

# 6. Jazyky relačních databázových systémů

Ing. Vladimír Bartík, Ph.D.

RNDr. Marek Rychlý, Ph.D.



# Osnova

- 6.1. Tabulky ilustračního příkladu - Spořitelna
- 6.2. Jazyk SQL
  - 6.2.1. Úvod
  - 6.2.2. Definice dat
  - 6.2.3. Manipulace s daty
  - 6.2.4. Pohledy
  - 6.2.5. Přístup k systémovému katalogu (slovníku dat)
  - 6.2.6. Práce s chybějící informací
  - 6.2.7. Další příkazy SQL
- 6.3. Programování s SQL
  - 6.3.1. Hostitelská verze SQL (embedded SQL)
  - 6.3.2. Dynamický SQL
- 6.4. Další relační jazyky
  - 6.4.1. Jazyk QBE (Query By Example)
  - 6.4.2. Jazyk Datalog

## 6.1. Tabulky ilustračního příkladu - Spořitelna

- relační DB je vnímána uživatelem jako kolekce tabulek

Klient

r_cislo	jmeno	ulice	mesto
440726/0672	Jan Novák	Cejl 8	Brno
530610/4532	Petr Veselý	Podzimní 28	Brno
601001/2218	Ivan Zeman	Cejl 8	Brno
510230/048	Pavel Tomek	Tomkova 34	Brno
580807/9638	Josef Mádr	Svatoplukova 15	Brno
625622/6249	Jana Malá	Brněnská 56	Vyškov

Ucet

c_uctu	stav	r_cislo	pobocka
4320286	52000	440726/0672	Jánská
1182648	10853	530610/4532	Palackého
2075752	126350	440726/0672	Palackého

Pobocka

nazev	jmeni
Jánská	10000000
Palackého	5000000

## 6.1. Tabulky ilustračního příkladu - Spořitelna

Transakce

c_uctu	c_transakce	datum	castka
4320286	1	10.10.1998	3000
4320286	2	12.10.1998	- 5000
2075752	1	14.10.1998	- 2000
2075752	2	14.10.1998	10000

## 6.2.1. Jazyk SQL - Úvod

- Historie jazyka
  - 1975 - Sequel v System R
  - **1986** - standard ANSI, 1986 - standard ISO-SQL/86, dominantní úloha dialektu SQL firmy IBM (DB2)
  - 1989 - integritní dodatek (Integrity Addendum) - SQL/89,
  - **1992 - SQL/92**, tři úrovně souladu (Entry/Intermediate/Full)
  - 1996 - dodatek týkající se uložených modulů (PSM/96)
  - **1999** – SQL1999 – objektově-relační rysy
  - 2002 – podpora multimédií a dolování v datech
  - **2003** – SQL2003 – podpora OLAP, podpora XML (SQL/XML)
  - 2006 – rozšířená podpora XML v SQL/XML (XQuery)
  - 2008 – dokončení SQL/XML a dalších částí (~4000 s.)
  - 2011 – rozšíření o podporu temporálních rysů
  - 2016 – podpora JSON a některá další rozšíření

## 6.2.1. Jazyk SQL - Úvod

- Historie jazyka (pokračování)
  - SQL/MM – podpora Full Text, Spatial, Still image, Data mining
  - řada dialektů SQL, základem SQL/92 (minimálně úroveň Entry ) + vlastní rozšíření

Part	Explanation
Part 1 - SQL/Framework	Information common to all parts of the standard. (90 p.)
Part 2 - SQL/Foundation	Data definition and data manipulation syntax and semantics, SQL embedded in non-object programming languages, od 2003 i SQL/OLAP. (1366 p.)
Part 3 - SQL/CLI	(1995) Call Level Interface: Corresponds to ODBC. (405 p.)
Part 4 - SQL/PSM	(1996) Persistent Stored Modules: Stored routines, external routines, and procedural language extensions to SQL. (191 p.)
Part 9 - SQL/MED	Management of External Data: SQL access to non-SQL data sources (files). (486 p.)
Part 10 - SQL/OLB	(1999) Object Language Bindings: Specifies the syntax and semantics of embedding SQL in Java™ (SQLJ). (415 p.)
Part 11 - SQL/Schemata	(2003) Information and Definition Schemas. (298 p.)
Part 13 - SQL/JRT	(2003) Java Routines and Types: Routines using the Java™ Programming Language (Persistent Stored SQLJ). (208 p.)
Part 14 - SQL/XML	(2003) SQL and XML. (447 p.)

Převzato z <http://www.jcc.com/resources/sql-standards> (upraveno)

## 6.2.1. Jazyk SQL - Úvod

- **Tři možné kontexty použití jazyka SQL/92 (binding styles):**
  - **přímý (direct) SQL**
  - **hostitelská verze (embedded) SQL**
  - **jazyk modulů**
- **Hlavní kategorie příkazů**
  - **definice dat a pohledů (DDL – Data Definition Language)**
  - **manipulace s daty (DML – Data Manipulation Language)**
    - **pro přímý SQL**
    - **pro hostitelskou verzi**
  - **autorizace (řízení přístupových práv)**
  - **integrita dat**
  - **řízení transakcí**

## 6.2.1. Jazyk SQL - Úvod

- Malá/velká písmena
  - SQL nerozlišuje s výjimkou typu řetězec znaků (tj. CHAR, VARCHAR, NCHAR, NVARCHAR, NCLOB a odpovídajících variant v dialektech SQL)
- Kolize identifikátorů s rezervovanými identifikátory – viz skript *[kolize\\_jmen.sql](#)*
  - Lze použít rovné uvozovky (ne rovné apostrofy jako pro řetězcové literály), rozlišení malé/velké
  - Implicitně (bez uvozovek) se převádí na velká písmena



## 6.2.2. Definice dat

- Základní příkazy:
  - **CREATE** - vytvoření databázového objektu
  - **DROP** - zrušení databázového objektu
  - **ALTER** - změna vlastností databázového objektu
- Vytvoření báze (skutečně existující v databázi) tabulky

```
CREATE TABLE jm_bázové_tabulky  
    (def_sloupce, ...  
    [definice_integritních_omezení_tabulky]  
    )
```

- vytvoří novou, prázdnou tabulku + popis uloží do katalogu
- Definice sloupce

```
jméno_sloupce typ [impl_hodnota] [seznam_io_sloupce]
```

## 6.2.2. Definice dat

- Definice integritních omezení (io)

- Integritní omezení jsou omezení kladená na hodnoty ve sloupcích tabulky, aby nedošlo k porušení integrity dat.
- Zopakování pojmů důležitých pro integritní omezení tabulky (formální definice – viz kap. 3 Relační model dat):
  - **Kandidátní klíč** - sloupec, resp. sloupce tabulky (pro složený kandidátní klíč), jehož hodnota, resp. kombinace hodnot je v rámci tabulky unikátní (a dle RM ještě neredukovatelná).
  - **Primární klíč** – jeden z kandidátních klíčů, který bude sloužit k „adresaci“ řádků tabulky. Musí splňovat vlastnosti kandidátního klíče, navíc nesmí být hodnota prázdná.
  - **Alternativní klíč** – kandidátní klíč, který není primárním klíčem.
  - **Cizí klíč** – sloupec, resp. sloupce tabulky (pro složený cizí klíč), jehož hodnota, resp. kombinace hodnot se musí rovnat hodnotě kandidátního klíče v nějaké tabulce nebo může být plně nedefinovaná (prázdná). Slouží k vytváření vazeb mezi řádky tabulek. Soulad hodnot cizího klíče a odkazovaného kandidátního klíče se nazývá **referenční integrity**.

## 6.2.2. Definice dat

- Integritní omezení (deklarativní)

```
[CONSTRAINT jmeno] omezení
```

- Omezení pro sloupce

```
NULL, resp. NOT NULL
```

```
CHECK (podmíněný_výraz)
```

```
PRIMARY KEY
```

```
UNIQUE
```

```
REFERENCES tabulka [(jm_sloupce)]
```

```
[MATCH FULL|PARTIAL|SIMPLE] [událost ref_akce]
```

- Omezení celé tabulky

```
PRIMARY KEY (jm_sloupce, ...)
```

```
UNIQUE (jm_sloupce, ...)
```

```
FOREIGN KEY (jm_sloupce, ...) REFERENCES
```

```
tabulka [(jm_sloupce, ...)] [MATCH...] [událost ref_akce]
```

```
CHECK (podmíněný_výraz)
```

## 6.2.2. Definice dat

Př.) Definice dat

**Klient**

<u>r_cislo</u>	jmeno	ulice	mesto

**Pobocka**

<u>nazev</u>	jmeni

**Ucet**

<u>c_uctu</u>	stav	r_cislo	pobocka

**Transakce**

<u>c_transakce</u>	c_uctu	datum	<u>castka</u>

```
CREATE TABLE Ucet
(c_uctu    NUMERIC(7,0),
stav      NUMERIC(10,2) DEAFULT 0,
r_cislo   CHAR(11) CONSTRAINT ucet_rcislo_NN NOT NULL,
pobocka   CHAR(20) CONSTRAINT ucet_pob_NN NOT NULL,
CONSTRAINT PK_ucet PRIMARY KEY (c_uctu),
CONSTRAINT FK_ucet_rcislo FOREIGN KEY (r_cislo) REFERENCES Klient
ON DELETE CASCADE,
CONSTRAINT FK_ucet_pobocka FOREIGN KEY (pobocka) REFERENCES Pobocka)
```

## 6.2.2. Definice dat

- Datové typy

- *řetězcové:*

- CHARACTER(n), CHARACTER VARYING(n),
    - BIT(n), BIT VARYING(n) – vyřazeny v SQL:2003

- *numerické*

- přesné - NUMERIC(p, q), DECIMAL(p, q), INTEGER, SMALLINT
    - přibližné - FLOAT(p), REAL, DOUBLE PRECISION

- *datum a čas:* DATE, TIME, TIMESTAMP

- *intervalové:* INTERVAL

SQL/99 zavádí další předdefinované datové typy, např.:

- *řetězcové:*

- NATIONAL CHARACTER(n), NATIONAL CHARACTER VARYING(n), CHARACTER LARGE OBJECT, BINARY LARGE OBJECT

- *booleovský:*

- BOOLEAN

## 6.2.2. Definice dat

- Datové typy (pokračování)

*Pozn.: Pro řadu datových typů existují zkrácené názvy, např. CHAR, VARCHAR, NCHAR, NVARCHAR, CLOB, BLOB.*

*Dialekty SQL poskytují další zabudované typy, např. NUMBER u Oracle, TINYINT u MySQL.*

- Literály

- řetězcové: 'řetězec 1' pro znakové
- numerické: 12345.67 pro přesné, -25.7E-3 pro přibližné
- datum a čas: DATE '2005-02-27', TIME '10:00:27.5'

*Pozn.: Oracle ukládá v hodnotě typu DATE i čas, formát data lze nastavit inicializačním parametrem nebo lze použít konverzní funkci TO\_DATE a TO\_CHAR, např.*

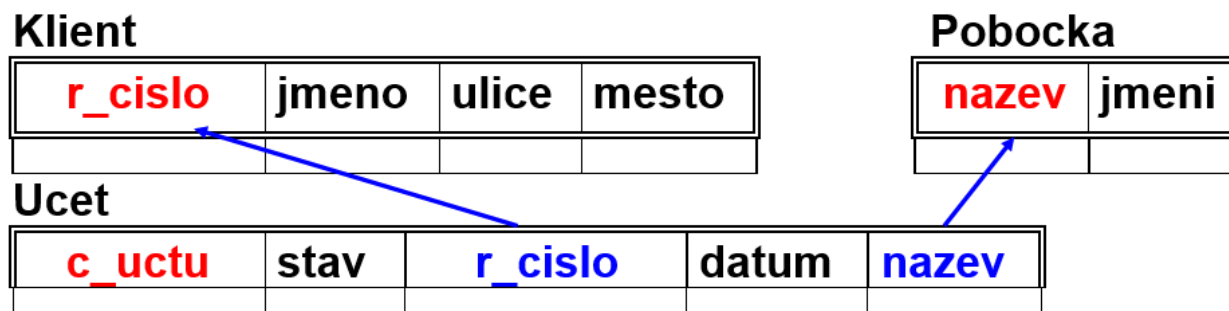
*TO\_DATE('98-DEC-25 17:30','YY-MON-DD HH24:MI')*

*TO\_CHAR(date\_expression, 'YY-MON-DD HH24:MI').*

## 6.2.2. Definice dat

- Operace s tabulkou, které mohou porušit referenční integritu a omezení primárního, alternativního klíče a NOT NULL (NN)

*Př.) Ilustrujeme na konkrétních tabulkách, ale diskutujeme obecně*



	SELECT	INSERT	DELETE	UPDATE
Klient	-	PK, AK, NN	<b>RI Ucet.r_cislo</b>	PK, AK, NN, <b>RI Ucet.r_cislo</b>
Ucet	-	PK, AK, NN, RI Klient, Pobocka	-	PK, AK, NN, RI Klient, Pobocka

- operace DELETE a UPDATE nad odkazovanou tabulkou mohou porušit referenční integritu
- Ize předepsat, jak se má SŘBD v této situaci zachovat (referenční akce)

## 6.2.2. Definice dat

- Referenční akce

- Referenční události: aktualizace (**ON UPDATE**), rušení (**ON DELETE**) řádků odkazované tabulky mající za následek porušení referenční integrity
- Referenční akce: co se má provést v případě referenční události s odkazujícími se řádky: **NO ACTION**, **CASCADE**, **SET DEFAULT**, **SET NULL**
  - Pozn.: Oracle: jen DELETE a akce NO ACTION, CASCADE, SET NULL

*Pozn.: Uvedení klauzule PRIMARY KEY, UNIQUE nebo FOREIGN KEY omezuje možné hodnoty v příslušném sloupci, resp. sloupcích, tj. říkáme, že daný sloupec je primárním, kandidátním, či cizím klíčem a požadujeme po SŘBD, aby kontroloval, že jsou odpovídající omezení dodržena.*



## 6.2.2. Definice dat

- Změna báze tabulky

**ALTER TABLE jm\_bázové\_tabulky akce**

- akce: přidání/zrušení (ADD/DROP [COLUMN] jm\_sl ...) sloupce; změna/zrušení (ALTER [COLUMN] jm\_sl SET/DROP DEFAULT) implicitní hodnoty; analogicky NOT NULL; přidání/zrušení (ADD/DROP CONSTRAINT jm) io pro tabulku
  - V některých případech (rušené mohlo být používáno) ještě [RESTRICT|CASCADE]
- modifikuje tabulku a změní informace v katalogu
- Použití ALTER TABLE k řešení referenčních cyklů – viz skript *referenční\_cyklus.sql*
  - Vzniká odkazy mezi tabulkami, které tvoří cyklus → nelze deklarovat i.o. FOREIGN KEY přímo v příkazu CREATE TABLE.  
Př) vedoucí oddělení: Oddělení (os\_c) → Zaměstnanec (os\_cislo),  
oddělení zaměstnance: Zaměstnanec (odd\_c) → Oddělení (odd\_c)).

