

---

## **3. Jazyky relačních databázových systémů**

<b>3.1. Tabulky ilustračního příkladu - Spořitelna .....</b>	<b>3</b>
<b>3.2. Jazyk SQL.....</b>	<b>5</b>
<b>3.2.1. Úvod .....</b>	<b>5</b>
<b>3.2.2. Definice dat.....</b>	<b>7</b>
<b>3.2.3. Manipulace s daty .....</b>	<b>14</b>
<b>3.2.4. Pohledy .....</b>	<b>29</b>
<b>3.2.5. Přístup k systémovému katalogu (slovníku dat).....</b>	<b>36</b>
<b>3.2.6. Práce s chybějící informací .....</b>	<b>37</b>
<b>3.2.7. Další příkazy SQL .....</b>	<b>39</b>
<b>3.3. Programování s SQL .....</b>	<b>40</b>
<b>3.3.1. Jazyk modulů.....</b>	<b>40</b>
<b>3.3.2. Hostitelská verze SQL (embedded SQL).....</b>	<b>43</b>
<b>3.3.3. Dynamický SQL .....</b>	<b>48</b>
<b>3.4. Další relační jazyky .....</b>	<b>50</b>
<b>3.4.1. Jazyk QBE (Query By Example).....</b>	<b>50</b>
<b>3.4.2. Jazyk Datalog .....</b>	<b>56</b>

**Literatura..... 57**

### 3.1. Tabulky ilustračního příkladu - Spořitelna

- relační DB je vnímána uživatelem jako kolekce tabulek

#### Klient

r_číslo	jméno	ulice	město
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

#### Účet

č_úctu	stav	r_číslo	pobočka
4320286	52000	440726/0672	Jánská
1182648	10853	530610/4532	Palackého
2075752	126350	440726/0672	Palackého

#### Pobočka

název	jmění
Jánská	1000000
Palackého	500000

## Transakce

č_účtu	č_transakce	datum	Částka
4320286	1	10.10.1998	3000
4320286	2	12.10.1998	- 5000
2075752	1	14.10.1998	- 2000
2075752	2	14.10.1998	10000

## 3.2. Jazyk SQL

### 3.2.1. 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)
  - další standardy X\_OPEN SQL (Unix), SAA-SQL (standard firmy IBM)
  - 1996 - dodatek týkající se uložených modulů (PSM/96)
  - 1999 - poslední sada dokumentů SQL/99 (SQL3)
  - řada dialektů SQL, základem SQL/92 (minimálně úroveň Entry)  
+ vlastní rozšíření

- **Tři možné kontexty použití jazyka SQL (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**

### 3.2.2. Definice dat

- **Základní příkazy:**

- **CREATE** - vytvoření

- **DROP** - zrušení databázového objektu

- **ALTER** - změna vlastností databázového objektu

- **Definice bázové (skutečně existující) tabulky**

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

→ vytvoří novou, prázdnou tabulka + popis uloží do katalogu

- **Definice sloupce**

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

- **Definice integritních omezení (io)**

Integritní omezení jsou omezení kladená na hodnoty ve sloupcích tabulky, aby nedošlo k porušení integrity dat.

- **Neformální definice pojmů důležitých pro integritní omezení tabulky:**

***Kandidátní klíč*** - sloupec, resp. sloupce tabulky (pro složený kandidátní klíč), jehož hodnota, resp. kombinace hodnot je v rámci tabulky unikátní.

***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. Slouží k vytváření vazeb mezi tabulkami. Soulad hodnot cizího klíče a odkazovaného kandidátního klíče se nazývá ***referenční integrita***.

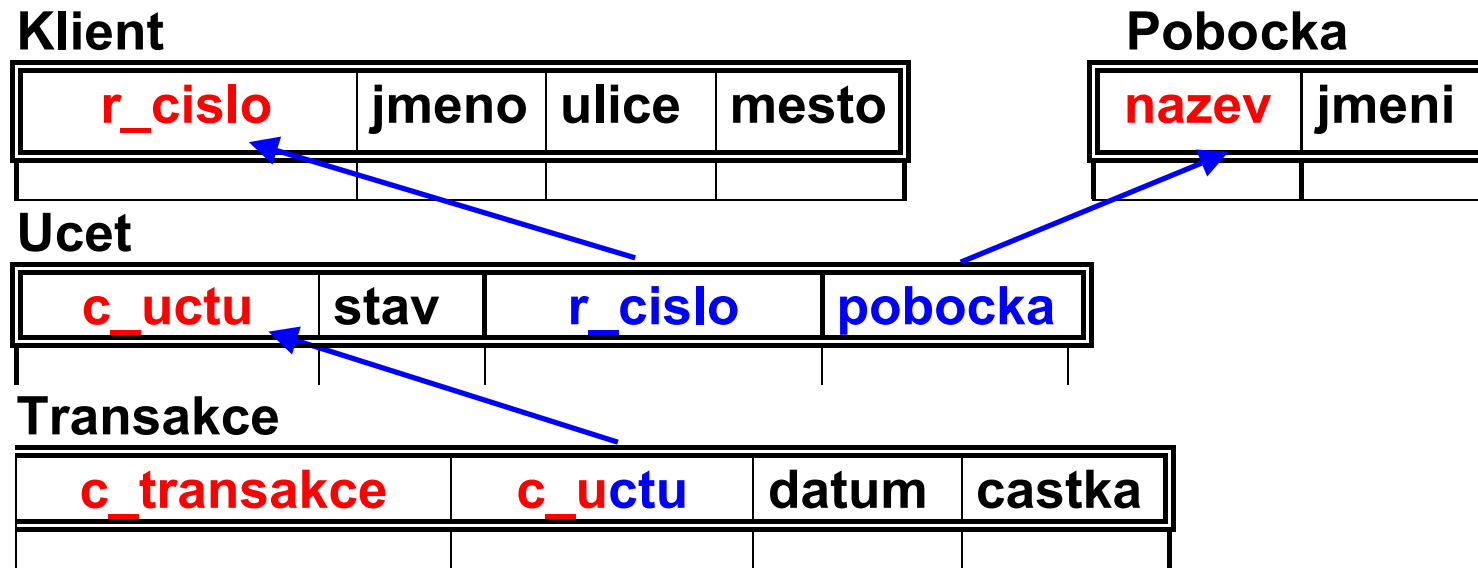
- **Integritní omezení sloupce**

```
NULL, resp NOT NULL  
CHECK (podmíněný_výraz)  
PRIMARY KEY  
UNIQUE  
FOREIGN KEY REFERENCES tabulka [(jm_sloupce)] [událost  
ref_akce]
```

