný dotaz: „Vrať id uživatelů, kteří pracují na projektu s id 1“, kde řešení je správně takto:

```
select idu from User_Project where idp = 1;
```

- ▶ Výsledek:

```
idu
1
2
```

jsem jen provedli negaci podmínky v selekci.

Příklady, které nemůžeme řešit testováním na nerovnost III

- ▶ Pro **dotaz**: Vrať id uživatelů, kteří nepracují na projektu s id 1.
- ▶ Jsme získali id uživatelů 1 a 2, nicméně tabulka User_Project obsahuje tyto záznamy:

idu	idp
1	1
1	2
1	3
1	4
2	1
2	2

- ▶ Tzn. na projektu s idp 1 pracují uživatelé s idu 1 a 2, jediný uživatel, který nepracuje na tomto projektu má idu = 3.

Příklady, které nemůžeme řešit testováním na nerovnost IV

- ▶ Místo **dotazu**: Vrať id uživatelů, kteří nepracují na projektu s id 1,
- ▶ jsme vyřešili **dotaz**: Vrať id uživatelů pro záznamy tabulky User_Project, kde idp <> 1, bez projekce na distinct idu:

```
select * from User_Project where idp <> 1;
-- idu  idp
-- 1      2
-- 1      3
-- 1      4
-- 2      2 -- chybi zaznamy: 1 1 a 2 1
```

- ▶ Kvůli vazbě 1 : N, tzn. uživatel může pracovat na více projektech, nám ve výsledku zůstali uživatelé 1 (pracuje ještě na projektech 2, 3 a 4) a 2 (pracuje ještě na projektu 2), ale uživatele 3 jsme tím navíc do výsledku nedostali, protože na žádném projektu nepracuje.
- ▶ Tento problém musíme řešit pomocí **množinového rozdílu**, jak si ukážeme později.

Operátory >, >=, < a <=

- ▶ Operátory >, >=, < a <= používáme v **selekci** pro datové typy, na kterých je definováno **uspořádání**, tj. celočíselné i reálné datové typy, řetězce (tzv. lexikografické – slovníkové uspořádání) i datumové a časové typy.
- ▶ **Dotaz:** Vrať oddělení s idd větší než 2.

```
select * from Department where idd > 2;  
select * from Department where idd >= 3;
```

```
-- idd    name  
-- 3      IT oddělení  
-- 4      Výroba
```

Operátory >, >=, < a <=

- **Dotaz:** Vrať projekty, jejichž názvy jsou v lexikografické (slovníkovém) uspořádání za řetězcem 'IS banky'.

```
select * from Project where name > 'IS banky';
```

-- idp	name	date_from	date_to
-- 2	IS firmy	2022-01-01	2023-12-31
-- 3	IS úřadu	2024-01-01	2025-12-31
-- 4	Spisová služba	2026-01-01	NULL

Ve výsledku není záznam projektu s názvem 'IS banky'.

- **Dotaz:** Vrať projekty, které začaly po datu '2022-01-01' včetně.

```
select * from Project where date_from >= '2022-01-01';
```

-- idp	name	date_from	date_to
-- 2	IS firmy	2022-01-01	2023-12-31
-- 3	IS úřadu	2024-01-01	2025-12-31
-- 4	Spisová služba	2026-01-01	NULL

Ve výsledku není záznam se začátkem '2020-01-01'.

Logické spojky v selekci

- ▶ Jak již bylo řečeno, **selektce je tvořena booleovským výrazem**; do výsledku jsou zařazeny všechny záznamy, pro které tento výraz nabývá hodnoty true.
- ▶ V booleovském výrazu můžeme používat **známé logické spojky**: **logický součet** – nebo (or) a **logický součin** – a zároveň (and).
- ▶ **Pořadí vyhodnocení** operátorů je následující: not, and, or. Když si nejsme jisti pořadím vyhodnocení, **používáme závorky**.

Logické spojky and a or

- Uvažujme tyto SQL dotazy:

```
select * from "User"  
where fname = 'Jan' and city = 'Olomouc';
```

```
select * from "User"  
where fname = 'Jan' or city = 'Olomouc';
```

- Připomeňme si pravdivostní tabulku logických funkcí or a and:

<i>A</i>	<i>B</i>	<i>A or B</i>	<i>A and B</i>
false	false	false	false
false	true	true	false
true	false	true	false
true	true	true	true

Logické spojky and a or

- Ve **výsledku dotazu s and** bude pouze jeden záznam:

```
-- idu  fname  lname  street  street_no  city  email  idd
-- 1    Jan    Sobota  17. listopadu  15    Olomouc  jan.sobota01@upol.cz  4
```

Protože pouze jeden záznam obsahuje hodnotu atributu fname 'Jan' a hodnotu atributu city 'Olomouc'.

