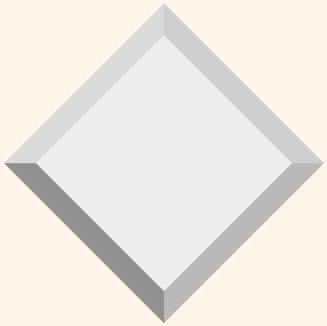


Jazyk SQL
slajdy k přednášce NDBI001

Jaroslav Pokorný

MFF UK, Praha

pokorny@ksi.mff.cuni.cz

Přehled SQL92

1) jazyk pro definici dat,
2) interaktivní jazyk pro manipulaci dat,
3) jazyk pro manipulaci dat v hostitelské verzi,
4) možnost definice pohledů,
5) možnost definice IO,
6) možnost definice přístupových práv,
7) systémový katalog
8) jazyk modulů,
9) řízení transakcí.

Schéma příkladu

VÝPŮJČKY(Č_KOPIE, Č_ZAK, ROD_Č, CENA, DATUM_V)
{údaje o výpůjčkách kopií - číslo zakázky, zákazník, cena, datum navrácení}

KINA(NÁZEV_K, ADRESA, JMÉNO_V) {údaje o kinech a jejich vedoucích}

FILMY(JMÉNO_F, REŽISÉR) {údaje o filmech a jejich režisérech}

PŘEDSTAVENÍ(NÁZEV_K, JMÉNO_F, DATUM)
{údaje o kinech hrajících filmy}

ZÁKAZNÍCI(ROD_Č, Č_ZAK, JMÉNO, ADRESA) {údaje o zákaznícovi}

ZAMĚSTNANCI(OSOBNÍ_Č, ADRESA, JMÉNO, PLAT)
{údaje o zaměstnancích půjčovny}

KOPIE(Č_KOPIE, JMÉNO_F) {kopie filmů}
REZERVACE(JMÉNO_F, ROD_Č) {rezervace filmů zákazníky}

1. Definice dat v SQL

- CREATE TABLE

```
CREATE TABLE VÝPŮJČKY
(č_kopie CHAR(3) NOT NULL,
č_zak CHARACTER(6) NOT NULL,
cena DECIMAL(5,2),
rod_č CHARACTER(10) NOT NULL,
datum_v DATE);
```

Možnosti:

globální temporární,

lokální temporární tabulky

(GLOBAL TEMPORARY, LOCAL TEMPORARY) - nejsou perzistentní

Dále: odvozené tabulky (⊃ pohledy).

1. Definice dat v SQL

IO sloupce

- NOT NULL — sloupec nesmí obsahovat hodnotu NULL,
- DEFAULT — určení implicitní hodnoty sloupce ,
- UNIQUE — všechny hodnoty ve sloupci musí být unikátní, NULL hodnota nevadí,
- PRIMARY KEY — sloupec je primárním klíčem tabulky,
- FOREIGN KEY — sloupec je cizím klíčem definujícím referenční integritu s jinou tabulkou
- CHECK — logický výraz definuje přídavné IO

IO tabulky (např. složený primární klíč), pojmenovaná IO

1. Definice dat v SQL

CREATE TABLE jméno-tabulky (seznam_prvků_tabulky)
seznam_prvků_tabulky ::= prvek_tabulky[,prvek_tabulky]...
prvek_tabulky ::= definice_sloupce | definice_IO_tabulky

- ALTER TABLE

  ADD sloupec, DROP sloupec, ALTER sloupec, ADD CONSTRAINT a DROP CONTRAINT

Př.: ALTER TABLE KINA ADD počet_míst INT

- DROP TABLE

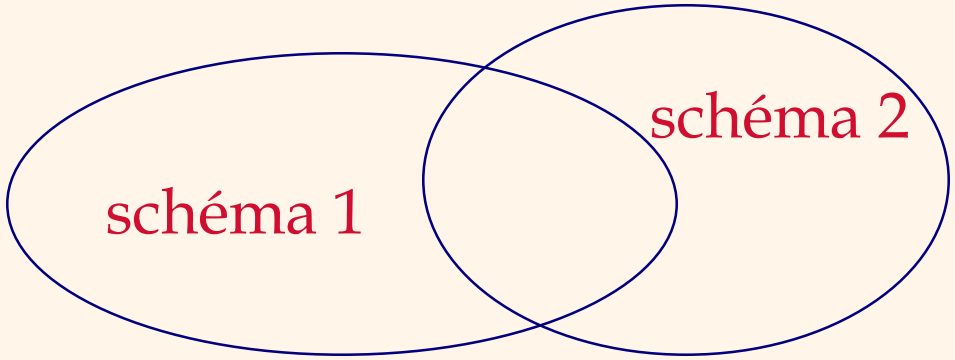
Nové: RESTRICT, CASCADE (i v ALTER TABLE)