- **Integritní omezení celé tabulky**

```
PRIMARY KEY (jm_sloupce, ...)  
UNIQUE (jm_sloupce, ...)  
FOREIGN KEY (jm_sloupce, ...) REFERENCES  
    tabulka [(jm_sloupce, ...)] [událost ref_akce]  
CHECK (podmíněný_výraz)
```

Př)



```
CREATE TABLE Ucet
  (c_uctu      DECIMAL(7,0) NOT NULL,
  stav        DECIMAL(10,2),
  r_cislo     CHAR(11) NOT NULL,
  pobočka     CHAR(20) NOT NULL,
  PRIMARY KEY (c_uctu),
  FOREIGN KEY (r_cislo) REFERENCES Klient
              ON DELETE CASCADE,
  FOREIGN KEY (pobočka) REFERENCES Pobočka )
```

## ➤ Typy dat

- *skalární*: CHAR(n), VARCHAR(n), BIT(n), BIT VARYING(n), NUMERIC(p, q), DECIMAL(p, q), INTEGER, SMALLINT, FLOAT(p)
- *datum a čas*: DATE, TIME, TIMESTAMP, INTERVAL

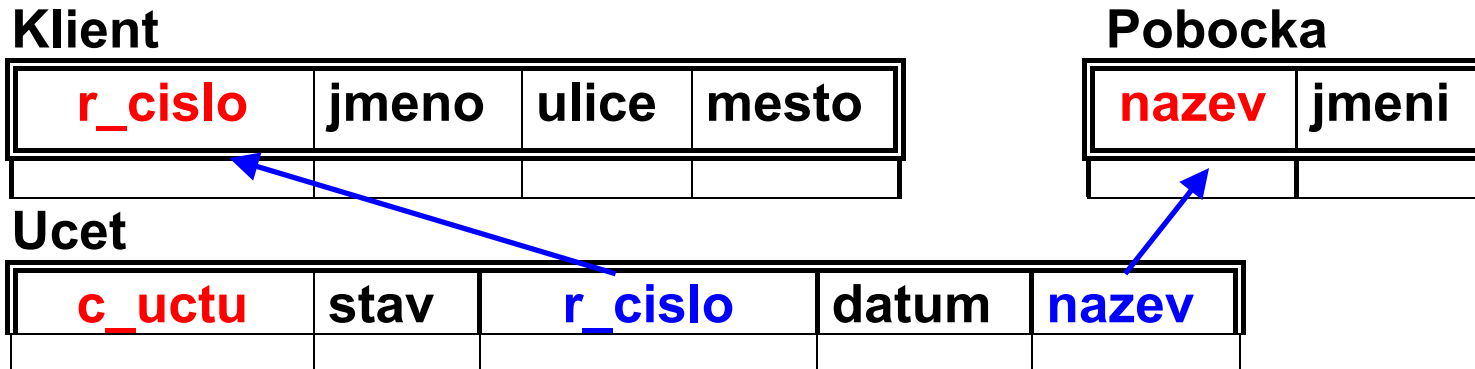
SQL/99 zavádí další datové typy, včetně typů označovaných jako BLOB (binary large objects).

- Referenční událost - aktualizace ( ON UPDATE), rušení (ON DELETE)
- Referenční akce - NO ACTION, CASCADE, SET DEFAULT, SET NULL

*Poznámka: Uvedením 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*

▪

Př)



	SELECT	INSERT	DELETE	UPDATE
Klient	-	PK, CK, NULL	RI Ucet.r_cislo	PK, CK, NULL RI Ucet.r_cislo
Ucet	-	PK, AK, NULL, RI Klient, Pobočka	-	PK, AK, NULL, RI Klient, Pobočka

- Změna báze tabulky

```
ALTER TABLE jm_bázové_tabulky akce
```

- akce - přidání (ADD), zrušení (DROP) sloupce, změna implicitní hodnoty (ALTER); přidání (ADD), zrušení (DROP) io pro tabulku

→ modifikuje tabulku a změní informace v katalogu

- **Zrušení tabulky**

```
DROP TABLE jm_bázové_tabulky
```

→ **zruší tabulku a informace o ní v katalogu**

- **Pohledy (viz dále)**
- **Domény (viz dále)**
- **Další typické databázové objekty (nejsou součástí SQL/92)**