- Zatímco ve **výsledku dotazu s or** budou tři záznamy:

```
-- idu  fname  lname  street  street_no  city  email  idd
-- 1    Jan    Sobota  17. listopadu  15    Olomouc  jan.sobota01@upol.cz  4
-- 2    Yveta  Švecová  Hněvotínská  215    Olomouc  yveta.svecova01@upol.cz  3
-- 3    Petr    Švec    Hněvotínská  217    Olomouc  petr.svec02@upol.cz  NULL
```

Protože pro tři záznamy nabývá hodnota výrazu city='Olomouc' hodnotu true, u prvního dokonce i výraz fname='Jan'.

Logické spojky **and** a **or** nad jedním atributem

- ▶ Zatímco má smysl vybírat záznamy pomocí **or** nad jedním atributem, například:

```
select * from "User"  
where fname='Jan' or fname='Petr';
```

Protože chceme všechny záznamy, kde hodnota fname je 'Jan' nebo 'Petr'.

- ▶ Podobný **booleovský výraz s and** vrací vždy false:

```
select * from "User"  
where fname='Jan' and fname='Petr';
```

Protože hodnota atributu jednoho záznamu nemůže nikdy nabývat dvou různých hodnot současně.

Konstrukce in

- ▶ Pokud **požadujeme záznamy s různými hodnotami jednoho atributu**, pak je úspornější použít **konstrukci in** namísto několika logických spojek or.
- ▶ **Výsledkem in je hodnota true**, pokud hodnota na levé straně obsahuje jednu z hodnot v seznamu za in.
- ▶ **Dotaz:** Vrať uživatele, jejichž jméno je 'Jan', 'Petr' nebo 'Yveta'.
- ▶ Následující **řešení** jsou ekvivalentní:

```
select * from "User"
where fname = 'Jan' or fname = 'Petr' or fname = 'Yveta';
```

```
select * from "User"
where fname in ('Jan', 'Petr', 'Yveta');
```

- ▶ **Výsledek** obsahuje všechny tři záznamy tabulky, jelikož uživatelé s jinými jmény v tabulce nejsou.

DeMorganova pravidla pro booleovské výrazy I

- ▶ **Pozor na booleovské výrazy s negací.**
- ▶ **Dotaz:** Vrať uživatele, kteří se nejmenují 'Jan' nebo 'Petr'.
- ▶ Pokud budeme následovat bez rozmyšlení tuto větu, snadno se dostaneme k chybnému řešení:

```
select * from "User"  
where fname <> 'Jan' or fname <> 'Petr';
```

- ▶ **Výsledek chybně** obsahuje všechny tři záznamy tabulky, protože dojde k následujícímu vyhodnocení:
 - ▶ Záznam s 'Jan': ('Jan'<>'Petr')=true (celý výraz je true).
 - ▶ Záznam s 'Petr': ('Petr'<>'Jan')=true (celý výraz je true).
 - ▶ Záznam s 'Yveta': obě porovnání jsou true (celý výraz je true).

DeMorganova pravidla pro booleovské výrazy II

- Dotaz, správně napsaný, je totiž ve skutečnosti tento:
Vrať uživatele, kteří se nejmenují ('Jan' nebo 'Petr').

- Což vede ke správným ekvivalentním řešením:

```
select * from "User"
where fname not in ('Jan', 'Petr');
```

```
select * from "User"
where not fname in ('Jan', 'Petr');
```

```
select * from "User"
where not (fname = 'Jan' or fname = 'Petr');
```

- Kde používáme postupně: not in, negaci výsledku in a negaci výsledku or. Řešení s **not in** je zjevně nejpráhlednější.
- Výsledek obsahuje vždy jediný záznam:

idu	fname	lname	street	street_no	city	email	idd
2	Yveta	Švecová	Hněvotínská	215	Olomouc	yveta.svecova01@upol.cz	3

DeMorganova pravidla pro booleovské výrazy III

- Uvažujme nyní správné řešení:

```
select * from "User"
where not (fname = 'Jan' or fname = 'Petr');
```

- A pokusme se jej převést pomocí DeMorganových pravidel na výraz s **and**, dostáváme:

```
select * from "User"
where fname <> 'Jan' and fname <> 'Petr';
```

-- původní, chybné řešení

```
where fname <> 'Jan' or fname <> 'Petr';
```

- Protože:

(A or B) = not (not A and not B)

not (A or B) = (not A and not B)