- CREATE SCHEMA
  - obsahuje definice základních tabulek, pohledů, domén, integritních omezení, uživatelská práva

1. Definice dat v SQL

- DROP SCHEMA

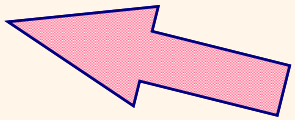
Nové: RESTRICT, CASCADE



Df.: *Databáze v SQL* je kolekce tabulek a pohledů. Může být definována jedním nebo více schématy.

1.1 Typy dat v SQL

- numerické (přesné a aproximativní),
- znakové řetězce,
- bitové řetězce,
- temporální data,
- časové intervaly.



Dále: NULL (je prvkem každého datového typu)

TRUE, FALSE, UNKNOWN

Konverze: automatické, explicitní (funkce CAST)

1.1 Typy dat v SQL

- přesné numerické typy

  INTEGER (celé), SMALLINT („menší“ implementace než INTEGER), NUMERIC, DECIMAL.

  – NUMERIC (p,q), *p* cifer, desetinná čárka, *q* cifer zprava.
  – DECIMAL (podobné NUMERIC) s přesností implementačně definovanou, *p* musí být menší nebo rovno této přesnosti.
  – DECIMAL a NUMERIC jsou funkčně ekvivalentní (ale ne stejné)

1.1 Typy dat v SQL

- aproximativní numerické typy

  FLOAT (reálné, parametrizované event. *p*)
  REAL (reálné, s pevnou přesností danou implementací)

  DOUBLE PRECISION (reálné, s pevnou přesností danou implementací, ale větší než REAL)

- znakové řetězce

  CHARACTER(n) (délka *n*, zprava mezery)
  CHARACTER VARYING(n) (max.délka *n*)

1.2 Příklad

```
. . .
CREATE TABLE KINA . . .
. . .
CREATE TABLE PŘEDSTAVENÍ
 (NAZEV_K   Char_Varying(20)          NOT NULL,
  JMENO_F   Char_Varying(20)          NOT NULL,
  DATUM     Date                      NOT NULL,
  PRIMARY KEY (NAZEV_K, JMENO_F),
  FOREIGN KEY (NAZEV_K) REFERENCES KINA,
  FOREIGN KEY (JMENO_F) REFERENCES FILMY);
```