➤ **Index**

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

➤ **Synonymum**

```
CREATE SYNONYM jm_synonyma FOR jm_tabulky
```

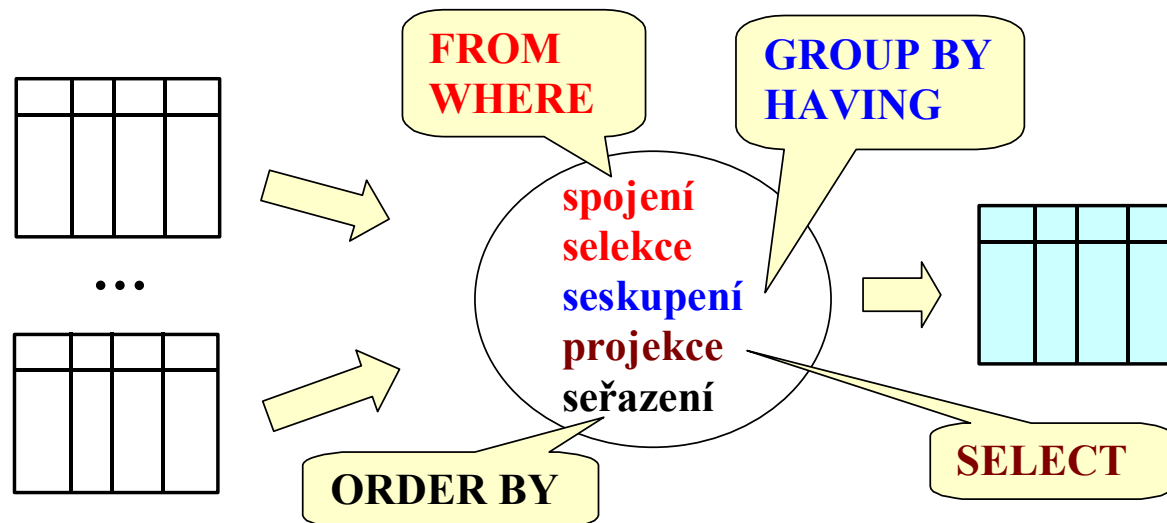
➤ **Generátory posloupností čísel (sekvence, čítače)**

### 3.2.3. Manipulace s daty

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

#### • Příkaz SELECT

```
SELECT [ALL|DISTINCT] položka [[AS] nové_jméno], ...  
FROM tabulkový výraz [[AS] [nové_jméno]], ...  
[WHERE podmínka]  
[GROUP BY jm_sloupceZ|číslo, ...]  
[HAVING podmínka]  
-----  
[ORDER BY jm_sloupceV|číslo [ASC|DESC]], ...
```



➤ **Jednoduché dotazy (nad jednou tabulkou)**

**„Kteří klienti mají účet u spořitelny?“**

```
SELECT r_číslo, jméno  
FROM Klient
```

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

```
SELECT *  
FROM Klient
```

- Uvedením klíčového slova DISTINCT se eliminují duplicitní řádky

**„Ze kterých měst jsou klienti spořitelny?“**

```
SELECT DISTINCT město  
FROM Klient
```

- Klauzule WHERE určuje podmínku pro výběr řádků

**„Které účty jsou u pobočky Jánská?“**

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

- **Přejmenování**

- Lze zavádět nová jména pro sloupce výsledné tabulky (v klauzuli SELECT)
- Lze zavádět nová jména tabulek v klauzuli FROM (tzv. n-ticové proměnné)

staré\_jméno [AS] nové\_jméno

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

- V klauzuli SELECT mohou být výrazy

„Kolik činí jmění poboček v USD při kursu 30 Kč/\$?“

```
SELECT nazev, jmeni/30 jmeni_v_USD  
FROM Pobočka
```

- Výslednou tabulku lze uspořádat - klauzule ORDER BY

```
SELECT nazev, jmeni/30 jmeni_v_USD  
FROM Pobočka  
ORDER BY jmeni_v_USD
```

nebo 2

➤ Spojení informací z více tabulek (operace JOIN)

„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'
```

„Kteří klienti prováděli transakce v pobočce Jánská 12.10.1998?“

```
SELECT DISTINCT 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'
```

▪ Typy spojení:

- **vnitřní (inner)** ◇ **obecné na základě podmínky(⊕join)**  
◇ **na základě rovnosti (equijoin)**  
◇ **přirozené (natural join)**
- **vnější (outer)**

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 JOIN T2 ON A<C  
T1 JOIN T2 ON A=D  
T1 NATURAL JOIN T2  
T1 NATURAL LEFT JOIN T2

- Lze spojit dvě stejné tabulky

„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
```

- 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, ...]
```

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

„Kteří klienti mají účet v pobočce Jánská?“

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

## ➤ Agregáční funkce

