Kapitola 4: SQL

- Základní struktura
- Množinové operace
- Souhrnné funkce
- Nulové hodnoty
- Vnořené poddotazy (Nested sub-queries)
- Odvozené relace
- Pohledy
- Modifikace databáze
- Spojené relace
- Jazyk definice dat (Data definition language)
- Vložený SQL

Základní struktura

- SQL je založen na množinových a relačních operacích s několika modifikacemi a vylepšeními
- Typický SQL dotaz má tvar:

  select A_1, A_2, …, A_n
  from r_1, r_2, …, r_m
  where P

  – A_i reprezentuje atributy
  – r_i reprezentuje relace
  – P je predikát
- Tento dotaz je ekvivalentní následujícímu výrazu relační algebry:

$$\Pi_{A_1, A_2, \ldots, A_n} (\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

- Výsledek každého SQL dotazu je relace.

Klauzule select

- Klauzule **select** odpovídá operaci projekce v relační algebře. Je používána k vypsání požadovaných atributů ve výsledku dotazu.
- Najděte jména všech poboček v relaci *půjčka*

  select *pobočka-jméno*
  from *půjčka*

  V syntaxi „čisté“ relační algebry se tento dotaz zapíše takto:

$$\Pi_{pobočka\text{-}jméno} (půjčka)$$

- Hvězdička značí „všechny atributy“

  select *
  from *půjčka*
- SQL umožňuje duplikáty v relacích i ve výsledcích dotazů.
- Pro eliminaci duplikátů vložte klíčové slovo **distinct** za **select**.
- Najděte jména všech poboček v relaci *půjčka* a odstraňte duplikáty

  select distinct *pobočka-jméno*
  from *půjčka*

- Klíčové slovo **all** specifikuje, že duplikáty odstraněny nebudou.
  select all *pobočka-jméno*
  from *půjčka*
- Klauzule **select** může obsahovat aritmetické výrazy s operátory +, -, * a / na konstantách nebo atributech n-tic.
- Dotaz
  select *pobočka-jméno, půjčka-číslo, částka* * 100
  from *půjčka*
  vrátí relaci shodnou s relací *půjčka*, jen atribut *částka* je vynásoben číslem 100

Klauzule where

- Klauzule **where** odpovídá operaci výběr v relační algebře. Skládá se z predikátu zahrnujícího atributy relací, které jsou v klauzuli **from**.
- Najděte všechna čísla půjček pro půjčky v pobočce Praha-Spořilov s částkami většími než $1200.
  select *půjčka-číslo*
  from *půjčka*
  where *pobočka-jméno* = „Praha-Spořilov“ **and** *částka* > 1200
- SQL používá logické spojky **and**, **or** a **not**. Umožňuje tak použití aritmetických výrazů jako operandů pro operátory porovnání.
- SQL zahrnuje operátor porovnání **between** pro zjednodušení klauzule **where**, který specifikuje hodnotu z určitého intervalu.
- Najděte čísla půjček s částkami mezi $90,000 a $100,000 (tj. ≥$90,000 a ≤$100,000)
  select *půjčka-číslo*
  from *půjčka*
  where *částka* **between** 90000 **and** 100000

Klauzule from

- Klauzule **from** odpovídá kartézskému součinu z relační algebry. Vypíše relace, které mají být procházeny při vyhodnocování výrazu.
- Najděte kartézský součin *půjčovatel* × *půjčka*
  select *
  from *půjčovatel, půjčka*
- Najděte jméno a číslo půjčky všech zákazníků, kteří mají půjčku v pobočce Praha-Spořilov.
  select distinct *zákazník-jméno, půjčovatel.půjčka-číslo*
  from *půjčovatel, půjčka*
  where *půjčovatel.půjčka*-číslo = *půjčka.půjčka-číslo*
  and pobočka-*jméno* = „Praha-Spořilov“