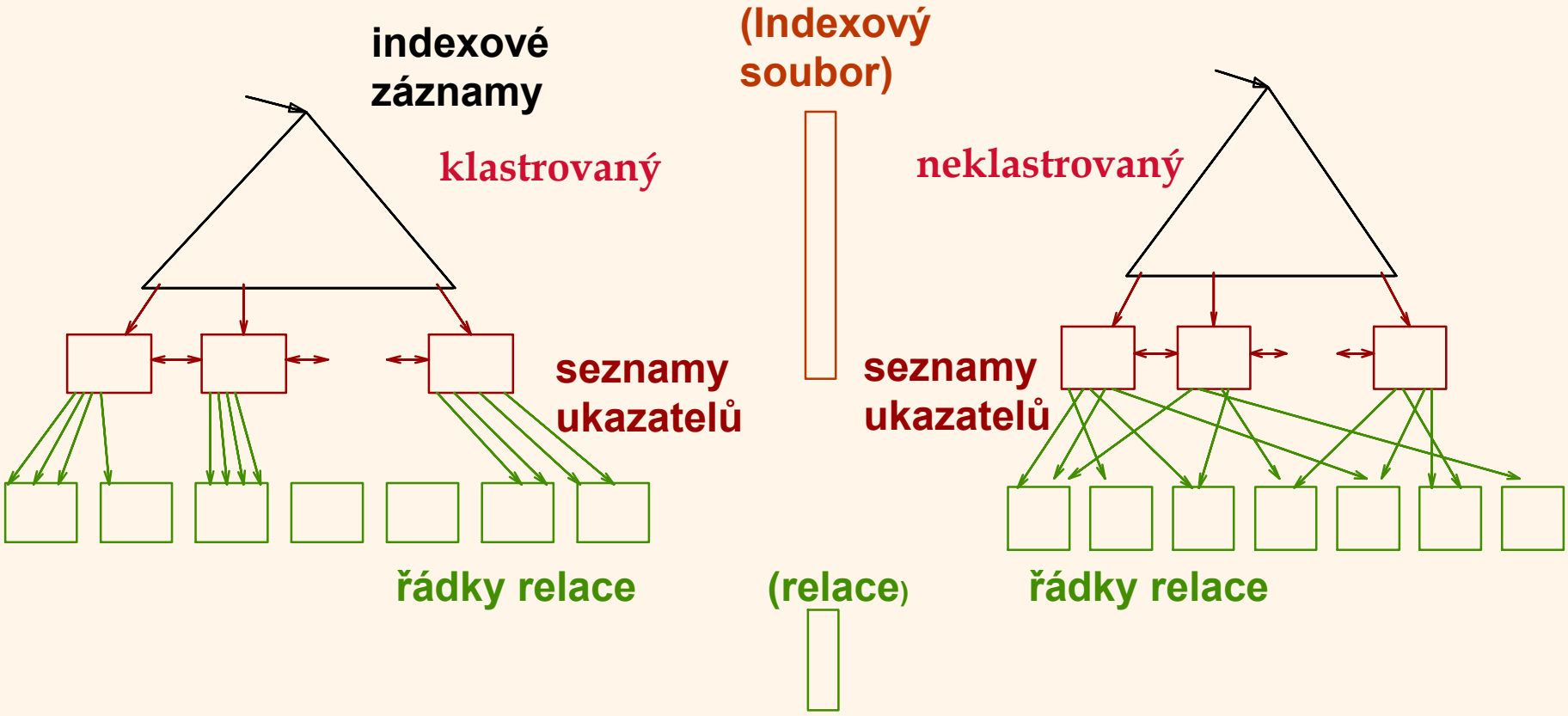
Pz.: Tabulka v SQL nemusí mít primární klíč!

1.3 Indexy v SQL

- nadrelační rys,
- podpora přístupových cest k datům v dotazu
- index obyčejný a typu cluster

```
CREATE INDEX  Idx_zak_jm_adr
                ON ZÁKAZNÍCI (jméno, adresa)
```

Obyčejný vs. klastr



2. Manipulace dat v SQL

```
SELECT [{DISTINCT | ALL}] [{* |  jméno_atr1[, jméno_atr2]... } ]
FROM jméno_rel1[, jméno_rel2]...
[WHERE podmínka]
[ORDER BY specifikace_třídění]
```

Jednoduché dotazy v SQL: za WHERE Boolský výraz, event. nové predikáty.

datum_v BETWEEN ‘2007-04-23' AND ‘2007-05-23’

D1. Vypiš tabulku jmen zákazníků s adresou

```
SELECT jméno, adresa
FROM Zákazníci
```

```
SELECT DISTINCT jméno,
FROM Zákazníci;
ORDER BY jméno ASC;
```

2. Manipulace dat v SQL

Sémantika:

SELECT DISTINCT $A_1,A_2,...,A_j$
FROM $R_1,R_2,...,R_k$
WHERE φ

$\cong$ $(R_1 \times R_2 \times ... \times R_k)(\varphi)[A_1,A_2,...,A_j]$

D2. Najdi dvojice zákazníků, kteří mají stejnou adresu.

```
SELECT X.rod_č AS první, Y.rod_č AS druhý
FROM Zákazníci X, Zákazníci Y
WHERE X.adresa = Y.adresa AND X.rod_č < Y.rod_č;
```

Ve verzi 92: *lokální přejmenování* atributů

2. Manipulace dat v SQL

D3. Vypiš z tabulky Výpůjčky řádky týkající se vrácení do 23.4. 2007.

```
SELECT * FROM Výpůjčky
            WHERE datum_v ≤ '2007-04-23';
```

D4. Najdi režiséry, jejichž některé filmy jsou rezervovány.

```
SELECT DISTINCT režisér
FROM Filmy, Rezervace
WHERE  Filmy.jméno_f = Rezervace.jméno_f;
```

2. Manipulace dat v SQL

Vyhodnocení logických podmínek.