```
COUNT (*) |  
AVG | MAX | MIN | SUM | COUNT ([ALL | DISTINCT] jm_sloupce)
```

„Kolik klientů má spořitelna?“

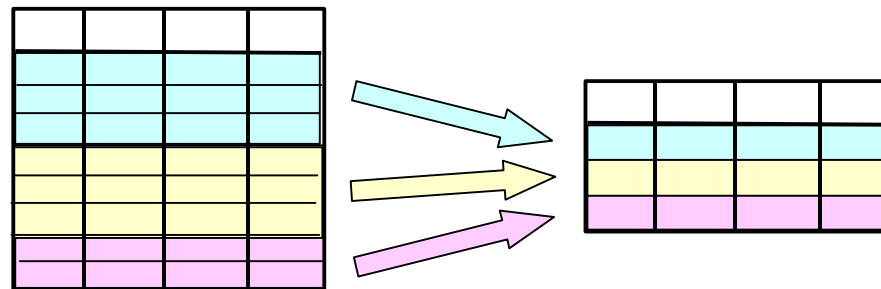
```
SELECT COUNT(*) pocet  
FROM Klient
```

- agregační funkce nelze zanořovat

## ➤ Klauzule GROUP BY

„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)

**„Kolik mají na svých účtech 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.jmeno,K.r_cislo
```

➤ **Klauzule HAVING**

- Analogie WHERE, ale aplikované na celé skupiny

**„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.jmeno,K.r_cislo  
HAVING SUM(stav)>50000
```

➤ **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)
```

**„Kteří mimobrněňší klienti mají uloženo více než brněňší?“**

```
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)
```

- Test na chybějící informaci

```
jm_sloupce IS [NOT] NULL
```

„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
```

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

„Které transakce proběhly v 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'
```

## ▪ Predikát EXISTS

[NOT] EXISTS (tabulkový\_výraz)

- Test na neprázdnot tabulky

„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.pobočka='Jánská' AND
      NOT EXISTS (SELECT *
                  FROM Ucet U
                  WHERE K.r_cislo=U.r_cislo AND
                       U.pobočka<>'Jánská' )
```

- Typický poddotaz s „\*“ v klauzuli SELECT

## ▪ Predikát UNIQUE

[NOT] UNIQUE (tabulkový\_výraz)

- Test na neexistenci duplicitních řádků v tabulce poddotazu.

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

## ▪ 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 '\'**

**„Kteří klienti mají křestní jméno Jan?“**

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

## ▪ Predikát IN

```
konstruktor_řádku [NOT] IN (tabulkový_výraz) |  
skalární_výraz [NOT] IN (seznam_skalárních_výrazů)
```

**„Kteří klienti jsou z Brna nebo Prahy?“**

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

**„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**

- **Obdoba IN s možností testu na shodu s právě jediným řádkem**

➤ Operátory pro sjednocení, rozdíl a průnik tabulek

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

Př) Předpokládejme další tabulku Pujcka

c_pujcky	r_cislo	pobočka	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.pobočka='Jánská'  
UNION
```

```
SELECT K.jmeno, K.r_cislo  
FROM Klient K, Pujcka P  
WHERE K.r_cislo=P.r_cislo AND P.pobočka='Jánská'
```

- Při provedení příkazu se implicitně odstraňují duplicity.

- Příkaz INSERT

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

→ Vloží jeden nebo více řádků tabulky

➤ Zdroje pro vkládání:

- Řádek implicitních hodnot (z příkazu CREATE TABLE)

```
DEFAULT_VALUES
```

- Řádek zadaných hodnot

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

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

- Výsledek poddotazu

```
tabulkový_výraz
```

„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)

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

→ zruší jeden nebo několik řádků tabulky splňující podmínku

„Zruš informace o klientech bez účtu.“

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

- Rozdíl příkazů DELETE FROM T a DROP TABLE T.

- Příkaz UPDATE (prohledávací)

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

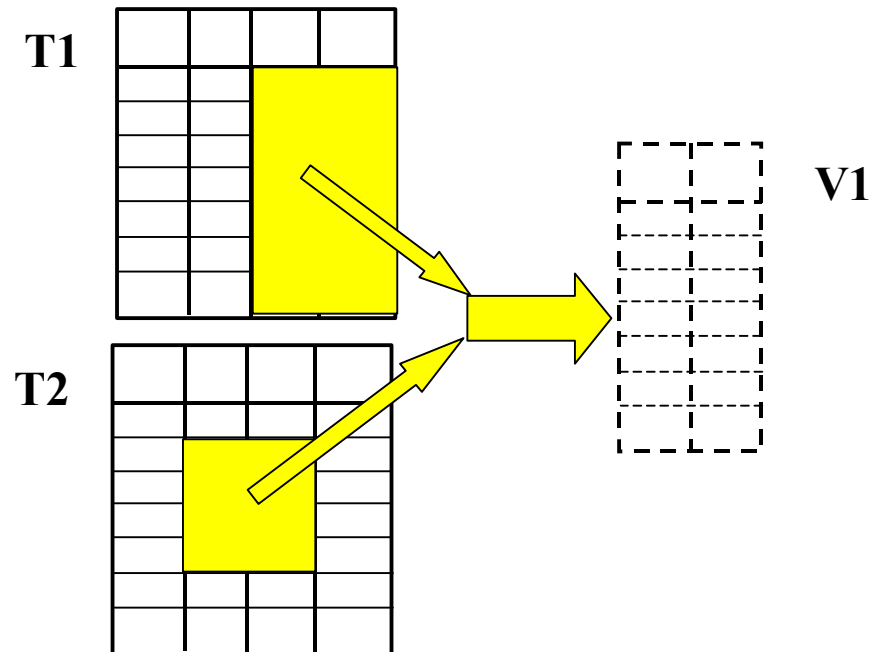
→ změní hodnoty specifikovaných sloupců v řádcích splňujících podmínku

„Poznač vklad 1000 Kč na účet číslo 100.“

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

### 3.2.4. Pohledy

Pojmenované **virtuální tabulky** odvozené z 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).