## 6.2.2. Definice dat

### ▫ Použití ALTER TABLE k řešení referenčních cyklů (pokračování)

- Integritní omezení je potřeba přidat příkazem ALTER TABLE X ADD CONSTRAINT po vytvoření příslušných tabulek (viz SQL skript).
- Analogicky při rušení
  - Použít ALTER TABLE X DROP CONSTRAINT nebo
  - Povolit šíření rušení/zneplatnění (CASCADE, Oracle CASCADE CONSTRAINTS) u příkazu DROP TABLE
- U sebeodkazující se tabulky problém s referenčním cyklem není.

### • Zrušení báze tabulky

```
DROP TABLE jm_bázové_tabulky [RESTRICT|CASCADE]
```

- zruší tabulku a informace o ní v katalogu

### • Pohledy (VIEW) – viz později

- tabulky odvozené od jiných tabulek, nemusí v databázi fyzicky existovat

## 6.2.2. Definice dat

- Další typické databázové objekty (nejsou součástí SQL/92)

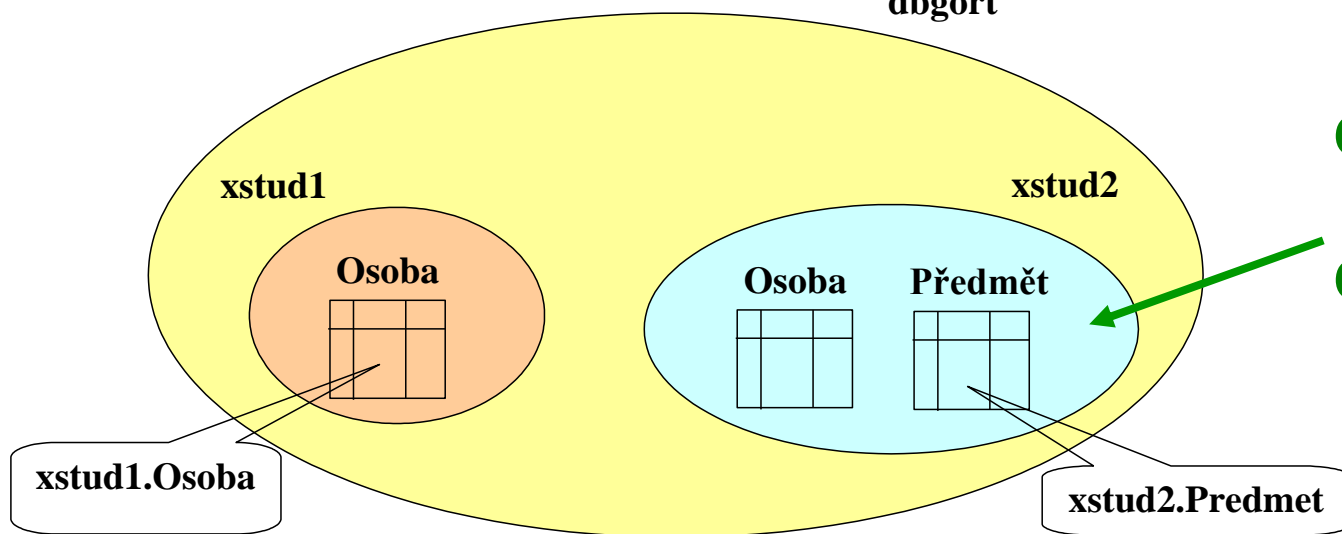
- Index

```
CREATE [UNIQUE] INDEX jm_indexu ON jm_bázové_tabulky  
(jm_sloupce [ASC|DESC], ... )
```

- Synonymum (nejen pro tabulky)

```
CREATE SYNONYM jm_synonyma FOR jm_tabulky
```

- umožňují zvýšit datovou nezávislost a nezávislost na umístění



```
CREATE SYNONYM Dalsi_osoba FOR  
xstud1.osoba;
```

```
CREATE SYNONYM Dalsi_osoba FOR  
xstud1.osoba@dbgort.fit.vutbr.cz;
```

## 6.2.2. Definice dat

- Další typické databázové objekty (nejsou součástí SQL/92)
  - Generátory posloupností čísel (sekvence, čítače) – standard z 2003

*Př) Oracle*

```
CREATE SEQUENCE osoby_seq  
START WITH 1000  
INCREMENT BY 1;
```

*MySQL*

```
CREATE TABLE Ucet (  
c_uctu UNSIGNED INT AUTO_INCREMENT, ... );
```

*Oracle 12c*

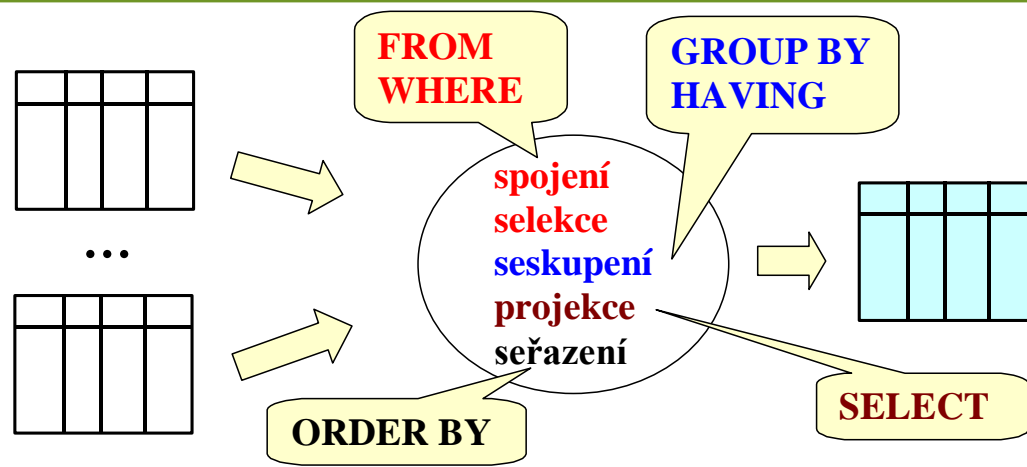
```
CREATE TABLE Osoba (  
osobaID INT DEFAULT osoby_seq.NEXTVAL PRIMARY KEY, ... );
```

```
CREATE TABLE Osoba (  
osobaID INT GENERATED AS IDENTITY PRIMARY KEY, ... );
```

## 6.2.3. Manipulace s daty

- příkazy: **SELECT**, **UPDATE**, **DELETE**, **INSERT**
  - operandem jsou báze tabulky nebo pohledy, výsledkem tabulka
- Příkaz **SELECT**

```
SELECT [ALL|DISTINCT] položka [[AS] alias_sl], ...  
FROM tabulkový_výraz [[AS] [alias_tab]], ...  
[WHERE podmínka]  
[GROUP BY jm_sloupce_z_FROM|číslo, ...]  
[HAVING podmínka]  
[ORDER BY jm_sloupce_z_SELECT|číslo [ASC|DESC], ...]
```



## 6.2.3. Manipulace s daty

- Jednoduché dotazy (nad jednou tabulkou)

Př.) „Kdo jsou klienti spořitelny?“

```
SELECT r_cislo, jmeno  
FROM Klient
```

- V klauzuli SELECT lze použít zástupný symbol „\*“ ve významu všechny sloupce

```
SELECT *  
FROM Klient
```

Klient

r_cislo	jmeno	ulice	mesto

Pobočka

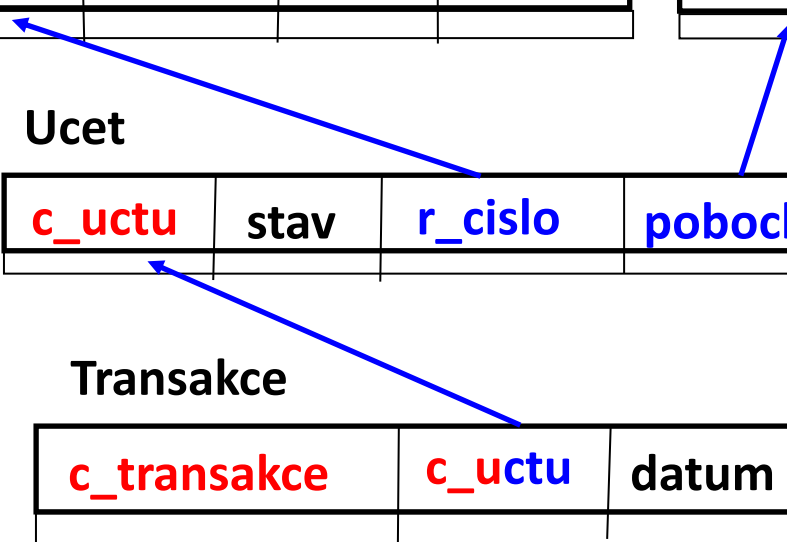
nazev	jmeni

Ucet

c_uctu	stav	r_cislo	pobocka

Transakce

c_transakce	c_uctu	datum	castka



## 6.2.3. Manipulace s daty

- Jednoduché dotazy (nad jednou tabulkou)

- Uvedením klíčového slova **DISTINCT** se eliminují duplicitní řádky

Př.) „Ze kterých měst jsou klienti spořitelny?“

```
SELECT DISTINCT mesto  
FROM Klient
```

- Klauzule **WHERE** určuje podmínku pro výběr řádků

Př.) „Které účty jsou u pobočky Jánská?“

```
SELECT c_uctu  
FROM Ucet  
WHERE pobočka = 'Jánská'
```

Klient

<b>r_cislo</b>	jmeno	ulice	mesto

Pobočka

<b>nazev</b>	jmeni

Ucet

<b>c_uctu</b>	stav	<b>r_cislo</b>	<b>pobočka</b>

Transakce

<b>c_transakce</b>	<b>c_uctu</b>	datum	castka

## 6.2.3. Manipulace s daty

- Přejmenování

- Lze zavádět nová jména (alias) pro sloupce výsledné tabulky (v klauzuli SELECT)
- Lze zavádět nová jména tabulek, resp. tabulkových výrazů (v klauzuli FROM)

výraz [AS] alias

- **Nová jména sloupců z klauzule SELECT lze používat pouze v klauzuli ORDER BY**, nová jména tabulek ve všech klauzulích příkazu SELECT



## 6.2.3. Manipulace s daty

- Dotazy s využitím přejmenování

- V klauzuli SELECT mohou být i výrazy

Př.) „Kolik činí jmění poboček v USD při kurzu 25 Kč/\$?“

```
SELECT nazev, jmeni/25 jmeni_v_$
FROM Pobocka
```