A	B	A and B	A or B	not A
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

sémantika porovnání:

x Θ y = UNKNOWN právě když alespoň jedno z x, y je NULL

Tedy: NULL = NULL se vyhodnocuje jako UNKNOWN

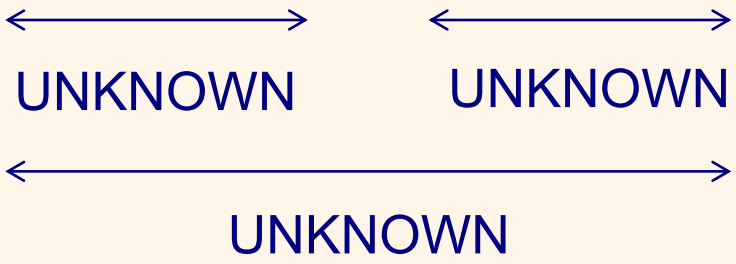
2. Manipulace dat v SQL

Zajímavý příklad:

ZAMĚSTNANCI(OSOBNÍ_Č, ADRESA, JMÉNO, PLAT)

281675	Tachov 21	Novák	NULL

```
SELECT Jméno
FROM Zaměstnanci
WHERE Plat < 29000 OR Plat >= 29000;
```



2.1 Aritmetika

D5. Vypište pro Heinricha Götze čísla kopií, které si vypůjčil, s cenami výpůjček v EUR.

```
SELECT č_kopie, cena/30.15
FROM Výpůjčky X, Zákazníci Y
WHERE Y.jméno = 'Götz H.' AND X.rod_č = Y.rod_č;
```

- operátory /,+, - a *, precedence z běžné praxe, Doporučení: raději důsledně závorkovat
- NULL se propaguje do výsledku, tj. je-li jeden z operandů NULL, je výsledek operace NULL.

2.2 Agregační funkce

agregační_funkce([{ALL|DISTINCT}] jméno_sloupce)

COUNT, SUM, MAX, MIN a AVG.

Aplikují se na dotazem specifikovaný sloupec tabulky,

Výjimka: COUNT(*), počítá prvky včetně jejich duplicit a *prázdných řádků*

- agregační funkce aplikované na sloupce ignorují hodnoty NULL.
- zahrnutí či nezahrnutí duplicit do výsledku se řídí pomocí ALL a DISTINCT.
- ∅ (prázdná tabulka), pak COUNT(∅) = 0.

2.2 Agregační funkce

D6. Kolik je rezervovaných filmů

```
SELECT  COUNT(DISTINCT Jméno_f)   FROM
Rezervace;
```

D7. Najdi počet výpůjček s cenou výpůjčky do 899 Kč.

```
SELECT COUNT(*)
FROM      Výpůjčky
WHERE   cena ≤ 899.00;
```

2.2 Agregační funkce

- SUM a AVG počítají (není-li specifikováno DISTINCT) i s duplicitními hodnotami.
- zahrnutí duplicitních hodnot i explicitně pomocí ALL.
- SUM(∅) = NULL a AVG(∅) = NULL.

D8. Kolik je celkem peněz ve výpůjčkách H. Götze.

```
SELECT SUM(X.cena)
FROM Výpůjčky X, Zákazníci Y
WHERE Y.jméno = ' Götz H.' AND X.rod_č = Y.rod_č;
```

- MIN(∅) = NULL a MAX(∅) = NULL.

2.2 Agregační funkce

```
SELECT [{DISTINCT | ALL}] {*|
hodnotový_výraz1[,hodnotový_výraz2] ...}
```

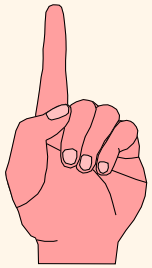
hodnotový výraz - využívá aritmetické výrazy, aplikace agregačních funkcí, hodnoty skalárních poddotazů (vrací právě jednu hodnotu).

Pravidlo: použití agregačních funkcí za SELECT vylučuje použití dalšího sloupce.

D9. Najděte čísla kopií s nejvyšší cenou výpůjčky.

Nesprávně

```
SELECT č_kopie, MAX(cena) FROM Výpůjčky;
```



2.2 Agregační funkce

D9 pomocí *skalárního poddtazu*:

```
SELECT č_kopie, cena
FROM Výpůjčky
WHERE cena = (SELECT MAX(cena) FROM Výpůjčky);
```

D10. Najdi rodná čísla zákazníků, kteří mají půjčeno více než 2 kopie.

```
SELECT rod_č, COUNT(č_kopie) AS počet_kopií
FROM Výpůjčky
GROUP BY rod_č
HAVING 2 < COUNT(č_kopie);
```

Pz.: chceme-li pouze rod_č, není nutné COUNT(č_kopie) za SELECT psát. Starší implementace SQL to často vyžadují.