## Př) Pohled pro klienty pobočky Jánská.

```
CREATE VIEW Janska AS
  SELECT K.*
  FROM Klient K, Ucet U
  WHERE K.r_číslo=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

- 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

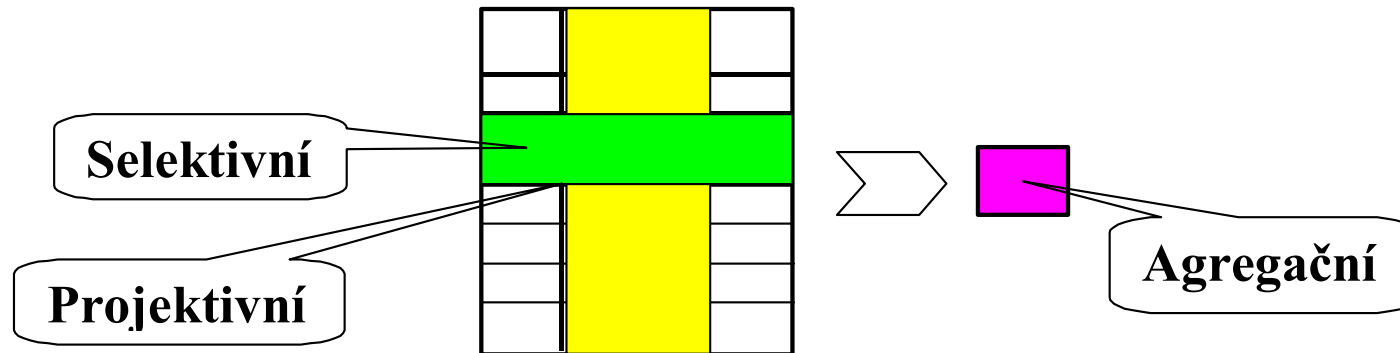
SŘBD musí být schopen jednoznačně transformovat operace vložení řádku, zrušení řádku a modifikace řádku pohledu na operace nad zdrojovými bázovými tabulkami pohledu.

Př)

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

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

## Př) Aktualizovatelnost pohledů nad jednou bázovou tabulkou



- **Selektivní pohled**

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

- **Projektivní *pohled bez PK***

```
CREATE VIEW Brnenst1 AS  
SELECT jmeno, ulice FROM Klient WHERE mesto='Brno'  
INSERT INTO Brnenst1  
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 pobocka, COUNT(*) FROM Ucet GROUP BY pobocka;
INSERT INTO Pocty VALUES ('Panská',20)
```

## ➤ Význam klauzule WITH CHECK OPTION

- Kontrola, že při aktualizaci nedochází k porušení definice pohledu.

```
CREATE VIEW Brněnští AS
    SELECT * FROM Klient WHERE město='Brno'
WITH CHECK OPTION
```

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

## ➤ Materializované pohledy

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

- 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.

## ➤ **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)**

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

U relačních systémů má katalog stejné rozhraní jako uživatelská DB s určitými omezeními, tj. tabulky (nejčastěji pohledy).

**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'
```

**SQLBase:**

```
SYSTABLES (NAME, CREATOR, COLCOUNT, ...),  
SYSCOLUMNS (NAME, TBNAME, COLTYPE, ...)
```

- Údaje z katalogu lze přímo pouze číst, ostatní manipulace se dějí zprostředkovaně (CREATE, ALTER, DROP).

### 3.2.6. 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 - vybírají se řádky s hodnotou podmínky *true*
  - porovnání řádků