= not (fname = 'Jan') and not (fname = 'Petr')

= fname <> 'Jan' and fname <> 'Petr'

not in: pozor na null

- **Dotaz:** Vrať id uživatele a id jeho oddělení, kde idd není 3 nebo 4:

```
select idu, idd from "User"  
where idd not in (3, 4);  
-- výsledek je prázdný
```

- Pokud chceme ve výsledku i záznamy uživatelů bez oddělení (idd je null), musíme to řešit tímto dotazem, jelikož porovnání s null není hodnota true / false, ale unknown:

```
select idu, idd from "User"  
where idd not in (3, 4) or idd is null;  
  
-- idu    idd  
-- 3      NULL
```

Konstrukce between and

- ▶ V případě dotazů na **rozsah hodnot atributu**, například:

```
select * from Department
where idd >= 1 and idd <= 3;
```

- ▶ Můžeme použít přehlednější konstrukci between and:

```
select * from Department
where idd between 1 and 3;
```

- ▶ Podobně můžeme uvažovat řešení **dotazu**: Vrať projekty, kde začátek je mezi daty '2020-01-01' a '2020-06-30':

```
select * from Project
where
    date_from >= '2020-01-01' and date_from <= '2020-06-30';
```

```
select * from Project
where date_from between '2020-01-01' and '2020-06-30';
```

```
-- idp  name      date_from      date_to
-- 1    IS banky    2020-01-01      2021-12-31
```

Operátor like

- ▶ Při dotazování na části řetězcových atributů používáme operátor **like** a znak % ve významu libovolného počtu libovolných znaků.

- ▶ **Dotaz:** Vrať uživatele jejichž příjmení začíná znakem 'S':

```
select * from "User"
where lname like 'S%';
-- 1      Jan      Sobota  17. listopadu  15      ...
```

- ▶ **Dotaz:** Vrať uživatele jejichž příjmení končí řetězcem 'ová':

```
select * from "User"
where lname like '%ová';
-- 2      Yveta    Švecová Hněvotínská  215      ...
```

- ▶ **Dotaz:** Vrať projekty jejichž název obsahuje řetězec 'fi':

```
select * from Project
where name like '%fi%';
-- 2      IS firmy      2022-01-01      2023-12-31
```

Shrnující SQL dotaz I

- ▶ **Dotaz:** Vrať id a jména uživatelů, id a názvy jejich oddělení a id a názvy jejich projektů, kde projekt začal a skončil mezi daty '2020-01-01' a '2021-12-31', uživatel bydlí v Olomouci a oddělení obsahuje řetězec 'IT'. (Pro uživatele, kteří mají zadáno nějaké oddělení).
- ▶ **Dotaz řešíme postupně**, nejprve spojíme všechny potřebné tabulky, tj. řešíme **dotaz**: „Vrať uživatele, jejich oddělení a jejich projekty. (Pro uživatele, kteří mají zadáno nějaké oddělení).“

```
select * from "User" u
join Department d on u.idd = d.idd
join User_Project up on u.idu = up.idu
join Project p on up.idp = p.idp;
```

- ▶ **Část zadání:** „Pro uživatele, kteří mají zadáno nějaké oddělení.“ je vyřešena použitím **vnitřního spojení (join)**.

Shrnující SQL dotaz II

- Dotaz řešíme postupně, nyní přidáme podmínku do selekce: „kde projekt začal a skončil mezi daty '2020-01-01' a '2021-12-31', uživatel bydlí v Olomouci a oddělení obsahuje řetězec 'IT'".

```
select * from "User" u
join Department d on u.idd = d.idd
join User_Project up on u.idu = up.idu
join Project p on up.idp = p.idp
where
  p.date_from between '2020-01-01' and '2021-12-31' and
  p.date_to between '2020-01-01' and '2021-12-31' and
  u.city = 'Olomouc' and d.name like '%IT%';
```

Shrnující SQL dotaz III

- Dotaz řešíme postupně, nyní přidáme projekci: „id a jména uživatelů, id a názvy jejich oddělení a id a názvy jejich projektů“.

```
select u.idu, u.fname, u.lname, d.idd, d.name,
       p.idp, p.name
from "User" u
join Department d on u.idd = d.idd
join User_Project up on u.idu = up.idu
join Project p on up.idp = p.idp
where
  p.date_from between '2020-01-01' and '2021-12-31' and
  p.date_to between '2020-01-01' and '2021-12-31' and
  u.city = 'Olomouc' and d.name like '%IT%';
```

```
-- idu fname lname idd name idp name
-- 2 Yveta Švecová 3 IT oddělení 1 IS banky
```