2.2 Agregační funkce

D11. Vyber kina a jejich adresy, kde mají více jak 8 filmů na programu.

```
SELECT DISTINCT K.název_k, K.adresa
FROM Kina K
WHERE 8 < (SELECT COUNT(jméno_f)
                FROM Představení P
                WHERE P.název_k = K. název_k);
```

Pz.: umístění skalárního poddotazu na obou stranách porovnávacího operátoru Θ je možné.

D12. Najdi průměrnou cenu z minimálních cen kopií pro každého zákazníka.

Ve SQL89 nelze jedním příkazem tento dotaz zapsat.

2.2 Agregační funkce

- víceúrovňová agregace

```
SELECT rod_č, cena, COUNT(č_kopie) AS počet_kopií,
                    (SELECT SUM(V.cena) FROM Výpůjčky V
                    WHERE V.rod_č = rod_č) AS celková_cena
FROM Výpůjčky
GROUP BY rod_č, cena;
```

D13. Ať se pro každého zákazníka k dané ceně objeví počet jeho výpůjček (s tou cenou) a celkový objem peněz za *všechny* jeho výpůjčky.

2.2 Agregační funkce

```
SELECT DISTINCT jméno_v
        FROM Kina K, Zákazníci Z
        WHERE K.jméno_v = Z.jméno AND
                        2000 >  (SELECT SUM (V.cena)
                                 FROM Výpůjčky. V
                                 WHERE V.rod_č = Z.rod_č);
```

D14.

Problém: Je-li počet výpůjček nulový, nedává SUM součet rovný 0, nýbrž NULL, tj vedoucí kin, kteří nemají zapůjčené žádné kopie se nedostanou do odpovědi.

Řešení: konverzí NULL do 0 pomocí funkce COALESCE (viz 2.3).

2.2 Agregační funkce

D15. Pro každého zákazníka a kopii: kolik by byla tržba, kdyby se vypůjčily všechny kopie daného filmu ve stejné ceně?

```
SELECT rod_č, č_kopie,
        cena * (SELECT COUNT(č-kopie) FROM Kopie K
                WHERE K.jméno_f =
                      (SELECT K1.jméno_f  FROM Kopie K1
                       WHERE K1.č_kopie = V.č_kopie)
                       )
    FROM Výpůjčky V
    GROUP BY rod_č, č_kopie;
```

Kde je chyba? cena není za GROUP BY. Jde v TransactSQL.

2.2 Agregační funkce

D15. Pro každého zákazníka a kopii: kolik by byla tržba, kdyby se vypůjčily všechny kopie daného filmu ve stejné ceně?

```
SELECT rod_č, č_kopie,
          cena * (SELECT COUNT(č-kopie) FROM Kopie K
                    WHERE K.jméno_f =
                              (SELECT K1.jméno_f  FROM Kopie K1
                               WHERE K1.č_kopie = V.č_kopie)
                               )
      FROM Výpůjčky V
      GROUP BY rod_č, č_kopie, cena;
```

Kde je chyba? cena není za GROUP BY.

2.3 Hodnotové výrazy

- výrazy CASE

```
CASE
        WHEN pohlaví = 'M' THEN 1
        WHEN pohlaví = 'Ž'  THEN 2
END
```

Lze zadat také ELSE. V příkladě se implicitně bere ELSE NULL, tj. není-li hodnota POHLAVÍ určena, pak na místě hodnoty daného sloupce se v řádku dosadí NULL.

2.3 Hodnotové výrazy

- funkce COALESCE

COALESCE(Výpůjčky.cena, "CENA NENÍ URČENA")

dává v případě, že výpůjční cena kopie je NULL, "CENA NENÍ URČENA", jinak, hodnotu Výpůjčky.CENA.

Obecněji: COALESCE(V1,V2,...,Vn)

vyhodnocuje se zleva doprava a vrací první hodnotu, která není NULL. Neexistuje-li, je výsledek NULL.