a	NULL	c
a	NULL	c

ani  $r1 = r2$ , ani  $r1 \neq r2$ , DISTINCT *true*

- 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
```

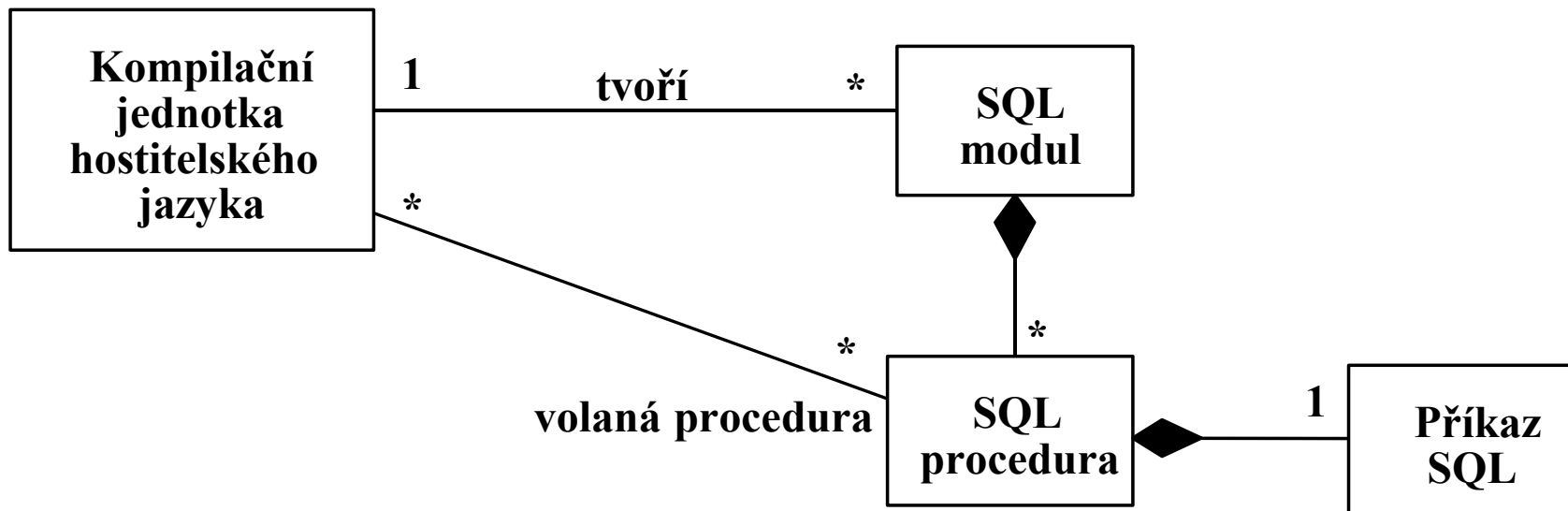
### 3.2.7. Další příkazy SQL

- **Integritní omezení (viz kap. 8)**  
**NOT NULL, CHECK, UNIQUE, ASSERTION, ...**
- **Řízení přístupových práv (viz kap. 7)**  
**GRANT, REVOKE**
- **Řízení sezení (viz kap. 9)**  
**CONNECT, DISCONNECT, SET CONNECTION, ...**
- **Transakční zpracování (viz kap. 9)**  
**COMMIT, ROLLBACK, SET TRANSACTION (izolační úroveň,...),...**
- **Další**

### 3.3. Programování s SQL

- Tři možné kontexty použití jazyka SQL (binding styles):
  - přímý (direct) SQL
  - hostitelská verze (embedded) SQL
  - jazyk modulů
- SQL/92 není výpočetně úplný
- Hostitelská verze SQL je obecně mocnější než přímý SQL.

#### 3.3.1. Jazyk modulů



```
MODULE [jméno_modulu]
LANGUAGE programovací_jazyk
[SCHEMA schéma][AUTHORIZATION uživatel]
[definice_přechodných_tabulek]
definice_kurzorů_a_procedur
```

➤ **Definice procedury:**

```
PROCEDURE jméno (seznam_parametrů);
    příkaz_SQL;
```

**Parametry procedury - SQLCODE (0-OK, 100-NO ROWS, <0-ERROR)  
nebo SQLSTATE, další parametry**

Př)

```
PROCEDURE ZrusKlienta (SQLSTATE, :rc CHAR(11));  
  DELETE  
  FROM Klient  
  WHERE r_cislo = :rc;
```

Použití:

```
...  
char SQLSTATE[6];  
char rodneCislo[12];  
...  
ZrusKlienta(SQLSTATE, rodneCislo); ...
```

### 3.3.2. 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 deklarační sekci:**

```
EXEC SQL BEGIN DECLARE SECTION
```

```
.....
```

```
END DECLARE SECTION
```

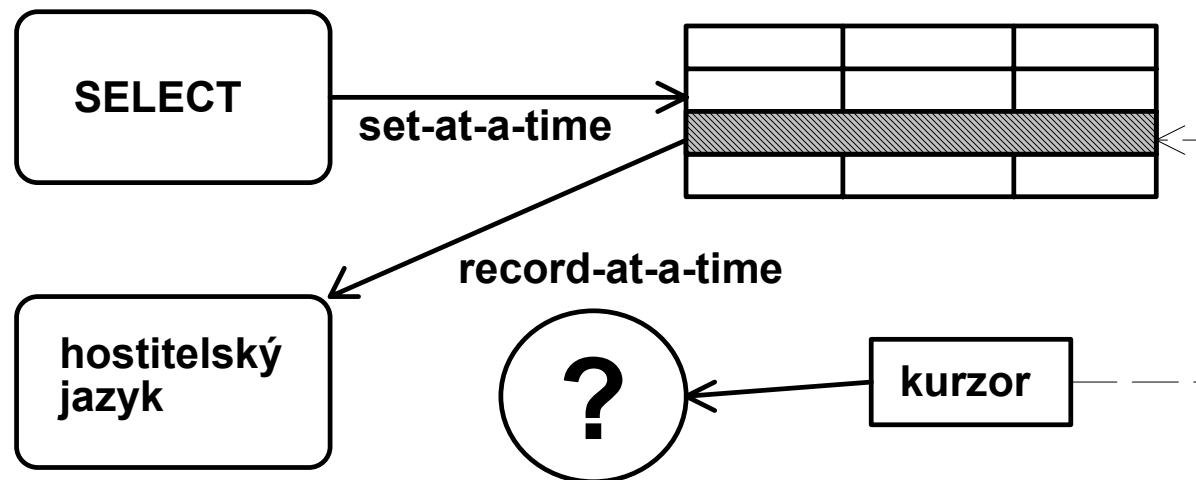
- **Každý program s vloženým SQL musí obsahovat hostitelskou proměnnou SQLCODE nebo SQLSTATE, jejíž hodnoty nastavuje SŘBD po provedení každého příkazu SQL.**
- **Hostitelské proměnné musí být vhodného typu s ohledem na použití.**
- **Proměnné i sloupce mohou mít stejná jména.**

- 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 GO TO návěští.

- Pojem kurzor



➤ 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 pobocka='Jánská'
```