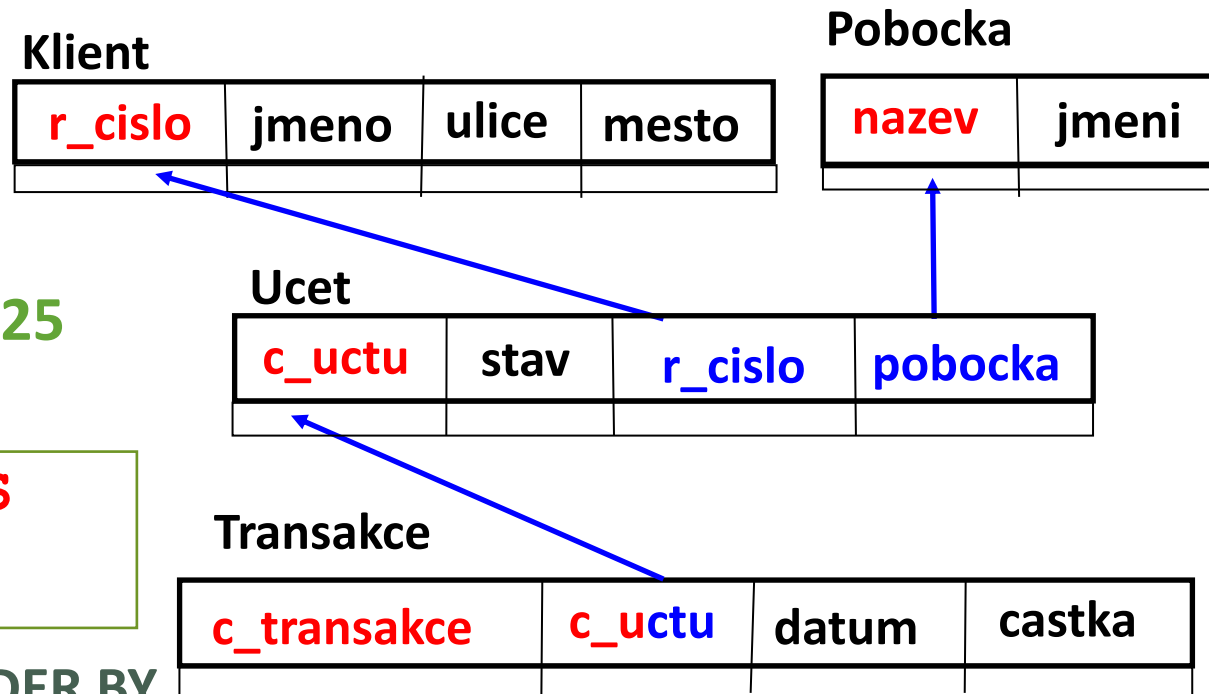
- Výslednou tabulku lze uspořádat - klauzule ORDER BY

```
SELECT nazev, jmeni/25 jmeni_v_USD
FROM Pobocka
```

```
ORDER BY jmeni_v_$
```

nebo

```
ORDER BY 2
```



## 6.2.3. Manipulace s daty

- Spojení informací z více tabulek  
(**operace JOIN**)

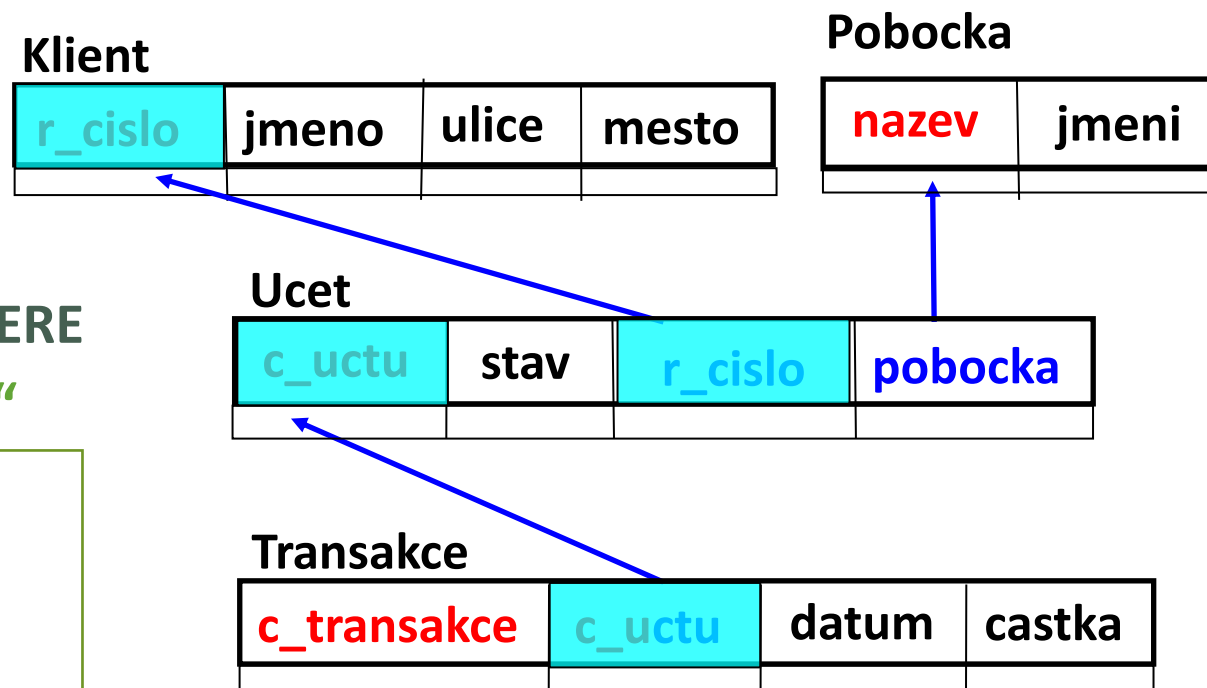
- Zápis spojení pomocí podmínky v klauzuli WHERE

Př.) „Kteří klienti mají účet v pobočce Jánská?“

```
SELECT DISTINCT K.*
FROM Klient K, Ucet U
WHERE K.r_cislo=U.r_cislo
      AND U.pobocka='Jánská'
```

Př.) „Kteří klienti prováděli transakce v pobočce Jánská 12.10.1998?“

```
SELECT K.r_cislo, K.jmeno, T.c_uctu, T.castka
FROM Klient K, Ucet U, Transakce T
WHERE K.r_cislo=U.r_cislo AND U.c_uctu=T.c_uctu
      AND U.pobocka='Jánská' AND T.datum='1998-10-12'
```



## 6.2.3. Manipulace s daty

### • Typy spojení:

#### ▣ vnitřní (inner)

- obecné na základě podmínky ( $\Theta$ join)
- na základě rovnosti (equijoin)
- přirozené (natural join)

#### ▣ vnější (outer)

T1			T2		
A	B	X	X	C	D
0	a	x	x	1	0
1	a	x	x	2	0
3	c	z	y	3	1

Př.) T1 JOIN T2 ON A<C

A	B	X <sub>T1</sub>	X <sub>T2</sub>	C	D
0	a	x	x	1	0
0	a	x	x	2	0
0	a	x	y	3	1
1	a	x	x	2	0
1	a	x	y	3	1

## 6.2.3. Manipulace s daty

- Typy spojení:

- **vnitřní (inner)**

- obecné na základě podmínky ( $\Theta$ join)
- na základě rovnosti (equijoin)
- přirozené (natural join)

- **vnější (outer)**

T1			T2		
A	B	X	X	C	D
0	a	x	x	1	0
1	a	x	x	2	0
3	c	z	y	3	1

**Př.) T1 JOIN T2 ON A=D**

A	B	X <sub>T1</sub>	X <sub>T2</sub>	C	D
0	a	x	x	1	0
0	a	x	x	2	0
1	a	x	y	3	1

## 6.2.3. Manipulace s daty

### • Typy spojení:

#### ▣ **vnitřní (inner)**

- obecné na základě podmínky( $\Theta$ join)
- na základě rovnosti (equijoin)
- **přírozené (natural join)**

#### ▣ **vnější (outer)**

T1

A	B	X
0	a	x
1	a	x
3	c	z

T2

X	C	D
x	1	0
x	2	0
y	3	1

### Př.) T1 NATURAL JOIN T2

A	B	X	C	D
0	a	x	1	0
0	a	x	2	0
1	a	x	1	0
1	a	x	2	0

## 6.2.3. Manipulace s daty

### • Typy spojení:

- vnitřní (inner)
  - obecné na základě podmínky( $\Theta$ join)
  - na základě rovnosti (equijoin)
  - přirozené (natural join)
- vnější (outer)

**T1**

A	B	X
0	a	x
1	a	x
3	c	z

**T2**

X	C	D
x	1	0
x	2	0
y	3	1

**Př.) Přirozené levé vnější spojení**  
**T1 NATURAL LEFT JOIN T2**

A	B	X	C	D
0	a	x	1	0
0	a	x	2	0
1	a	x	1	0
1	a	x	2	0
3	c	z		

## 6.2.3. Manipulace s daty

- V klauzuli FROM lze uvádět i tabulkové výrazy, resp. **výraz spojení** (join expression) tvaru:

```
tabulka CROSS JOIN tabulka |  
tabulka [NATURAL] [typ_spojení] JOIN tabulka  
    [ON podmínka | USING (sloupec , ...)]
```

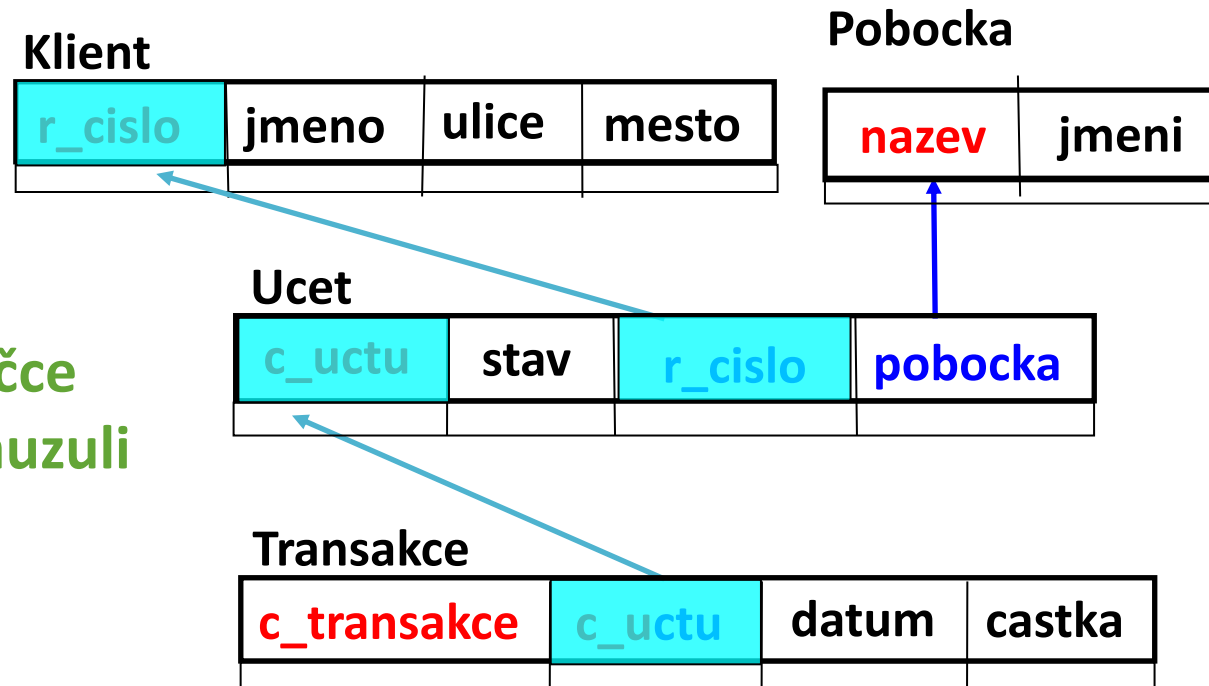
- Typy spojení: INNER | (LEFT|RIGHT|FULL)[OUTER] | UNION

Př.) „Kteří klienti mají účet v pobočce Jánská?“ (s výrazem spojení v klauzuli FROM)

```
SELECT DISTINCT r_cislo, jmeno, ulice, mesto  
FROM Klient NATURAL JOIN Ucet  
WHERE pobočka='Jánská'
```

## 6.2.3. Manipulace s daty

Př.) „Kteří klienti prováděli transakce v pobočce Jánská 12.10.1998?“ (s výrazem spojení v klauzuli FROM)



```
SELECT r_cislo, jmeno, c_uctu, castka
FROM Klient NATURAL JOIN Ucet NATURAL JOIN Transakce
WHERE pobočka='Jánská' AND datum='1998-10-12'
ORDER BY r_cislo
```



## 6.2.3. Manipulace s daty

- Lze spojit i dvě stejné tabulky

Klient

r_cislo	jmeno	ulice	mesto

Klient

r_cislo	jmeno	ulice	mesto

Př.) „Bydlí někteří klienti na stejné adrese?”

```
SELECT K1.jmeno, K1.r_cislo, K2.jmeno, K2.r_cislo,  
       K1.ulice, K1.mesto  
FROM Klient K1, Klient K2  
WHERE K1.mesto=K2.mesto AND K1.ulice=K2.ulice  
      AND K1.r_cislo>K2.r_cislo
```

```
SELECT K1.jmeno, K1.r_cislo, K2.jmeno, K2.r_cislo,  
       ulice, mesto  
FROM Klient K1 JOIN Klient K2 USING (mesto, ulice)  
WHERE K1.r_cislo>K2.r_cislo
```