Operace přejmenování

- Mechanismus pro přejmenovávání relací je v SQL řešen pomocí klauzule **as**:
  staré-jméno **as** *nové-jméno*
- Najděte jméno a číslo půjčky všech zákazníků, kteří mají půjčku v pobočce Praha-Spořilov; nahraďte jméno sloupce *půjčka-číslo* jménem *půjčka-id*.
  select distinct *zákazník-jméno, půjčovatel.půjčka číslo* **as** *půjčka-id*
  from *půjčovatel, půjčka*
  where *půjčovatel.půjčka-číslo* = *půjčka.půjčka-číslo*
  and *pobočka-jméno* = „Praha-Spořilov“

Proměnné n-tic (Tuple variables)

- Proměnné n-tic jsou definovány v klauzuli **from** pomocí klauzule **as**.
- Najděte jména zákazníků a čísla jejich půjček pro všechny zákazníky, kteří mají půjčku ve stejné pobočce.
  select distinct *zákazník-jméno, T.půjčka-číslo*
  from *půjčovatel* **as** *T*, *půjčka* **as** *S*
  where *T.půjčka-číslo* = *S.půjčka-číslo*
- Najděte jména všech poboček, které mají větší aktiva než nějaká pobočka v Praze.
  select distinct *T.pobočka-jméno*
  from *pobočka* **as** *T*, *pobočka* **as** *S*
  where *T.aktiva* > *S.aktiva* **and** *S.pobočka-město* = „Praha“

Operace s řetězci

- SQL zahrnuje operátor pro porovnávání řetězců znaků. Vzorky jsou popsány použitím dvou speciálních znaků:
  - procento (%). Znak % vyhovuje jakémukoliv podřetězci.
  - podtržítko (_). Znak _ vyhovuje jakémukoliv znaku.
- Najděte jména všech zákazníků, jejichž ulice obsahuje podřetězec ‚Main‘.
  select *zákazník-jméno*
  from *zákazník*
  where *zákazník-ulice* **like** „%Main%“
- Co vyhovuje jménu „Main%“?
  like „Main\%“ **escape** „\“

Uspořádání zobrazení n-tic

- Výpis všech zákazníků, kteří mají půjčku v pobočce Praha-Spořilov seřazený podle abecedy

  select distinct *zákazník-jméno*
  from *půjčovatel, půjčka*
  where *půjčovatel.půjčka-číslo* = *půjčka.půjčka-číslo* **and**
  pobočka-jméno = „Praha-Spořilov“
  order by *zákazník-jméno*
- Můžeme ještě specifikovat **desc** pro sestupné pořadí nebo **asc** pro vzestupné pořadí přidáním na konec dotazu; vzestupné pořadí je implicitní.
- SQL musí provést řazení, aby vyhověl požadavku **order by**. Protože řazení velkého množství n-tic může být drahé, je vhodné řadit pouze v nezbytných případech.

Duplikáty

- V relaci s duplikáty může SQL definovat kolik kopií n-tic se objeví ve výsledku.
- *Více-množinové* verze některých operátorů relační algebry – mějme více-množinové relace r_1 a r_2:
  - Je-li c_1 kopií n-tice t_1 v r_1 a t_1 vyhovuje výběru σ_θ, pak je c_1 kopií n-tice t_1 ve výsledku $\sigma_\theta(r_1)$
  - Pro každou kopii n-tice t_1 v r_1 je kopie n-tice $\Pi_A(t_1)$ v $\Pi_A(r_1)$, kde $\Pi_A(t_1)$ značí projekci jednoduché n-tice t_1.
  - Je-li c_1 kopií n-tice t_1 v r_1 a c_2 kopií n-tice t_2 v r_2, pak je $c_1 \times c_2$ kopií n-tice $t_1.t_2$ v $r_1 \times r_2$.
- Předpokládejme, že relace r_1 se schématem (*A, B*) a r_2 se schématem (C) jsou následující více-množiny:

  $r_1 = \{(1, a), (2, a)\}$ $r_2 = \{(2), (3), (3)\}$
- Pak $\Pi_B(r_1)$ bude $\{(a), (a)\}$, zatímco $\Pi_B(r_1) \times r_2$ bude

  $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$
- SQL sémantika duplikátů:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  je ekvivalentní *více-množinové* verzi výrazu:

  $$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

Množinové operace

- Množinové operace **union**, **intersect** a **except** operují na relacích a odpovídají operacím relační algebry $\cup$, $\cap$ a –.
- Každá z těchto operací automaticky eliminuje duplikáty; chceme-li i duplikáty, použijeme více-množinové verze **union all**, **intersect all** a **except all**. Předpokládejme, že se n-tice objeví *m*-krát v *r* a *n*-krát v *s*, pak se objeví:
  - *(m + n)*-krát v *r* **union all** *s*
  - *min(m, n)*-krát v *r* **intersect all** *s*
  - *max(0, m – n)*-krát v *r* **except all** *s*

- Najděte všechny zákazníky, kteří mají půjčku, účet nebo obojí:
  (**select** *zákazník-jméno* **from** *vkladatel*)
  union
  (**select** *zákazník-jméno* **from** *půjčovatel*)
- Najděte všechny zákazníky, kteří mají půjčku i účet:
  (**select** *zákazník-jméno* **from** *vkladatel*)
  intersect
  (**select** *zákazník-jméno* **from** *půjčovatel*)
- Najděte všechny zákazníky, kteří mají účet, ale ne půjčku:
  (**select** *zákazník-jméno* **from** *vkladatel*)
  except
  (**select** *zákazník-jméno* **from** *půjčovatel*)

Souhrnné funkce

Tyto funkce operují na více-množinových hodnotách sloupců relace a vracejí hodnotu
avg: průměrná hodnota
min: minimální hodnota
max: maximální hodnota
sum: suma hodnot
count: počet hodnot

- Najděte průměrnou hodnotu zůstatku účtu v pobočce Praha-Spořilov.
  select avg (*zůstatek*)
  from *účet*
  where *pobočka-jméno* = „Praha-Spořilov“
- Najděte počet n-tic v relaci *zákazník*.
  select count (*)
  from *zákazník*
- Najděte počet vkladatelů v bance.
  select count (**distinct** *zákazník-jméno*)
  from *vkladatel*

Souhrnné funkce – Group by

- Najděte počet vkladatelů v každé pobočce.
  select *pobočka-jméno*, **count** (**distinct** *zákazník-jméno*)
  from *vkladatel, účet*
  where *vkladatel.číslo-účtu = účet.číslo-účtu*
  group by *pobočka-jméno*

  Poznámka: Atributy v klauzuli **select** mimo souhrnné funkce se musí objevit v seznamu **group by**!!!

Souhrnné funkce – klauzule having

- Najděte jména všech poboček, jejichž průměrný zůstatek účtu je větší než $1,200.

  select *pobočka-jméno*, **avg** (*zůstatek*)
  from *účet*
  group by *pobočka-jméno*
  having avg (*zůstatek*) > 1200

  Poznámka: predikáty v klauzuli **having** jsou po zformování skupin

Nulové hodnoty

- N-tice můžou mít nulovou hodnotu, značenou *null*, ta znamená neznámou hodnotu nebo hodnotu, která neexistuje.
- Výsledek jakékoliv aritmetické operace zahrnující *null* je *null*.
- Všechna porovnávání zahrnující *null* vrací *false*. Precizněji:
  - Jakékoliv porovnání s *null* vrací *unknown*
  - (*true* or *unknown*) = *true* (*false* or *unknown*) = *unknown*
    (*unknown* or *unknown*) = *unknown* (*true* and *unknown*) = *unknown*
    (*false* and *unknown*) = *false* (*unknown* and *unknown*) = *unknown*
  - Výsledek klauzule **where** je brán jako *false*, je-li *unknown*
  - „*P* **is unknown**“ se vyhodnotí jako *true*, jestliže predikát *P* je vyhodnocen jako *unknown*
- Najděte všechna čísla půjček, které se objeví v relaci *půjčka* s nulovou hodnotou *částka*.