- **Provedení dotazu**

```
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
```

## 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;
```

### 3.3.3. 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
```

- **Použití v definici kurzoru**

```
DECLARE jméno_kurzoru 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;
```

### **Gupta (jazyk SAL)**

```
Call SqlPrepare(hSql,  
    "UPDATE zákazník SET jméno = :nove_jmeno WHERE  
    r_cislo = :r_cislo")  
Call SqlExecute(hSql)  
  
Call SqlImmediate(" .... ")
```

- **Pružnost vs. efektivnost**

- možnost sestavení příkazu až za běhu programu
- kompilace až v době běhu ⇒ kontroly, pozdní vazba

## 3.4. Další relační jazyky

### 3.4.1. Jazyk QBE (Query By Example)

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

„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)

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

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

- logické spojky

„Najdi všechny zákazníky z Brna a Kuřimi.“

KLIENT	JMENO	MESTO
P.		Brno
		Kuřim

- lze se dotazovat na několik tabulek (spojovat informace)

„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.

„Kolik zákazníků má účet u pobočky Jánská?“

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

- operátor G. (group by)

„Jaký je průměrný stav účtu u jednotlivých poboček?“

UCET	C_UCTU	STAV	R_CISLO	POBOCKA
		P.AVG.ALL.		P.G.

- okno podmínky

„Ve kterých pobočkách je průměrný stav vyšší než 100000?“

UCET	C_UCTU	STAV	R_CISLO	POBOCKA
		P.AVG.ALL._x		P.G.