## 6.2.3. Manipulace s daty

- **Agregační funkce**

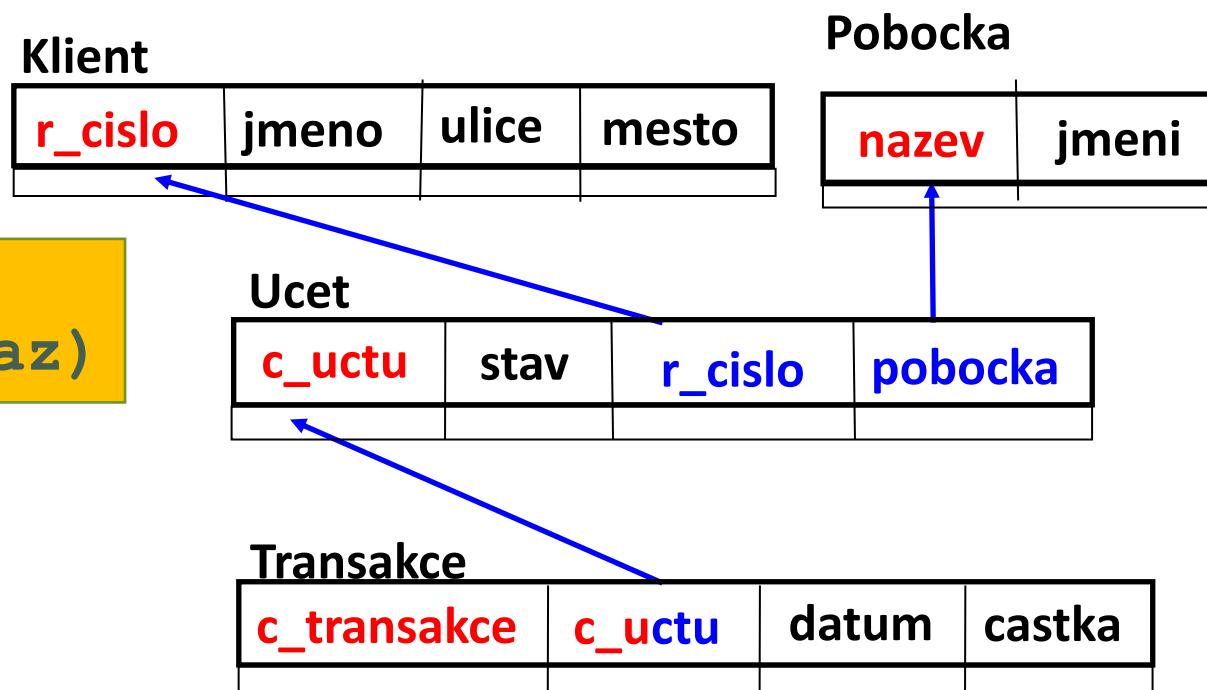
COUNT (\*) | AVG | MAX | MIN | SUM |  
COUNT ([ALL | DISTINCT] skal\_výraz)

Př.) „Kolik klientů má spořitelna?“

```
SELECT COUNT(*) pocet
FROM Klient
```

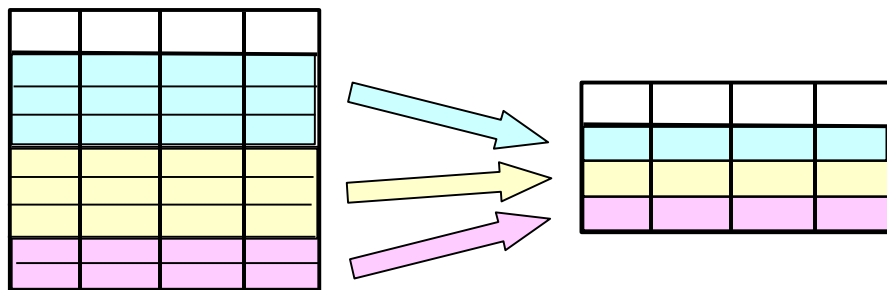
- **agregační funkce nelze zanořovat**

*Oracle: lze, např. SELECT AVG(MAX(plat)) FROM ... GROUP BY*



## 6.2.3. Manipulace s daty

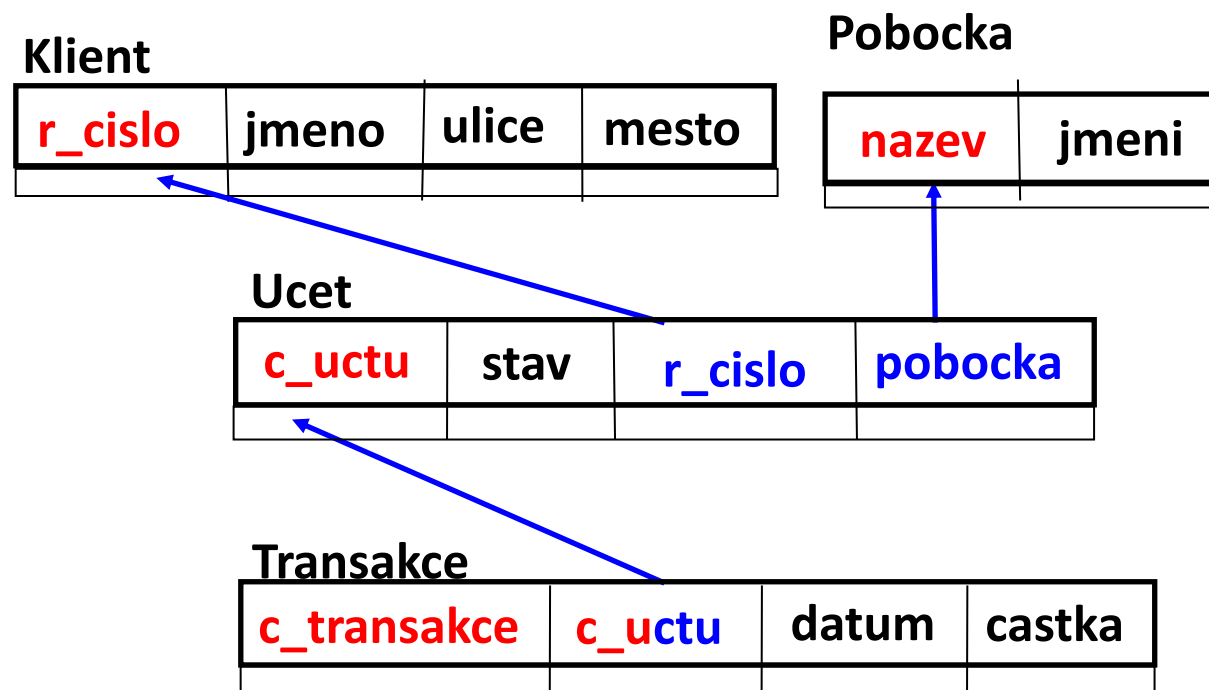
### • Klausule GROUP BY



Př.) „Jaká částka je na účtech v jednotlivých pobočkách?“

```
SELECT pobočka, SUM(stav) celkem_na_uctech
FROM Ucet
GROUP BY pobočka
```

- Omezení pro výrazy v klauzuli SELECT: agregační funkce, prvky seznamu v GROUP BY, konstanty



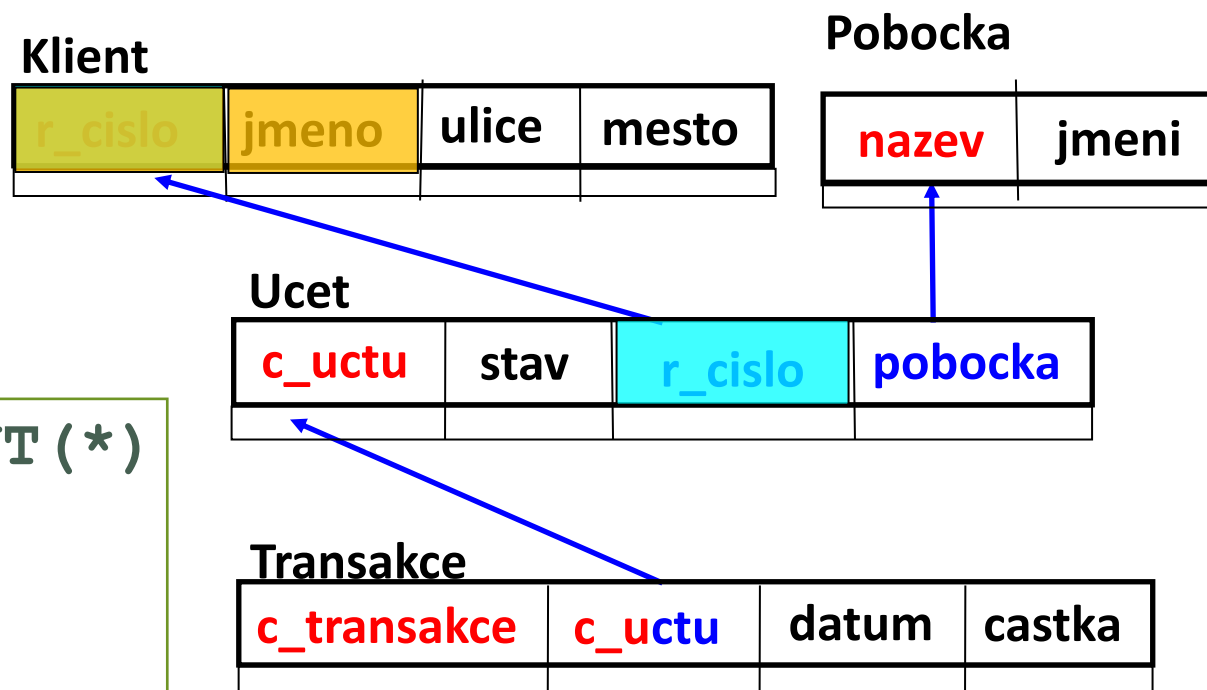
## 6.2.3. Manipulace s daty

- Klauzule GROUP BY (pokračování)

Př.) „Kolik mají účtů a celkem na nich peněz jednotliví klienti?“

```
SELECT K.jmeno, K.r_cislo, COUNT(*)
      pocet, SUM(stav) celkem
FROM Klient K, Ucet U
WHERE K.r_cislo=U.r_cislo
GROUP BY K.r_cislo, K.jmeno
```

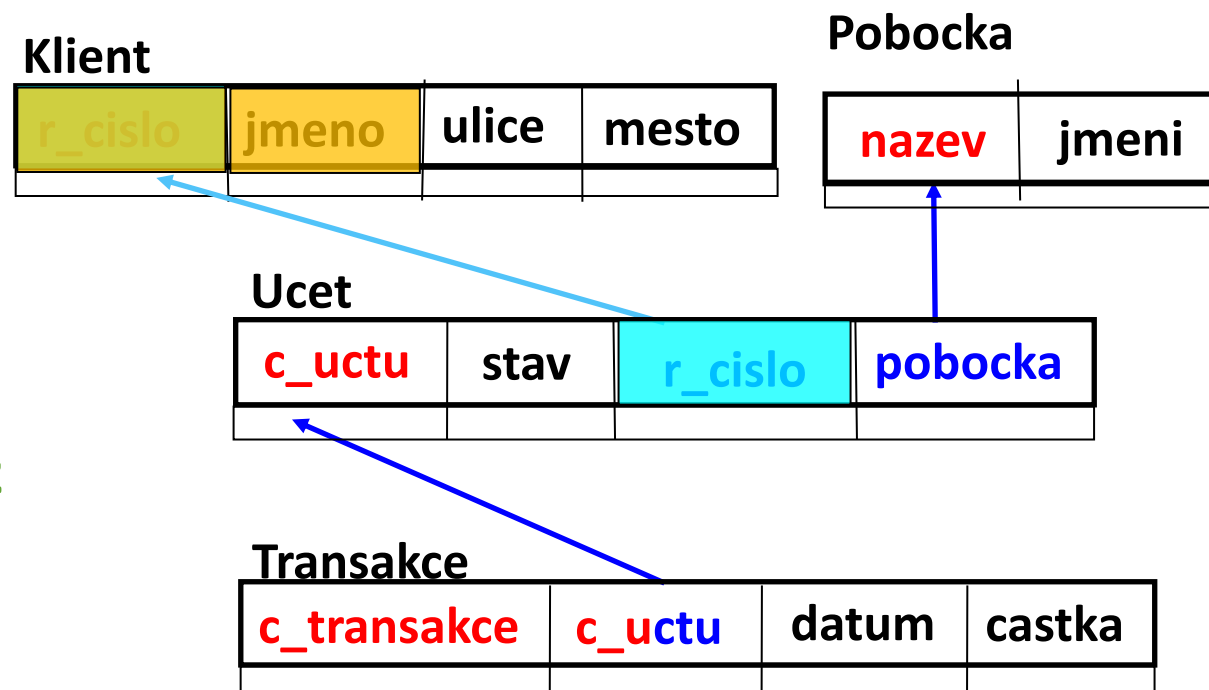
```
SELECT jmeno, r_cislo, COUNT(*) pocet, SUM(stav) celkem
FROM Klient NATURAL JOIN Ucet
GROUP BY r_cislo, jmeno
```



## 6.2.3. Manipulace s daty

- Klauzule GROUP BY (pokračování)

Př.) „Kolik mají účtů a celkem na nich peněz jednotliví klienti?“ (s využitím vnějšího spojení – zobrazí i klienty, kteří nevlastní žádný účet)



```
SELECT jmeno, r_cislo, COUNT(c_uctu) pocet, SUM(stav) celkem
FROM Klient NATURAL LEFT JOIN Ucet
GROUP BY r_cislo, jmeno
ORDER BY 4 DESC;
```