  select *půjčka-číslo*
  from *půjčka*
  where *částka* **is null**
- Součet všech částek půjček

  select sum (*částka*)
  from *půjčka*

  Tyto výrazy ignorují nulové částky; výsledek je nula, jestliže zde není žádná nenulová částka.
- Všechny souhrnné operace s výjimkou **count(*)** ignorují n-tice s nulovými hodnotami na souhrnných atributech.

Vnořené poddotazy (Nested subqueries)

- SQL poskytuje mechanismus pro vnořování poddotazů.
- Poddotaz je výraz **select-from-where**, který je vnořen do jiného dotazu.
- Obvyklé použití poddotazů je provádění testů na členství v množině, porovnávání množin a kardinalitu množin.

Členství v množině

- F **in** $r \Leftrightarrow \exists\, t \in r\ (t = \mathrm{F})$

(5 **in** | 0 / 4 / 5 |) = true

(5 **in** | 0 / 4 / 6 |) = false

(5 **not in** | 0 / 4 / 6 |) = true

Příklady dotazů

- Najděte všechny zákazníky, kteří mají v bance účet i půjčku.

```
select distinct zákazník-jméno
from půjčovatel
where zákazník-jméno in (select zákazník-jméno
                         from vkladatel)
```

- Najděte všechny zákazníky, kteří mají v bance půjčku, ale ne účet.

```
select distinct zákazník-jméno
from půjčovatel
where zákazník-jméno not in (select zákazník-jméno
                             from vkladatel)
```

- Najděte všechny zákazníky, kteří mají účet i půjčku v pobočce Praha-Spořilov.

```
select distinct zákazník-jméno
from půjčovatel, půjčka
where půjčovatel.půjčka-číslo = půjčka.půjčka-číslo and
      pobočka-jméno = „Praha-Spořilov“ and
      (pobočka-jméno, zákazník-jméno) in
            (select pobočka-jméno, zákazník-jméno
            from vkladatel, účet
            where vkladatel.číslo-účtu = účet.číslo-účtu)
```

Porovnávání množin

- Najděte všechny pobočky, které mají větší aktiva než nějaká pobočka v Praze.

```
select distinct T.pobočka-jméno
from pobočka as T, pobočka as S
where T.aktiva > S.aktiva and S.pobočka-město = „Praha“
```

Klauzule some

- F <comp> **some** $r \Leftrightarrow \exists\, t\, (t \in r \wedge$ [F <comp> t])

kde <comp> může být: <, ≤, >, ≥, =, ≠

(5 < **some** | 0 / 5 / 6 |) = true (čteme: 5 je menší než nějaká n-tice z relace

(5 < **some** | 0 / 5 |) = false

(5 = **some** | 0 / 5 |) = true

(5 ≠ **some** | 0 / 5 |) = true (neboť 0 ≠ 5)

- (= **some**) je stejné jako **in**
- Ale: (≠ **some**) je různé od **not in** !!!

Příklad dotazu

- Najděte pobočky, které mají větší aktiva než nějaká pobočka v Praze.

```
select pobočka-jméno
from pobočka
where aktiva > some
        (select aktiva
        from pobočka
        where pobočka-město = „Praha“)
```

Klauzule all

- F <comp> **all** $r \Leftrightarrow \forall\, t\, (t \in r \wedge$ [F <comp> t])

(5 < **all** | 0 / 5 / 6 |) = false

(5 < **all** | 6 / 10 |) = true

(5 = **all** | 4 / 5 |) = false

(5 ≠ **all** | 4 / 6 |) = true (neboť 5 ≠ 4 **and** 5 ≠ 6)

- (≠ **all**) je stejné jako **not in**
- Ale: (**= all**) je různé od **in** !!!