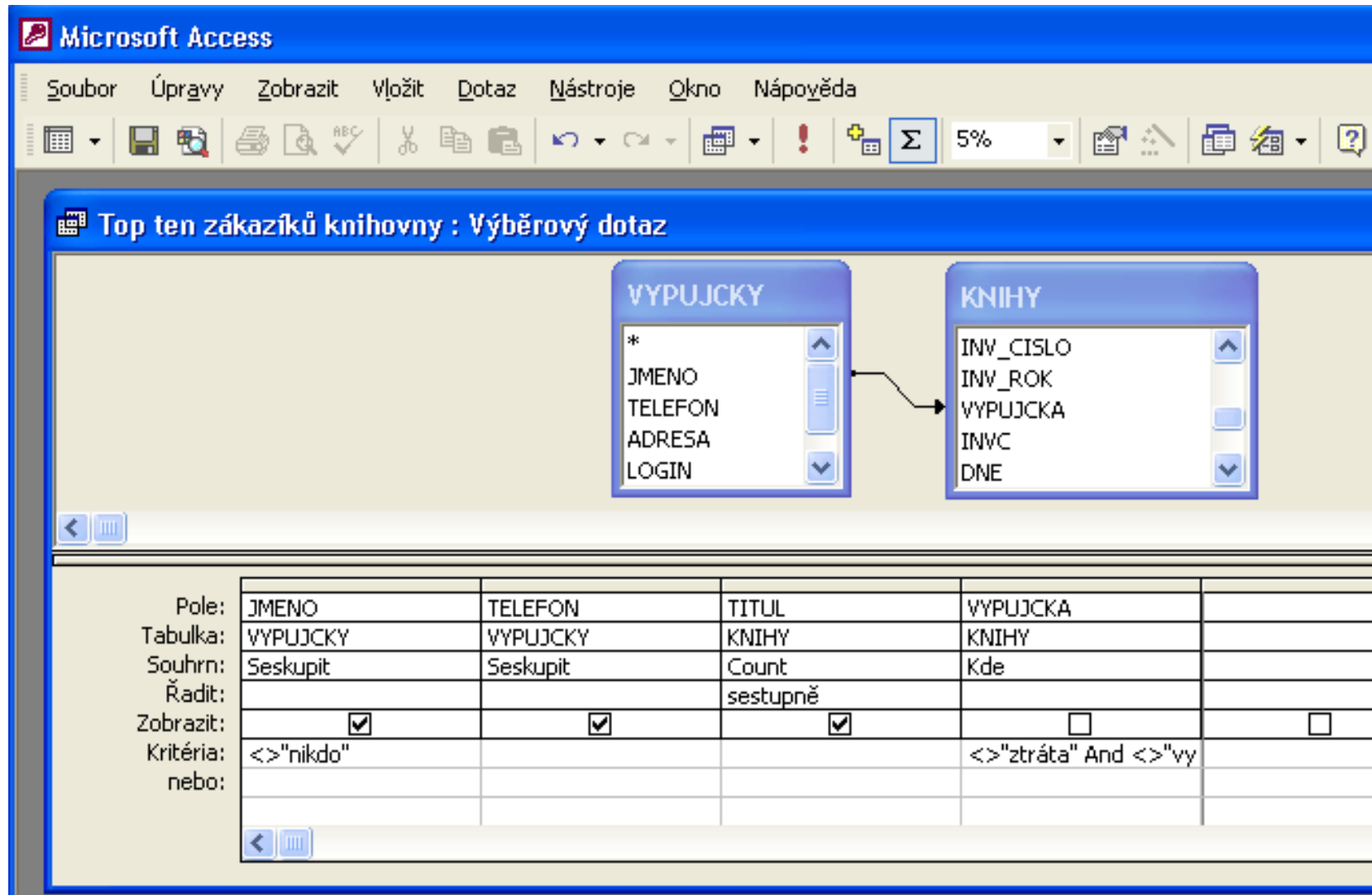
Conditions
AVG.ALL._x>100000

- je-li použit jako manipulační jazyk, pak umožňuje i rušení a vkládání

„Ulož na účet 286549 1000 Kč.“

UCET	C_UCTU	STAV	R_CISLO	POBOCKA
U.	286549	_x+1000		
		_x		

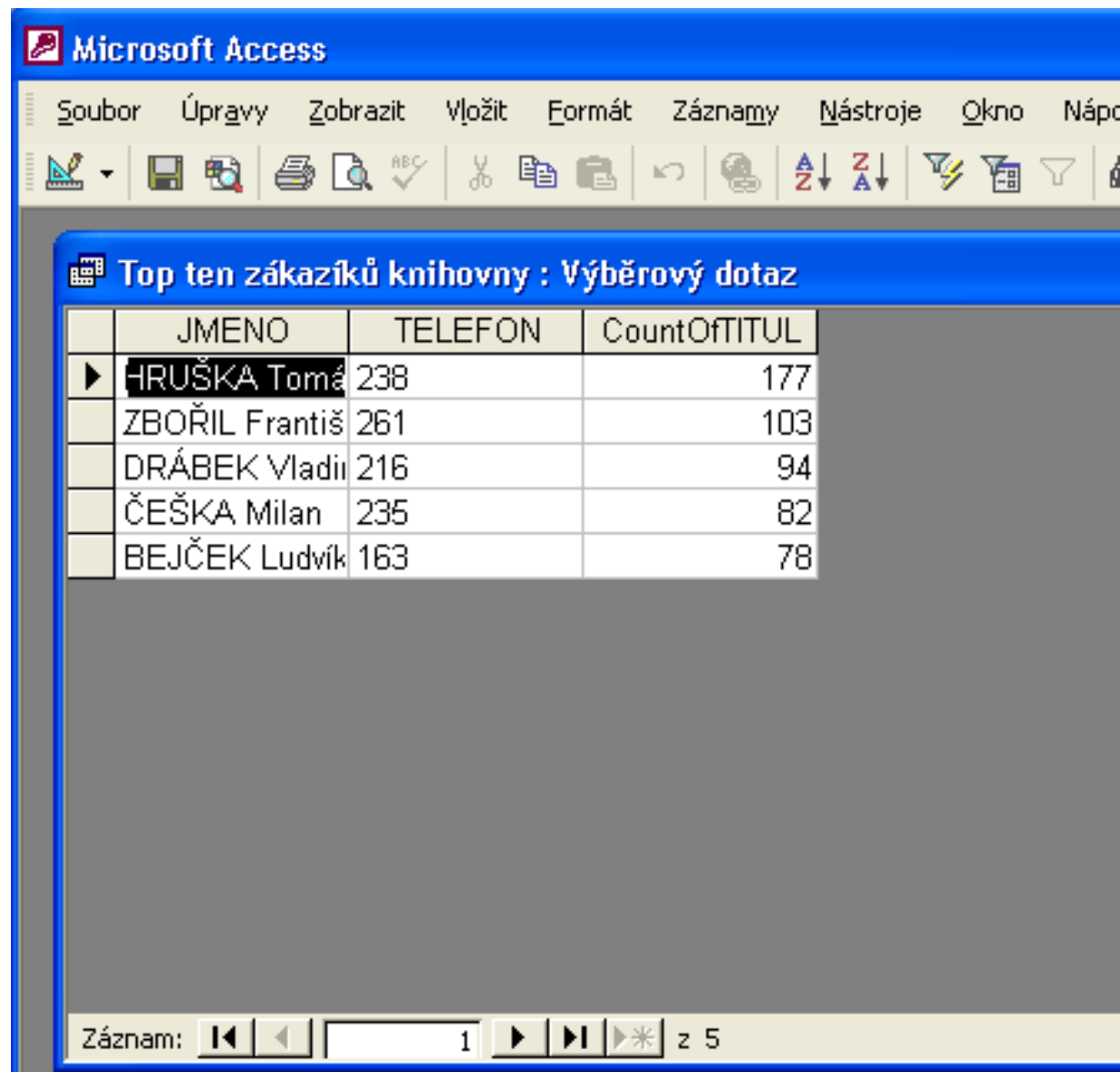
- Dotazování příkladem v systémech s GUI
    - Někdy označované jako GQBE (Graphical Query By Example)
- Př) Microsoft Access**



## 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;
```

## Výsledek dotazu:



Microsoft Access

Soubor Úpravy Zobrazit Vložit Formát Záznamy Nástroje Okno Nápc

Top ten zákazníků knihovny : Výběrový dotaz

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

Záznam: 1 z 5

### 3.4.2. 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)
```

# Literatura

1. Date, C., J., Darwen, H.: A Guide to the SQL Standard. Fourth Edition. Addison-Wesley, 1997.
2. Silberschatz, A., Korth H.F, Sudarshan, S.: Database System Concepts. Fourth Edition. McGRAW-HILL. 2001, str. 135 – 225.
3. Pokorný, J.: Dotazovací jazyky. Science, Veletiny, 1994, str. 47 - 134.
4. Pokorný, J.: Databazová abeceda. Science, Veletiny, 1998, str. 97 – 103.