## 6.2.3. Manipulace s daty

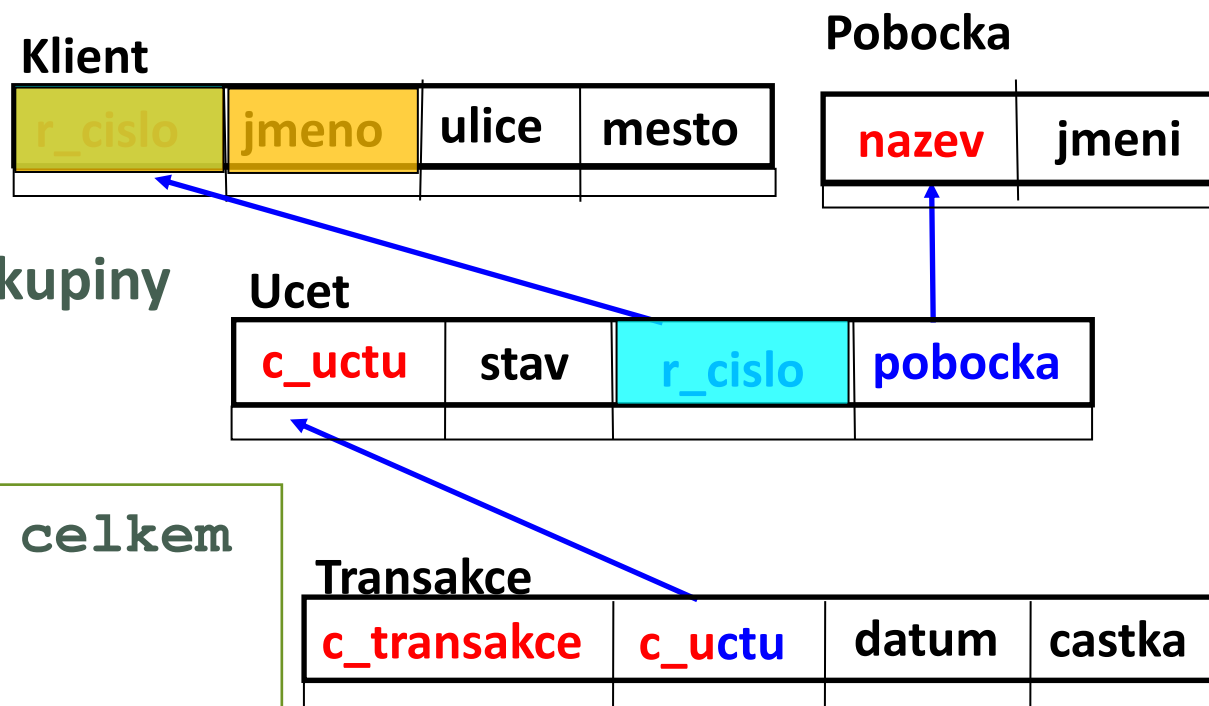
### • Klauzule HAVING

- analogie WHERE, ale aplikované na celé skupiny

Př.) „Kteří klienti mají na účtech více než 100000Kč?“

```
SELECT K.jmeno, K.r_cislo, SUM(stav) celkem
FROM Klient K, Ucet U
WHERE K.r_cislo=U.r_cislo
GROUP BY K.r_cislo, K.jmeno
HAVING SUM(stav)>100000
```

```
SELECT jmeno, r_cislo, SUM(stav) celkem
FROM Klient NATURAL JOIN Ucet
GROUP BY r_cislo, jmeno
HAVING SUM(stav)>100000
```



## 6.2.3. Manipulace s daty

- **Klauzule WHERE (podmíněný výraz)**

- může obsahovat tyto predikáty (případně s operátorem NOT a spojené logickými spojkami AND, OR):
- Porovnání

```
konstruktor_řádku rel_op konstruktor_řádku |  
konstruktor_řádku rel_op {ANY|SOME|ALL} (tabulkový výraz)
```

a	b	c
---	---	---

=

<>

>=

...

ANY/SOME

FALSE

A	B	C
X	X	X
X	X	X
X	X	X
X	X	X

## 6.2.3. Manipulace s daty

### ▣ Porovnání (pokračování)

a	b	c
---	---	---

=

<>

>=

...

ANY/SOME

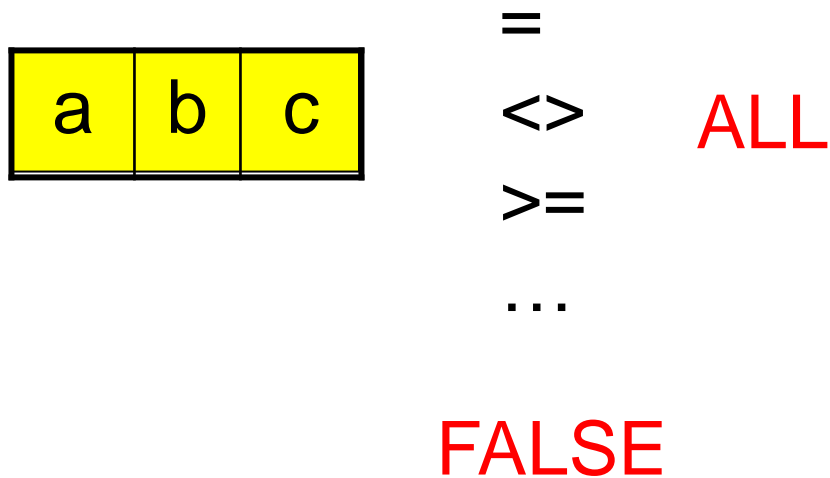
TRUE

A	B	C
x	x	x
a	b	c
x	x	x
x	x	x



## 6.2.3. Manipulace s daty

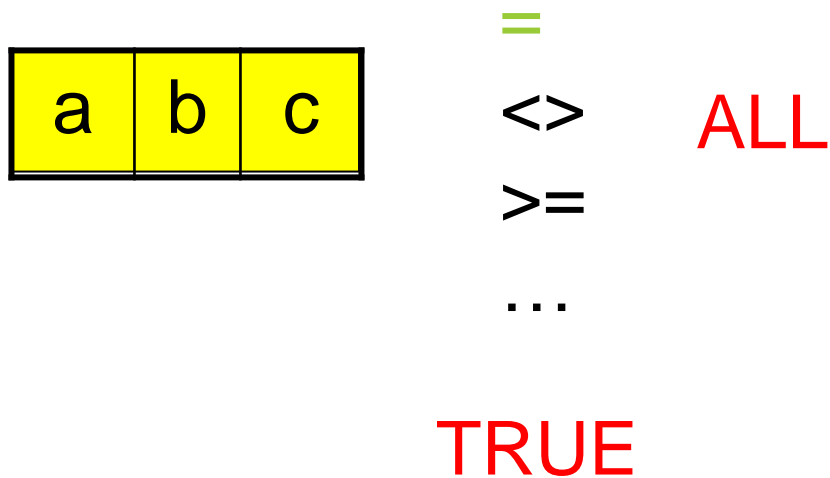
### ▣ Porovnání (pokračování)



A	B	C
a	b	c
x	x	x
a	b	c
a	b	c

## 6.2.3. Manipulace s daty

### ▣ Porovnání (pokračování)



A	B	C
a	b	c
a	b	c
a	b	c
a	b	c

## 6.2.3. Manipulace s daty

- Porovnání (pokračování)

Př.) „Kteří mimobrněňští klienti mají uloženo více než brněňští?“

```
SELECT K.jmeno, K.r_cislo, SUM(stav) celkem
FROM Klient K, Ucet U
WHERE K.r_cislo=U.r_cislo AND K.mesto<>'Brno'
GROUP BY K.jmeno, K.r_cislo
HAVING SUM(stav) > ALL
  (SELECT SUM(stav)
   FROM Klient K, Ucet U
   WHERE K.r_cislo=U.r_cislo AND K.mesto='Brno'
   GROUP BY K.r_cislo)
```

## 6.2.3. Manipulace s daty

- Porovnání (pokračování)

Př.) „Kteří mimobrněňští klienti mají uloženo více než brněňští?“ (varianta s NATURAL JOIN)

```
SELECT jmeno, r_cislo, SUM(stav) celkem
FROM Klient NATURAL JOIN Ucet
WHERE mesto<>'Brno'
GROUP BY r_cislo, jmeno
HAVING SUM(stav) > ALL
  (SELECT SUM(stav)
   FROM Klient NATURAL JOIN Ucet
   WHERE mesto='Brno'
   GROUP BY r_cislo)
```

## 6.2.3. Manipulace s daty

- Test na chybějící informaci

```
jmeno_sloupce IS [NOT] NULL
```

Př.) „Má některý klient neúplně zadanou adresu?“

```
SELECT *  
FROM Klient WHERE ulice IS NULL OR mesto IS NULL
```

- Predikát BETWEEN

```
výraz [NOT] BETWEEN výraz AND výraz
```

- $e \text{ BETWEEN } c1 \text{ AND } c2$  je ekvivalentní:  $e \geq c1 \text{ AND } e \leq c2$

Př.) „Kteří klienti prováděli transakce na svých účtech v měsíci říjnu 1998?“

```
SELECT K.jmeno, K.r_cislo, U.c_uctu, T.datum, T.castka  
FROM Klient K, Ucet U, Transakce T  
WHERE K.r_cislo=U.r_cislo AND U.c_uctu=T.c_uctu  
AND T.datum BETWEEN '1998-10-01' AND '1998-10-31'
```

## 6.2.3. Manipulace s daty

### ▫ Predikát EXISTS

[NOT] EXISTS (tabulkový\_výraz)

- Test na neprázdnot tabulky, která je výsledkem poddotazu

EXISTS

A	B	C

FALSE

## 6.2.3. Manipulace s daty

- Predikát EXISTS (pokračování)

EXISTS

TRUE

A	B	C
x	x	x

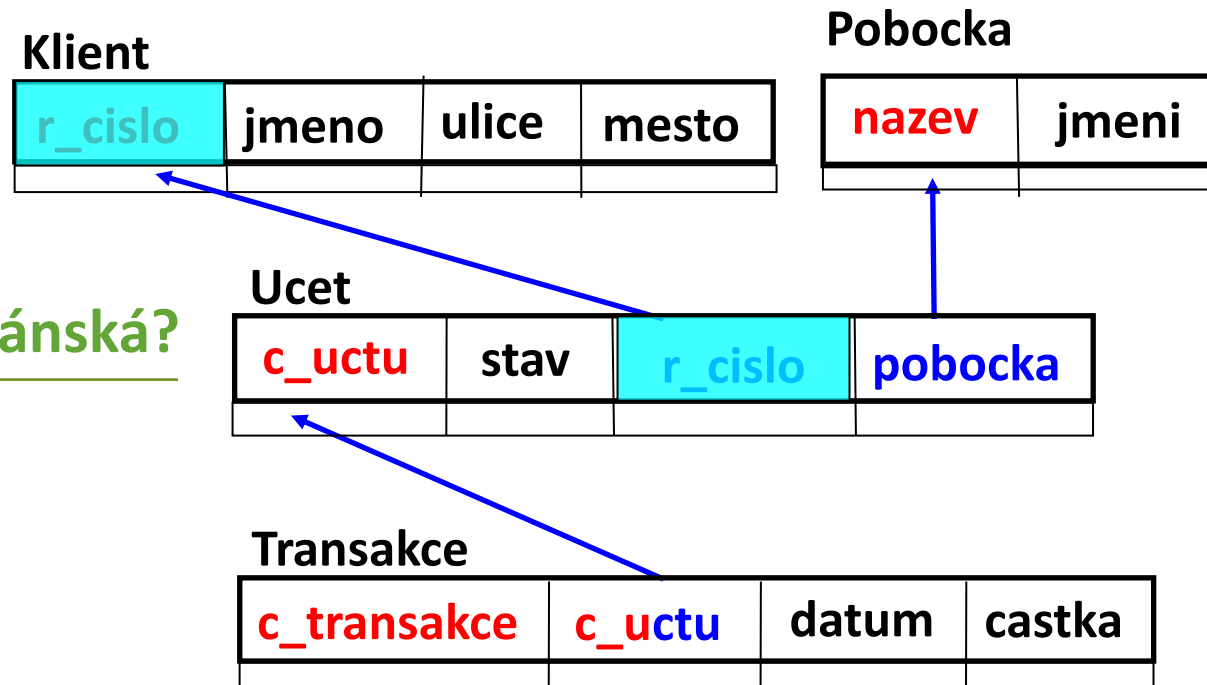
## 6.2.3. Manipulace s daty

### ▣ Predikát EXISTS

- Typický poddotaz s „\*“ v klauzuli SELECT

Př.) „Kteří klienti mají účet jen u pobočky Jánská?”

```
SELECT DISTINCT K.*
FROM Klient K, Ucet U
WHERE K.r_cislo=U.r_cislo
AND U.pobocka='Jánská'
AND NOT EXISTS (SELECT *
                  FROM Ucet U
                  WHERE K.r_cislo=U.r_cislo AND
                        U.pobocka<>'Jánská')
```



*tzv. korelovaný  
poddotaz*



## 6.2.3. Manipulace s daty

- Predikát UNIQUE (Oracle NE)
  - Test na neexistenci duplicitních řádků v tabulce poddotazu

[NOT] UNIQUE (tabulkový\_výraz)

Př.) „Kteří klienti mají u pobočky Jánská jen jeden účet?“

```
SELECT K.*  
FROM Klient K  
WHERE UNIQUE  
  (SELECT c_uctu  
   FROM Ucet U  
   WHERE K.r_cislo=U.r_cislo AND  
        U.pobocka='Jánská')
```

## 6.2.3. Manipulace s daty

Př.) „Kteří klienti mají u pobočky Jánská jen jeden účet?“ (alternativní řešení)

```
SELECT K.*  
FROM Klient K  
WHERE (SELECT COUNT(*)  
       FROM Ucet U  
       WHERE K.r_cislo=U.r_cislo AND  
             U.pobocka='Jánská') = 1
```

```
SELECT K.*  
FROM Klient K, Ucet U  
WHERE K.r_cislo=U.r_cislo AND U.pobocka='Jánská '  
AND NOT EXISTS  
  (SELECT *  
   FROM Ucet U1  
   WHERE K.r_cislo=U1.r_cislo AND  
         U1.pobocka='Jánská' AND U.c_uctu <> U1.c_uctu)
```