Příklad dotazu

- Najděte pobočky, které mají větší aktiva než všechny pobočky v Praze.

```
select pobočka-jméno
from pobočka
where aktiva > all
        (select aktiva
        from pobočka
        where pobočka-město = „Praha“)
```

Test na prázdné relace

- Konstrukce **exists** vrací hodnotu **true**, je-li poddotaz v argumentu neprázdný.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

Příklad dotazu

- Najděte všechny zákazníky, kteří mají účet ve všech pobočkách v Praze.

```
select dictinct S.zákazník-jméno
from vkladatel as S
where not exists (
        (select pobočka-jméno
        from pobočka
        where pobočka-město = „Praha“)
        except
        (select R.pobočka-jméno
        from vkladatel as T, účet as R
        where T.číslo-účtu = R.číslo-účtu and
                S.zákazník-jméno = T.zákazník-jméno))
```

- Poznámka: $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

Test na nepřítomnost duplikátních n-tic

- Konstrukce **unique** testuje, zda poddotaz má ve svém výsledku nějaké duplikátní n-tice.
- Najděte všechny zákazníky, kteří mají pouze jeden účet v pobočce Praha-Spořilov.

```
select T.zákazník-jméno
from vkladatel as T
where unique (
        select R.zákazník-jméno
        from účet, vkladatel as R
        where T.zákazník-jméno = R.zákazník-jméno and
                R.číslo-účtu = účet.číslo-účtu and
                účet.pobočka-jméno = „Praha-Spořilov“)
```

Příklad dotazu

- Najděte všechny zákazníky, kteří mají alespoň dva účty v pobočce Praha-Spořilov.

```
select T.zákazník-jméno
from vkladatel as T
where not unique (
        select R.zákazník-jméno
        from účet, vkladatel as R
        where T.zákazník-jméno = R.zákazník-jméno and
                R.číslo-účtu = účet.číslo-účtu and
                účet.pobočka-jméno = „Praha-Spořilov“)
```

Odvozené relace

- Najděte průměrný zůstatek účtu v těch pobočkách, kde je průměrný zůstatek větší než $1200.

```
select pobočka-jméno, prům-zůstatek
from (select pobočka-jméno, avg (zůstatek)
        from účet
        group by pobočka-jméno)
        as result (pobočka-jméno, prům-zůstatek)
where prům-zůstatek > 1200
```

Poznámka: nepotřebujeme použít klausuli **having**, jestliže v klauzuli **from** spočítáme dočasnou relaci *result* a její atributy můžeme použít přímo v klauzuli **where**.

Pohledy

- Poskytují mechanismus, jak skrýt nějaká data před nějakými uživateli. Pro vytvoření pohledu použijeme příkaz:

  create view *v* **as** <výraz dotazu>

  kde:
  - <výraz dotazu> je jakýkoliv dovolený výraz
  - *v* reprezentuje jméno pohledu

Příklady dotazů

- Pohled skládající se z poboček a jejich zákazníků

```
create view všichni-zákazníci as
(select pobočka-jméno, zákazník-jméno
from vkladatel, účet
where vkladatel.číslo-účtu = účet.číslo-účtu)
union
(select pobočka-jméno, zákazník-jméno
from půjčovatel, účet
where půjčovatel.půjčka-číslo = půjčka.půjčka-číslo)
```

Modifikace databáze – mazání

- Smažte všechny záznamy o účtech z pobočky Praha-Spořilov

```
delete from účet
where pobočka-jméno = „Praha-Spořilov“
```

- Smažte všechny účty z každé pobočky v Brně

```
delete from účet
where pobočka-jméno in (
        select pobočka-jméno
        from pobočka
        where pobočka-město = „Brno“)
delete from vkladatel
where číslo-účtu in (
        select číslo-účtu
        from pobočka, účet
        where pobočka-město = „Brno“
        and pobočka.pobočka-jméno = účet.pobočka-jméno)
```