## 6.2.3. Manipulace s daty

### ▫ Predikát LIKE

**výraz\_řetězec [NOT] LIKE vzor [ESCAPE esc\_znak]**

- Vzor je řetězcový výraz, může obsahovat **zástupné znaky**:
  - **\_** - libovolný znak,
  - **%** - libovolný počet libovolných znaků (i žádný)
- *esc\_znak* je znak rušící ve vzoru význam zástupného znaku

Př.) řetězec LIKE **'\\_%' ESCAPE ,\'**

Př.) „Kteří klienti mají křestní jméno Jan?“

```
SELECT *  
FROM Klient  
WHERE jmeno LIKE 'Jan %'
```

## 6.2.3. Manipulace s daty

### ▫ Predikát IN

konstruktor\_řádku [NOT] IN (tabulkový\_výraz) |  
konstruktor\_řádku [NOT] IN (seznam\_konstruktorů\_řádku)

Př.) „Kteří klienti jsou z Brna nebo Prahy?“

```
SELECT *  
FROM Klient  
WHERE mesto IN ('Praha', 'Brno')
```

a	b	c
---	---	---

IN

≡

= ANY

FALSE

A	B	C
x	x	x
x	x	x
x	x	x

## 6.2.3. Manipulace s daty

### ▣ Predikát IN (pokračování)

a	b	c
---	---	---

TRUE

IN

≡

= ANY

A	B	C
x	x	x
a	b	c
x	x	x

## 6.2.3. Manipulace s daty

- Predikát IN (pokračování)

Př.) „Kteří klienti prováděli transakce v říjnu 1998?“

```
SELECT *  
FROM Klient  
WHERE r_cislo IN  
      (SELECT r_cislo FROM Ucet  
        WHERE c_uctu IN  
              (SELECT c_uctu FROM Transakce  
                WHERE datum BETWEEN '1998-10-01' AND '1998-10-31'))
```

- Predikát MATCH (Oracle NE)
  - Obdoba IN s možností testu na shodu s právě jediným řádkem

## 6.2.3. Manipulace s daty

- Operátory pro sjednocení, rozdíl a průnik tabulek

```
tabulkový_výraz UNION|EXCEPT|INTERSECT [ALL]  
                tabulkový_výraz [klauzule_ORDER_BY]
```

Př.) Předpokládejme další tabulku Půjčka

c_pujcky	r_cislo	pobocka	castka	splaceno
----------	---------	---------	--------	----------

„Kteří klienti mají u pobočky Jánská účet nebo půjčku?“

```
SELECT K.jmeno, K.r_cislo  
FROM Klient K, Ucet U  
WHERE K.r_cislo=U.r_cislo AND U.pobocka='Jánská'  
UNION  
SELECT K.jmeno, K.r_cislo  
FROM Klient K, Pujcka P  
WHERE K.r_cislo=P.r_cislo AND P.pobocka='Jánská'
```

- při provedení příkazu se automaticky odstraňují duplicity

## 6.2.3. Manipulace s daty

- Některé užitečné výrazy používané v klauzuli SELECT

- Operátor CASE

```
CASE WHEN podmínka THEN skalární_výraz  
      ...  
      ELSE skalární_výraz  
END
```

- funkce COALESCE (x, y) – náhrada prázdné hodnoty

- je ekvivalentní výrazu:

```
CASE WHEN x IS NOT NULL THEN x  
      ELSE y
```

```
END
```

- obecně (x, y, ...) - je-li x NULL, pak první neprázdná hodnota



## 6.2.3. Manipulace s daty

- Některé užitečné výrazy používané v klauzuli SELECT (pokračování)

- funkce NULLIF (x, y) – náhrada prázdnou hodnotou

- je ekvivalentní výrazu:

```
CASE WHEN x = y THEN NULL  
      ELSE x
```

```
END
```

- Řada dalších funkcí (u ORACLE např. SYSDATE, ...)

- SQL WITH klauzule (od SQL-99)

- Specifikuje tabulku, kterou lze následně použít v dotazu

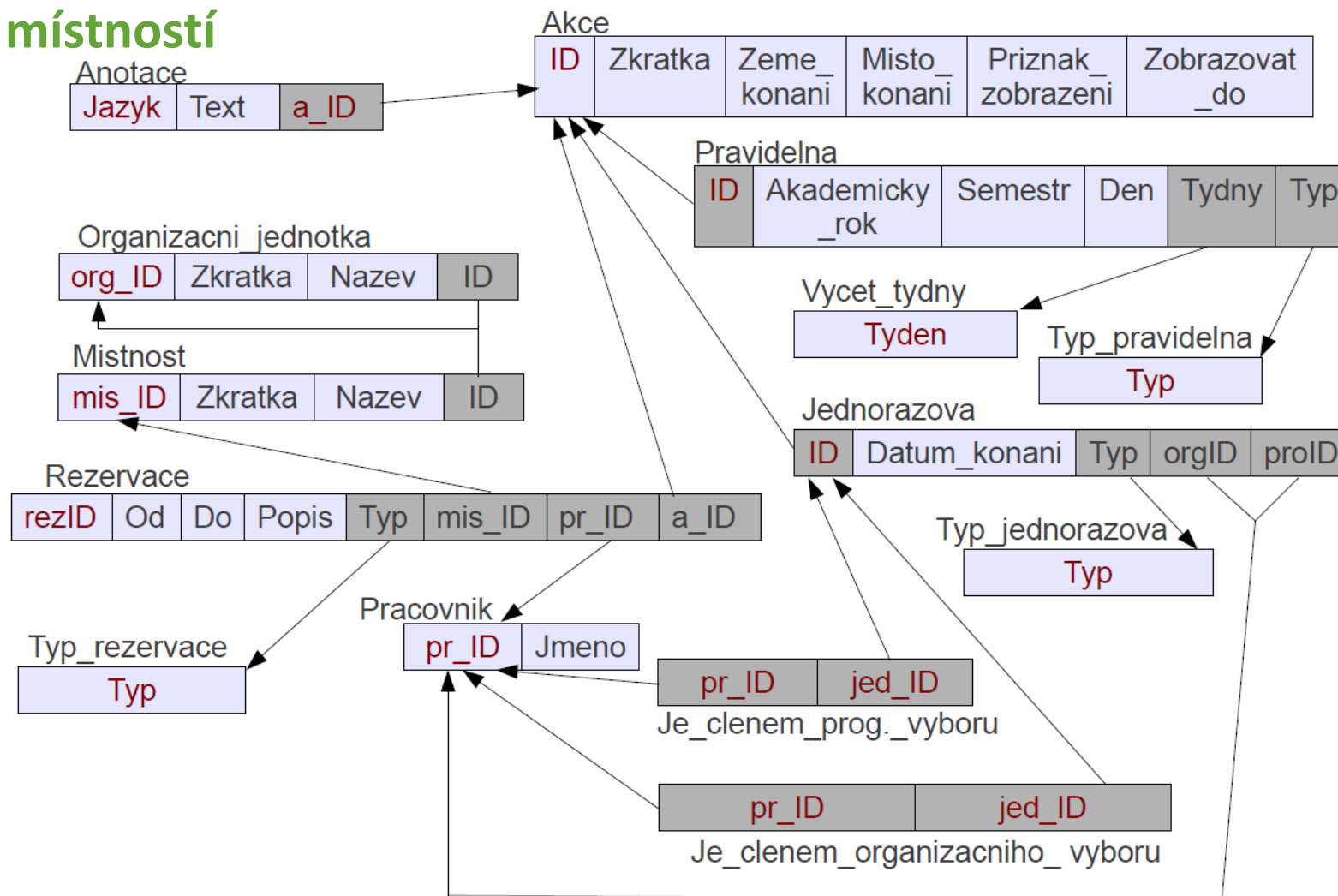
```
WITH jméno_tab AS (dotaz) , ...  
    příkaz_SELECT
```

- zjednodušuje složité dotazy
- tabulku lze použít v příkazu SELECT, který je klauzulí WITH uvozen

## 6.2.3. Manipulace s daty

- SQL WITH klauzule (pokračování)

### Př) Rezervace místností



## 6.2.3. Manipulace s daty

- SQL WITH klauzule (pokračování)

Př.) „Kolik hodin je v jednotlivých dnech týdne v tomto semestru vytížena místnost E112 pravidelnou výukou?“ Očekávaný výsledek je tabulka (den, hodin).

```
WITH day_totals AS (  
  SELECT p.den,  
    CEIL (SUM(extract (hour from r.do) + extract(minute  
      from r.do)/60.0 - extract (hour from r.od) -  
      extract(minute from r.od)/60.0)) vytizeni  
  FROM rezervace r, mistnost m, akce a, pravidelna p  
  WHERE r.mis_id = m.mis_id AND m.zkratka='E112' AND  
    r.a_id = a.id AND p.id = a.id AND  
    p.akademicky_rok = '2012/2013' AND p.semestr = 1  
  GROUP BY p.den  
)  
SELECT d.jmeno, COALESCE(day_totals.vytizeni,0)  
FROM den_v_tydnu d NATURAL LEFT JOIN day_totals
```

## 6.2.3. Manipulace s daty

- Příkaz INSERT

```
INSERT INTO jm_tabulky [(jm_sloupce, ...)] zdroj
```

- **Pokusí se vložit do tabulky jeden řádek nebo obsah jiné tabulky**
- Zdroje pro vkládání:
  - Řádek implicitních hodnot (z příkazu CREATE TABLE):

```
DEFAULT_VALUES
```

- Řádek zadaných hodnot:

```
VALUES (skalární_výraz|NULL|DEFAULT, ...)
```

```
Př.) INSERT INTO Klient  
VALUES ('440726/0672', 'Jan Novák', 'Cejl 8', 'Brno')
```

- Výsledek poddotazu:

```
tabulkový_výraz
```

## 6.2.3. Manipulace s daty

- Příkaz INSERT (pokračování)

Př.) „Vlož do tabulky ZJ informace o klientech s účtem na Jánské.“

```
INSERT INTO ZJ
  SELECT DISTINCT K.*
  FROM Klient K, Ucet U
  WHERE K.r_cislo=U.r_cislo AND U.pobocka='Jánská'
```

- Příkaz DELETE (prohledávací), SQL:2008 navíc TRUNCATE TABLE

```
DELETE FROM jm_tabulky
  [WHERE podmínka]
```

- Pokusí se zrušit jeden nebo několik řádků tabulky splňující podmínku

Př.) „Zruš informace o klientech bez účtu.“

```
DELETE FROM Klient
  WHERE r_cislo NOT IN (SELECT r_cislo FROM Ucet)
```

## 6.2.3. Manipulace s daty

- Příkaz UPDATE (prohledávací)

```
UPDATE jm_tabulky  
    SET jm_sloupce = výraz|NULL|DEFAULT, ...  
    [WHERE podmínka]
```

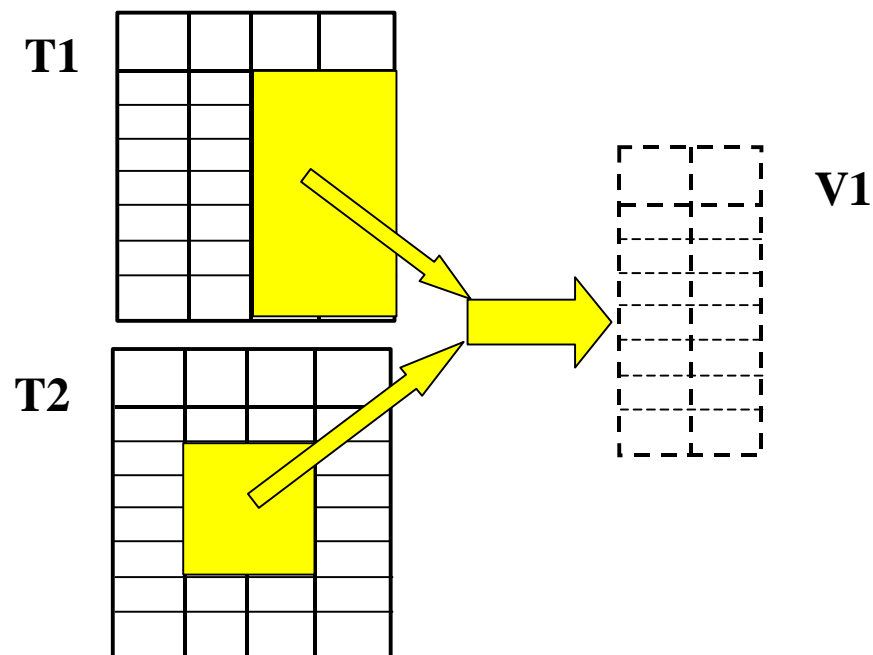
- Pokusí se změnit hodnoty specifikovaných sloupců v řádcích splňujících podmínku

Př.) „Poznač vklad 1000 Kč na účet číslo 100.“

```
UPDATE Ucet  
SET stav=stav+1000  
WHERE c_uctu=100
```

## 6.2.4. Pohledy

- Pojmenované **virtuální tabulky** odvozené od bázových



- Vytvoření pohledu

```
CREATE VIEW jm_pohledu [(jm_sloupce , ...)]  
    AS tab_výraz  
    [WITH CHECK OPTION]
```

- Uloží definici pohledu do systémového katalogu
- Sloupce musí mít jednoznačná jména (případně přejmenovaná).

## 6.2.4. Pohledy

- Vytvoření pohledu (pokračování)

Př.) Pohled pro klienty pobočky Jánská.

```
CREATE VIEW Janska AS
  SELECT K.*
  FROM Klient K, Ucet U
  WHERE K.r_cislo=U.r_cislo AND U.pobočka='Jánská'
  WITH CHECK OPTION
```

- Zrušení pohledu

```
DROP VIEW jm_pohledu [RESTRICT|CASCADE]
```

- Zruší informaci o pohledu ze systémového katalogu




## 6.2.4. Pohledy

- Manipulace na pohledech

- Při dotazu se provede transformace na operace nad báзовými tabulkami.