- Smažte záznamy o všech účtech se zůstatkem pod průměrem banky

```
delete from účet
where zůstatek < (
        select avg (zůstatek)
        from účet)
```

  - Problém: když smažeme n-tice z *vklad*, průměrný zůstatek se změní
  - Řešení použité v SQL:
    * Nejprve spočítáme průměrný zůstatek a najdeme všechny n-tice ke smazání
    * Potom smažeme všechny n-tice nalezené výše

Modifikace databáze – vkládání

- Vložení nové n-tice do relace *účet*

```
insert into účet
        values („Praha-Spořilov“, A-9732, $1200)
```

  nebo ekvivalentně

```
insert into účet (pobočka-jméno, zůstatek, číslo-účtu)
        values („Praha-Spořilov“, $1200, A-9732)
```

- Přidání nové n-tice do relace *účet* se *zůstatkem* nastaveným na nulu.

```
insert into účet
        values („Praha-Spořilov“, A-9732, null)
```

- Jako dárek poskytněte všem zákazníkům půjček v pobočce Praha-Spořilov $200 termínovaný účet. Nechť číslo půjčky slouží jako číslo účtu pro nový termínovaný účet.

```
insert into účet
    select pobočka-jméno, půjčka-číslo, 200
    from půjčka
    where pobočka-jméno = „Praha-Spořilov“
insert into vkladatel
    select zákazník-jméno, půjčka-číslo
    from půjčka, půjčovatel
    where pobočka-jméno = „Praha-Spořilov“
    and půjčka.číslo-účtu = půjčovatel.číslo-účtu
```

Modifikace databáze – aktualizace

- Všechny účty se zůstatkem přes $10,000 o 6 %, ostatní o 5 %.
  - Napíšeme dva výrazy **update**:

```
update účet
set zůstatek = zůstatek * 1.06
where zůstatek > 10000

update účet
set zůstatek = zůstatek * 1.05
where zůstatek ≤ 10000
```

  - pořadí je důležité !!! Co se stane v případě opačného pořadí?
  - lépe lze vyřešit pomocí výrazu **case**

Aktualizace pohledu

- Vytvoříme pohled na všechna data v relaci *půjčka*, zakryjte atribut *částka*

```
create view pobočka-půjčka as
    select pobočka-jméno, půjčka-číslo
    from půjčka
```

- Přidáme novou n-tici do pohledu

```
insert into pobočka-půjčka
    values („Praha-Spořilov“, „L-307“)
```

Toto vkládání musí být reprezentováno vložením n-tice („Praha-Spořilov“, „L-307“, *null*) do relace *půjčka*.

- Aktualizaci komplexnějších pohledů je náročné nebo nemožné přeložit a proto nejsou povoleny.

Spojené relace (Joined relations)

- Operace spojení vezme dvě relace a jako výsledek vrátí jednu relaci.
- Tyto operace jsou typicky používány jako poddotazy v klauzuli **from**.
- Podmínka spojení (join condition) – definuje, které n-tice ve dvou relacích si odpovídají a které atributy jsou ve výsledku spojení.
- Typ spojení (join type) – definuje kolik je n-tic v každé relaci, které neodpovídají žádné n-tici v jiné relaci.

Typy spojení
vnitřní spojení (inner join)
levé vnější spojení (left outer join)
pravé vnější spojení (right outer join)
plné vnější spojení (full outer join)

Podmínky spojení
přirozené (natural)
na <predikát>
on <predicate>
použitím (A_1, A_2, …, A_n)
using (A_1, A_2, …, A_n)

Příklady

- Relace *půjčka*

pobočka-jméno	*půjčka-číslo*	*částka*
Brno-Vinohrady	L-170	3000
Praha-Vršovice	L-230	4000
Praha-Spořilov	L-260	1700

- Relace *půjčovatel*

zákazník-jméno	*půjčka-číslo*
Janda	L-170
Starý	L-230
Horák	L-155

- *půjčka* **inner join** *půjčovatel* **on** *půjčka.půjčka-číslo* = *půjčovatel.půjčka-číslo*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*	*půjčka-číslo*
Brno-Vinohrady	L-170	3000	Janda	L-170
Praha-Vršovice	L-230	4000	Starý	L-230

- *půjčka* **left outer join** *půjčovatel* **on** *půjčka.půjčka-číslo* = *půjčovatel.půjčka-číslo*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*	*půjčka-číslo*
Brno-Vinohrady	L-170	3000	Janda	L-170
Praha-Vršovice	L-230	4000	Starý	L-230
Praha-Spořilov	L-260	1700	*null*	*null*

- *půjčka* **natural join** *půjčovatel*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*
Brno-Vinohrady	L-170	3000	Janda
Praha-Vršovice	L-230	4000	Starý

- *půjčka* **natural right outer join** *půjčovatel*

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*
Brno-Vinohrady	L-170	3000	Janda
Praha-Vršovice	L-230	4000	Starý
null	L-155	*null*	Horák

- *půjčka* **full outer join** *půjčovatel* **using** (*půjčka-číslo*)

pobočka-jméno	*půjčka-číslo*	*částka*	*zákazník-jméno*
Brno-Vinohrady	L-170	3000	Janda
Praha-Vršovice	L-230	4000	Starý
Praha-Spořilov	L-260	1700	*null*
null	L-155	*null*	Horák

- Najděte všechny zákazníky, kteří mají v bance buď účet nebo půjčku (ale ne obojí).

select *zákazník-jméno*
from (*vkladatel* **natural full outer join** *půjčovatel*)
where *číslo-účtu* **is** *null* **or** *půjčka-číslo* **is** *null*

Jazyk definice dat (Data definition language; DDL)

Umožňuje specifikaci nejen množin atributů, ale také informaci o každé relaci, zahrnující:

- Schéma každé relace.
- Doménu hodnot spojenou s každým atributem.
- Omezení integrity.
- Množinu indexů, které budou udržovány pro každou relaci.
- Bezpečnostní a autorizační informace o každé relaci.
- Fyzickou strukturu uložení na disk pro každou relaci.

Doménové typy v SQL

- **char(n)**. Řetězec znaků s pevnou délkou *n*.
- **varchar(n)**. Řetězec znaků s proměnlivou délkou (maximálně *n*).
- **int**. Celé číslo; závislé na implementaci.
- **smallint**. Krátké celé číslo; závislé na implementaci.
- **numeric(p,d)**. Číslo s pevnou desetinnou čárkou s přesností na *p* míst s *d* místy vpravo od desetinné tečky.
- **real, double precision**. Čísla s plovoucí desetinnou čárkou; závislé na implementaci.
- **float(n)**. Číslo s plovoucí desetinnou čárkou s přesností nejméně na *n* míst.
- **date**. Datumy obsahující (4bitový) rok, měsíc a den.
- **time**. Čas v hodinách, minutách a sekundách.
  - Ve všech doménových typech jsou povoleny nulové hodnoty. Deklarování atributu **not null** je zakazuje nulové hodnoty pro daný atribut.
  - V SQL-92 vytvoří konstrukce **create domain** uživatelské doménové typy

create domain *osoba-jméno* **char**(20) **not null**

Konstrukce create table

- SQL relace je definována použitím příkazu **create table**:

  create table *r* (A_1 D_1, A_2 D_2, …, A_n D_n,
  <omezení-integrity$_1$>,
  …,
  <omezení-integrity$_k$>)

  - *r* je jméno relace
  - každé A_i je jméno atributu ve schématu relace *r*
  - D_i je datový typ hodnot v doméně atributu A_i
- Příklad:

```
create table pobočka
        (pobočka-jméno      char(15) not null,
        pobočka-město       char(30),
        aktiva              integer)
```