**Př.)** `SELECT * FROM Janska WHERE mesto = 'Brno'`



`SELECT K.*  
FROM Klient K, Ucet U  
WHERE K.mesto = 'Brno' AND  
K.r_cislo=U.r_cislo AND U.pobocka='Jánská'`

- Aktualizovatelnost pohledů

- SŘBD musí být schopen jednoznačně transformovat operace vložení, zrušení a modifikace řádku pohledu na operace nad zdrojovými báзовými tabulkami pohledu.

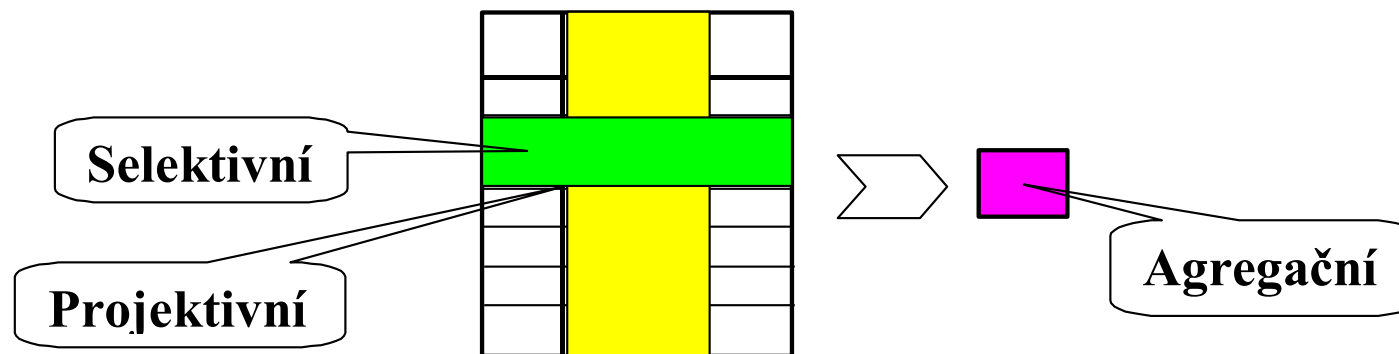
**Př.)** `CREATE VIEW pocty (nazev,pocet)  
AS SELECT pobocka, COUNT(*)  
FROM Ucet  
GROUP BY pobocka;`

## 6.2.4. Pohledy

- Aktualizovatelnost pohledů (pokračování)

- Pohledy s klauzulemi DISTINCT, GROUP BY, HAVING, s agregačními funkcemi a spojující několik tabulek umožňují jen čtení.

Př.) Aktualizovatelnost pohledů nad jednou tabulkou



- Selektivní pohled

```
CREATE VIEW Brnensti AS  
SELECT * FROM Klient WHERE mesto='Brno'
```

## 6.2.4. Pohledy

- Aktualizovatelnost pohledů (pokračování příkladu)

- **Projektivní pohled bez PK**

```
CREATE VIEW Brnensti1 AS
  SELECT jmeno, ulice FROM Klient WHERE mesto='Brno';
INSERT INTO Brnensti1
  VALUES ('Josef Vlk', 'Koliště 55')
```

- **Projektivní pohled s PK, sloupce mimo pohled dovolují NULL**

```
CREATE VIEW Brnensti2 AS
  SELECT r_cislo, jmeno FROM Klient WHERE mesto='Brno';
INSERT INTO Brnensti2
  VALUES ('112233/4444', 'Josef Vlk')
```

- **Agregační**

```
CREATE VIEW Pocty (nazev, pocet) AS
  SELECT pobočka, COUNT(*) FROM Ucet GROUP BY pobočka;
INSERT INTO Pocty VALUES ('Panská', 20)
```

## 6.2.4. Pohledy

- Význam klauzule WITH CHECK OPTION
  - Kontrola, že při aktualizaci nedochází k porušení definice pohledu

```
CREATE VIEW Brnensti AS
  SELECT * FROM Klient WHERE mesto='Brno'
WITH CHECK OPTION

UPDATE Brnensti
SET mesto='Praha'
WHERE r_číslo=...
```

## 6.2.4. Pohledy

- **Materializované pohledy**

- Pohledy, u nichž je výsledek dotazu definujícího pohled skutečně fyzicky uložen v databázi a je zajištěna aktualizace obsahu.

```
CREATE MATERIALIZED VIEW MBrnensti  
REFRESH ON COMMIT AS  
    SELECT * FROM Klient WHERE mesto='Brno '
```

- **Důvod**
  - Zvýšení efektivity, resp. omezený přístup k datům.
- **Hlavní oblasti použití:**
  - Datové sklady – sumarizační pohledy.
  - Distribuované databáze – replikace dat v uzlech.
  - Mobilní databáze – materializace pohledů používaných mobilními klienty.

## 6.2.4. Pohledy

- Použití pohledů – mezi hlavní důvody použití pohledů patří:
  - Omezení přístupu, skrytí logické struktury (bezpečnost)
  - Skrytí složitosti dotazu (zjednodušení)
  - Skrytí způsobu získání dat (nezávislosti na případné změně dotazu)

## 6.2.5. Přístup k systémovému katalogu (slovníku dat)

- U relačních systémů má katalog stejné rozhraní jako uživatelská DB s určitými omezeními, tj. tabulky (nejčastěji pohledy).
- Standard SQL (část 11 SQL/Schemata) pohledy:

`TABLES (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, ...)`

`COLUMNS (TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, ...)`

**Př) Oracle: pohledy: ALL\_, DBA\_, USER\_**

`USER_TABLES (TABLE_NAME, TABLESPACE_NAME, ...)` ,

`USER_TAB_COLUMNS (TABLE_NAME, COLUMN_NAME, DATA_TYPE, ...)`

**„Které sloupce má tabulka Klient?“**

```
SELECT column_name
FROM User_tab_columns
WHERE table_name = 'KLIENT'
```

- Údaje z katalogu lze přímo pouze číst, ostatní manipulace se dějí zprostředkovaně (CREATE, ALTER, DROP).
- Někdy se používají speciální příkazy (**Př.: MySQL: SHOW TABLES; DESCRIBE Klient;**)

## 6.2.6. Práce s chybějící informací

- potřeba v praxi
- řešení:
  - jedna vybraná hodnota oboru
  - speciální „hodnota“ (**NULL** v SQL)
- vliv na operace ( $A+B$ ,  $A>B$ ) → **tříhodnotová logika** (3VL) - {true, false, **unknown**}
- Pravidla
  - skalární výrazy - výsledek NULL, je-li některý z operandů NULL
  - porovnání - výsledek je *unknown*, je-li některý z operandů NULL
  - agregační funkce - jako neutrální hodnota vůči prováděné operaci
  - klauzule WHERE, HAVING - vybírají se řádky s hodnotou podmínky *true*
  - porovnání řádků

a	NULL	c
a	NULL	c

ani  $r1 = r2$ , ani  $r1 <> r2$ ,  
pro DISTINCT *true*



## 6.2.6. Práce s chybějící informací

- Testování chybějící informace

`jm_sloupce IS [NOT] NULL`

- Vnější spojení (OUTER JOIN - pravé, levé, úplné), vnější sjednocení

Př.)

T1

A	B	X
0	A	x
1	A	x
3	C	z

T2

X	C	D
x	1	0
x	2	0
y	3	1

Výsledek?

T1 NATURAL LEFT JOIN T2

T1 NATURAL RIGHT JOIN T2

T1 NATURAL FULL JOIN T2

T1 UNION JOIN T2

„Kolik mají jednotliví klienti účtů a peněz na nich?“

```
SELECT jmeno, r_cislo, COUNT(c_uctu) pocet, SUM(stav) celkem
FROM Klient NATURAL LEFT JOIN Ucet
GROUP BY jmeno, r_cislo
ORDER BY celkem DESC
```

## 6.2.7. Další příkazy SQL

- Integritní omezení
  - NOT NULL, CHECK, UNIQUE, ASSERTION, ...
- Řízení přístupových práv (viz kap. 7)
  - GRANT, REVOKE
- Řízení sezení (viz kap. 11)
  - CONNECT, DISCONNECT, SET CONNECTION, ...
- Transakční zpracování (viz kap. 11)
  - COMMIT, ROLLBACK, SET TRANSACTION (izolační úroveň, ...), ...
- Další

## 6.3. Programování s SQL

- Tři možné kontexty použití jazyka SQL (binding styles):
  - přímý (direct) SQL
  - hostitelská verze (embedded) SQL
  - jazyk modulů – možnost vytvářet tzv. SQL moduly pro konkrétní vyšší programovací jazyk. Modul obsahuje SQL procedury, každá obsahuje jediný příkaz SQL. SQL modul tvoří kompilační jednotku, jehož procedury lze volat z daného vyššího programovacího jazyka.
- SQL/92 není výpočetně úplný
- Hostitelská verze SQL je obecně mocnější než přímý SQL.

## 6.3.1. Hostitelská verze SQL (Embedded SQL)

- Zásady

- Příkazy mají tvar

```
EXEC SQL SQL_příkaz
```

a jsou ukončeny dle zvyklosti jazyka (např. ; pro C).

- Libovolný příkaz přímého SQL lze použít v hostitelském prostředí.
- Odkazy na proměnné host. jazyka (vázané - „bind“) začínají „:“.
- Referované host. proměnné musí být definovány v deklarční sekci:

```
EXEC SQL BEGIN DECLARE SECTION
```

```
.....
```

```
END DECLARE SECTION
```

- Každý program s vloženým SQL musí obsahovat hostitelskou proměnnou SQLCODE nebo SQLSTATE, jejíž hodnoty nastavuje SŘBD po provedení každého příkazu SQL.

## 6.3.1. Hostitelská verze SQL (Embedded SQL)

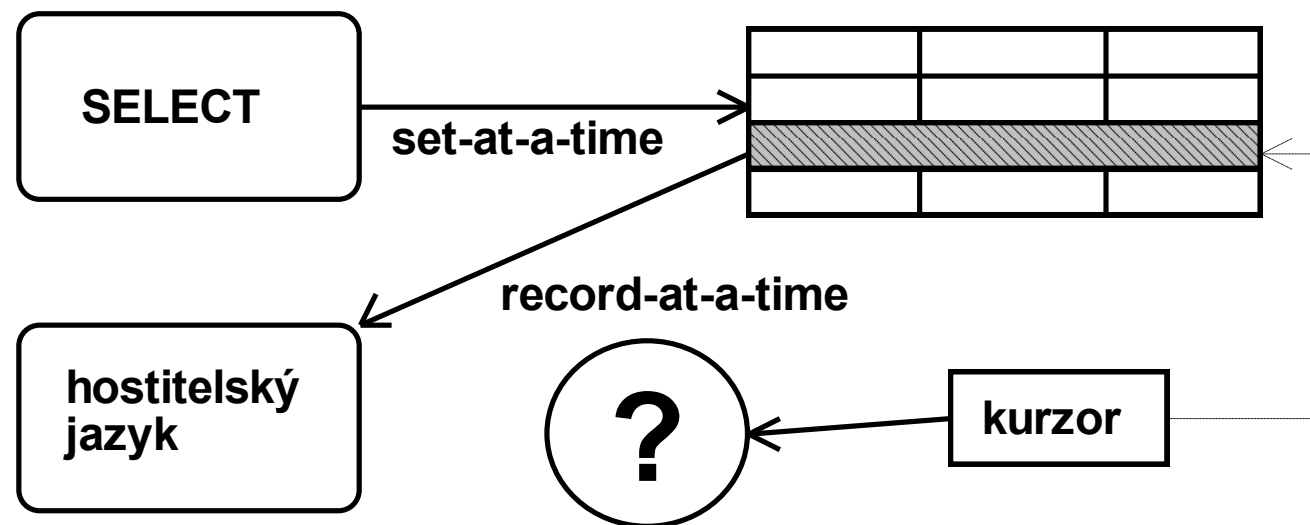
- Zásady (pokračování)

- Hostitelské proměnné musí být vhodného typu s ohledem na použití.
- Za každým příkazem SQL by měl následovat test SQLCODE nebo SQLSTATE, příkaz WHENEVER zjednodušuje:

**EXEC SQL WHENEVER podmínka akce**

- podmínkou je SQLERROR nebo NOT FOUND, akci CONTINUE nebo GOTO návěští

- Pojem kurzor



## 6.3.1. Hostitelská verze SQL (Embedded SQL)

### ▫ Příkazy nevyžadující kurzor:

- Jednořádkový SELECT

```
SELECT ... INTO host_pr [INDICATOR indik], ... FROM ...
```

- INSERT, prohledávací UPDATE a DELETE

### ▫ Příkazy související s kurzorem

- Deklarace kurzoru

```
DECLARE [INTENSIVE|SCROLL] jm_kurzoru CURSOR  
FOR př_select_přip_s ORDER BY  
[FOR READONLY|UPDATE[OF sloupce]]
```

- Zásady pro aktualizovatelnost podobné pohledům

```
Př.) DECLARE Janska CURSOR FOR  
      SELECT K.r_cislo, K.jmeno, K.ulice, K.mesto  
      FROM Klient K, Ucet U  
      WHERE K.r_cislo=U.r_cislo AND pobočka='Jánská'
```