Omezení integrity v konstrukci create table

- **not null**
- **primary key** (A_1, …, A_n)
- **check** (*P*), kde *P* je predikát

Příklad: Deklarace *pobočka-jméno* jako primárního klíče pro *pobočka* a zajištění, že hodnota atributu *aktiva* bude nezáporná.

```
create table pobočka
        (pobočka-jméno      char(15) not null,
        pobočka-město       char(30),
        aktiva              integer,
        primary key (pobočka-jméno),
        check (aktiva >= 0))
```

- Deklarace atributu jako **primary key** zajišťuje v SQL-92 automaticky i **not null**.

Konstrukce drop table a alter table

- Příkaz **drop table** smaže všechny informace o dané relaci z databáze.
- Příkaz **alter table** přidá atributy k existující relaci. Ke všem n-ticím v relaci přiřadí *null* jako hodnotu po nové atributy. Tvar příkazu je následující

  alter table *r* **add** *A D*

  kde *A* je jméno atributu, který je přidán do relace *r,* a *D* je jeho doména.
- Příkaz **alter table** je často používán též k odstranění (drop) atributů z relace:

  alter table *r* **drop** *A*

  kde *A* je jméno atributu relace *r*.

Zabudovaný SQL

- Standard SQL definuje zabudování SQL do řady programovacích jazyků jako jsou Pascal, PL/I, Fortran, C a Cobol.
- Jazyk, ve kterém jsou zabudovány SQL dotazy, je nazýván *hostitelský* jazyk a SQL struktury povolené v hostitelském jazyce vytvářejí *zabudovaný* SQL.
- Výraz EXEC SQL je používán pro identifikaci vloženého SQL dotazu pro preprocesor

EXEC SQL <vložený SQL výraz> END EXEC

Příklad dotazu

Z hostitelského jazyka najděte jména a čísla účtu s více než *částka* dolary na nějakém účtu.

- Specifikujte dotaz v SQL a deklarujete pro ni *kurzor*

```
EXEC SQL
    declare c cursor for
    select zákazník-jméno, číslo-účtu
    from vkladatel, účet
    where vkladatel.číslo-účtu = účet.číslo-účtu
        and účet.zůstatek > :částka
END-EXEC
```

Zabudovaný SQL (pokračování)

- Výraz **open** způsobí vyhodnocení dotazu

EXEC SQL **open** *c* END-EXEC

- Výraz **fetch** způsobí, že hodnoty jedné n-tice budou vloženy do proměnných hostitelského jazyka.

EXEC SQL **fetch** *c* **into** :*cn* :*an* END-EXEC

- Výraz **close** zavře databázový systém, takže se smaže dočasná relace, která obsahuje výsledek dotazu

EXEC SQL **close** *c* END-EXEC

Dynamický SQL

- Umožňuje programům konstruovat SQL dotazy za běhu.
- Příklad použití dynamického SQL z programu v jazyce C.

```
char *sqlprog = „update účet set zůstatek = zůstatek * 1.05
                where číslo-účtu = ?“
EXEC SQL prepare dynprog from :sqlprog;
char účet[10] = „A-101“;
EXEC SQL execute dynprog using :účet;
```

- Dynamický SQL program obsahuje znak ?, který je místem pro hodnotu, která je poskytována před spuštěním SQL programu.

Další rysy SQL

- *Jazyky čtvrté generace* –speciální jazyky asistující aplikačním programátorům při tvorbě uživatelského rozhraní a formátování výstupu
- SQL sezení (SQL session) – poskytuje abstrakci klienta a serveru
  - klient se *připojí* k SQL serveru, zahájí sezení
  - spustí sérii výrazů
  - *odpojí* sezení
  - může odevzdat nebo vrátit práci uskutečněnou v sezení
- SQL prostředí obsahuje několik komponent (identifikátor uživatele a schéma, které identifikuje, které z několika schémat sezení používá).