## 6.3.1. Hostitelská verze SQL (Embedded SQL)

### ▫ Příkazy související s kurzorem (pokračování)

- Provedení příkazu

```
OPEN jm_kurzoru
```

```
Př.) OPEN Janska
```

- Výběr řádku tabulky

```
FETCH [ [NEXT|PRIOR|FIRST|...] FROM] jm_kurzoru  
      [INTO seznam_proměnných]
```

```
Př.) FETCH Janska INTO :rc, :jm, :ul, :mesto
```

- Uzavření (deaktivace) kurzoru

```
CLOSE jm_kurzoru
```

```
Př.) CLOSE Janska
```

- Poziční varianty příkazu UPDATE a DELETE

```
... WHERE CURRENT OF jm_kurzoru
```

## 6.3.1. Hostitelská verze SQL (Embedded SQL)

- Příkazy související s kurzorem (pokračování)

### Př.) Práce s kurzorem v PL/SQL (Oracle)

```
DECLARE CURSOR z IS
    SELECT r_cislo, jmeno FROM Klient WHERE mesto = 'Brno' ;
...
BEGIN
    OPEN z ;
    LOOP
        FETCH z INTO rc, jm;
        EXIT WHEN z%NOTFOUND;
        ...
    END LOOP;
    CLOSE z ;
END ;
```



## 6.3.2. Dynamický SQL

- Poskytuje možnost vytváření příkazů SQL jako textových řetězců za běhu
- Vytvoření příkazu

```
PREPARE jméno_příkazu FROM řetězec | proměnná
```

- *přípravitelný příkaz* - jednořádkový SELECT bez INTO, INSERT, prohledávací UPDATE a DELETE, specifikační část deklarace kurzoru
  - náhrady - „?“
- Vykonání příkazu

```
EXECUTE jm_příkazu [INTO ...] [USING vstupní_hodnoty]
```

- Uvolnění prostoru

```
DEALLOCATE PREPARE jm_příkazu
```

- Vytvoření příkazu a bezprostřední provedení

```
EXECUTE IMMEDIATE řetězec|proměnná
```

## 6.3.2. Dynamický SQL

- Použití v definici kurzoru

```
DECLARE jméno_kurzuor CURSOR FOR jméno_příkazu
```

### Př.) Oracle Pro\*C

```
sprintf(s1,"%s","UPDATE Klient SET jmeno=? WHERE r_cislo=?");  
EXEC SQL PREPARE prikaz FROM :s1;  
EXEC SQL EXECUTE prikaz USING :nove_jmeno,:rc;
```

- Pružnost vs. efektivnost
  - možnost sestavení příkazu až za běhu programu
  - kompilace až v době běhu  $\Rightarrow$  kontroly, pozdní vazba

## 6.4.1. Další relační jazyky – jazyk QBE (Query By Example)

- vyvinutý firmou IBM v 70.letech - původně DBS i jazyk
- k dispozici podpora pro dotazování příkladem na úrovni vývojových prostředí (generátory formulářů) a dotazovacích nástrojů pro koncové uživatele
- původně založen na použití tabulek, dnes zpravidla použití formulářů

Př.) „Vypiš klienty spořitelny“

KLIENT	R_CISLO	JMENO	ULICE	MESTO
P.				

- lze používat proměnné - \_jméno
- P. – které sloupce ve výsledku (tvar výsledné tabulky)
- výsledek lze uspořádat, např, P.AO(1)

Př.) „Vypiš účty pobočky Jánská s částkou větší než 100000.“

UCET	C_UCTU	R_CISLO	STAV	POBOCKA
	P._u	P._r	P.>100000	Jánská

## 6.4.1. Další relační jazyky – jazyk QBE (Query By Example)

- lze se dotazovat na několik tabulek (spojovat informace)

Př.) „Kteří zákazníci mají účet v pobočce Jánská?“

KLIENT	R_CISLO	JMENO	ULICE	MESTO
P.	_x			

UCET	C_UCTU	R_CISLO	STAV	POBOCKA
		_x		Jánská

- agregační funkce - CNT, SUM, AVG, MAX, MIN, povinně s ALL.

Př.) „Kolik zákazníků má účet u pobočky Jánská?“

UCET	C_UCTU	STAV	R_CISLO	POBOCKA
			P.CNT.UNQ.ALL.	Jánská

## 6.4.1. Další relační jazyky – jazyk QBE (Query By Example)

- Dotazování příkladem v systémech s GUI
  - Někdy označované jako GQBE (Graphical Query By Example)

### Př.) Microsoft Access

Microsoft Access

Soubor Úpravy Zobrazit Vložit Dotaz Nástroje Okno nápověda

Top ten zákazníků knihovny : Výběrový dotaz

VYPUJCKY

\*

JMENO

TELEFON

ADRESA

LOGIN

KNIHY

INV\_CISLO

INV\_ROK

VYPUJCKA

INVC

DNE

Pole:	JMENO	TELEFON	TITUL	VYPUJCKA	
Tabulka:	VYPUJCKY	VYPUJCKY	KNIHY	KNIHY	
Souhrn:	Seskupit	Seskupit	Count	Kde	
Řadit:			sestupně		
Zobrazit:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Kritéria:	<>"nikdo"			<>"ztráta" And <>"vy"	
nebo:					

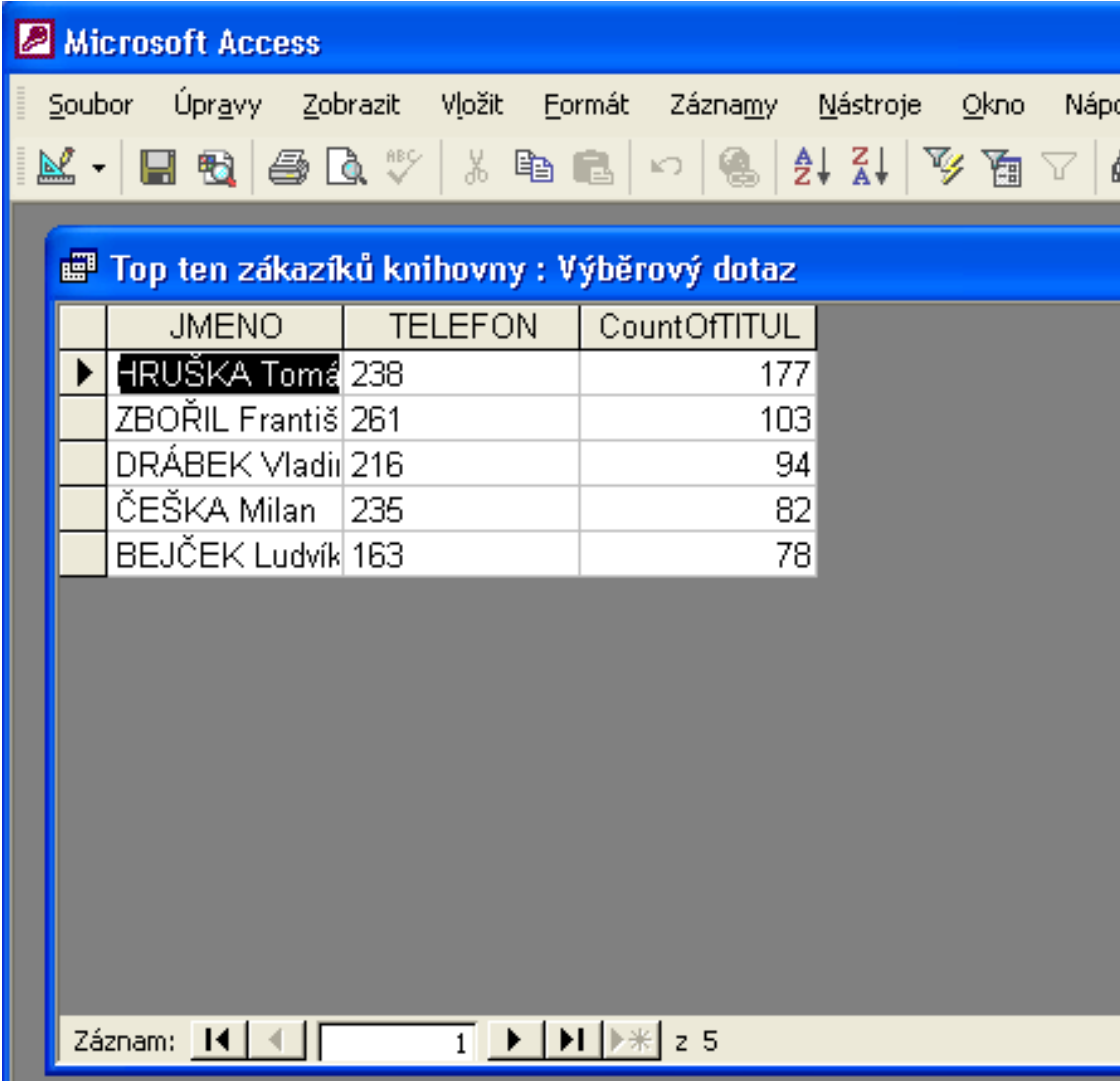
## 6.4.1. Další relační jazyky – jazyk QBE (Query By Example)

**Př.) Microsoft Access (pokračování) – vygenerovaný dotaz**

```
SELECT TOP 5 PERCENT VYPUJCKY.JMENO, VYPUJCKY.TELEFON,  
Count(KNIHY.TITUL) AS CountOfTITUL  
FROM VYPUJCKY LEFT JOIN KNIHY ON VYPUJCKY.JMENO =  
KNIHY.VYPUJCKA  
WHERE ( ( (KNIHY.VYPUJCKA) <> "ztráta" AND  
(KNIHY.VYPUJCKA) <> "vyřazeno" ) )  
GROUP BY VYPUJCKY.JMENO, VYPUJCKY.TELEFON  
HAVING ( ( (VYPUJCKY.JMENO) <> "nikdo" ) )  
ORDER BY Count(KNIHY.TITUL) DESC;
```

## 6.4.1. Další relační jazyky – jazyk QBE (Query By Example)

### Př.) Microsoft Access (pokračování) – výsledek dotazu



The screenshot shows the Microsoft Access application window. The title bar reads 'Microsoft Access'. The menu bar includes 'Soubor', 'Úpravy', 'Zobrazit', 'Vložit', 'Formát', 'Záznamy', 'Nástroje', 'Okno', and 'Nápověda'. The toolbar contains various icons for file operations, editing, and navigation. The main window displays a table titled 'Top ten zákazníků knihovny : Výběrový dotaz'. The table has three columns: 'JMENO', 'TELEFON', and 'CountOfTITUL'. The data is as follows:

JMENO	TELEFON	CountOfTITUL
HRUŠKA Tomáš	238	177
ZBOŘIL František	261	103
DRÁBEK Vladimír	216	94
ČEŠKA Milan	235	82
BEJČEK Ludvík	163	78

At the bottom of the window, the status bar shows 'Záznam: 1 z 5' with navigation buttons.

## 6.4.2. Další relační jazyky – jazyk Datalog

- Neprocedurální dotazovací jazyk vycházející z jazyka pro logické programování Prolog.
- Program je tvořen množinou pravidel, která definují pohledy.

Př.) Účty u pobočky Jánská

```
uctyJanska (CU, S, V) :- ucet (CU, S, V, "Jánská")
```

„Kolik je na účtu číslo 100 a kdo je vlastníkem?“

```
? uctyJanska (100, S, V)
```

- Existují implementace Datalogu, které umožňují rekurzivní dotazy.

Př.) Najdi zaměstnance přímo či nepřímo řízené panem Novákem.

```
podNovakem (X) :- vedouci (X, "Novák")
```

```
podNovakem (X) :- vedouci (X, Y), podNovakem (Y)
```

```
? podNovakem (X)
```



# Literatura

1. Silberschatz, A., Korth H.F., Sudarshan, S.: Database System Concepts. Fifth Edition. McGRAW-HILL. 2006, str. 75-162.
2. Lemahieu, W., Broucke, S., Baesens, B.: Principles of Database Management. The Practical Guide to Storing, Managing and Analyzing Big and Small Data. Cambridge University Press 2018, str. 146-206.

# SQL skripty

## 1. Ke kap. 6.2.2. Definice dat:

- *banka\_CREATE.sql* – ukázky použití příkazů DDL.

## 2. Ke kap. 6.2.3. Manipulace s daty:

- *banka\_CREATE\_DB.sql* – vytvoření ukázkové DB (stačí spustit jako celek)
- *banka\_SELECT.sql* – příklady probírané na přednášce při výkladu příkazu SELECT a některé další, je zde ukázán i způsob řešení tzv. výběru TOP K řádků (obdoba ORDER BY ... LIMIT..)
- *banka\_JOIN.sql* – varianty operace spojení a vyjádření v SQL
- *banka\_UNION.sql* – příklady na sjednocení
- *banka\_funkce.sql* – použití některých funkcí COALESCE, NULLIF a výrazu CASE
- *kolize\_jmen.sql* – použití uvozovek pro řešení kolize identifikátorů s rezervovanými.

# SQL skripty (pokračování)

## 3. Ke kap. 6.2.4. Pohledy:

- *banka\_VIEW.sql* – příklady probírané na přednášce při výkladu k pohledům.

## 4. Ke kap. 6.2.5. Přístup k systémovému katalogu (slovníku dat):

- *banka\_katal.sql* – ukázky přístupu k systémovému katalogu.

## 5. Ke kap. 6.2.6. Práce s chybějící informací:

- *null.sql* – ukázky vlivu chybějící informace.