2.3 Hodnotové výrazy

- funkce NULLIF

  NULLIF(V1, V2), je ekvivalentní zápisu

  CASE WHEN V1 = V2 THEN NULL ELSE V1 END

D14.(SQL92)

```
SELECT DISTINCT jméno_v
FROM Kina K, Zákazníci Z
WHERE K.jméno_v = Z.jméno AND
          2000 > COALESCE((SELECT SUM
                                   (V.cena)
                    FROM Výpůjčky  V
                    WHERE V.rod_č = Z.rod_č),0);
```

2.4 Predikát LIKE

D16. Najdi platy zaměstnanců, kteří jsou z Kolína. Problém je, že nevíme, zda je v datech ‘Kolin’, nebo ‘Kolín‘.

```
SELECT Z.plat
FROM Zaměstnanci Z
WHERE Z.adresa LIKE '%Kol_n%';
```

_ se shoduje na jakékoliv jednotlivém znaku,
% se shoduje na 0 nebo více znacích.

2.5 Další predikáty SQL92

- řádkové výrazy

  (R.cena, R.datum) > (S.cena, S.datum)

  nahrazuje Boolský výraz

  R.cena > S.cena OR (R.cena = S.cena AND
  R.datum > S.datum)

- predikát MATCH (pro aktualizace tabulek)

  ...WHERE K MATCH (SELECT název_k FROM Kina)

  – lze např. zkontrolovat, zda vstupující hodnoty jsou z dané množiny (referenční integrita).
  – Obecněji: je-li řádek *r* z dané množiny řádků *Q* dané poddotazem

2.5 Další predikáty SQL92

… r MATCH [UNIQUE] [{FULL | PARTIAL}] poddotaz ...}

Sémantika:

- Bez FULL, PARTIAL

  TRUE, je-li nějaká hodnota v *r* rovna NULL,
  není-li žádná NULL a *r* je roven nějakému řádku z *Q* (právě jednomu v případě UNIQUE)

- S FULL

  TRUE, je-li každá hodnota v *r* rovna NULL,
  není-li žádná NULL a r je roven nějakému řádku z Q (právě jednomu v případě UNIQUE)

- S PARTIAL

  TRUE, je-li každá hodnota v *r* rovna NULL,
  každá neprázdná hodnota v *r* je rovna korespondující hodnotě v nějakého řádku z Q (právě jednomu v případě UNIQUE)

2.5 Další predikáty SQL92

UNIQUE poddotaz

- predikát UNIQUE

  testování duplicit

  Jestliže dva řádky jsou si rovny, nabývá predikát hodnoty FALSE. Pro tabulku s prázdnými řádky (označme je $\aleph$) platí UNIQUE($\aleph$) = TRUE.

D17. Vypiš jména a adresy zákazníků, přičemž vždy alespoň dva bydlí na stejné adrese.

```
SELECT Z.jméno, Z.adresa
FROM Zákazníci Z
WHERE NOT UNIQUE( SELECT adresa
                  FROM Zákazníci T
                  WHERE Z.adresa = T.adresa);
```

2.5 Další predikáty SQL92

```
SELECT č_zak
FROM Výpůjčky
WHERE datum_v IS NULL;
```

D18. Vypiš čísla zakázek od výpůjček, které jsou půjčeny neomezeně (chybí hodnota data vrácení).

možnosti: IS NOT NULL,

porovnání s TRUE, FALSE a UNKNOWN.

2.6 Množinové predikáty

- Predikát IN

```
jméno_sloupce [NOT] IN poddotaz
nebo
jméno_sloupce [NOT] IN (seznam_hodnot)
```

D19. Najděte adresy kin, ve kterých dávají film Kolja.

```
SELECT adresa FROM Kina
WHERE název_k  IN     (SELECT  název_k
                       FROM Představení
                       WHERE jméno_f = 'Kolja');
```

- jméno_atributu IN (∅) vrací FALSE
- jméno_atributu IN (ℵ) vrací UNKNOWN

2.6 Množinové predikáty

D20. Najdi filmy, s danými režiséry.

```
SELECT jméno_f  FROM  Filmy
 WHERE Režisér IN (' Menzel ',' Chytilová ',
                                ‘Kachyňa');
```

D21. Najdi jména zákazníků s rezervací filmu od režiséra Menzela.

```
SELECT jméno  FROM Zákazníci  WHERE rod_č IN
  (SELECT rod_č FROM Rezervace R
   WHERE R. jméno_f = (SELECT F.jméno_f FROM Filmy F
                                WHERE F.režisér = ‘Menzel’));
```

2.7. *Predikáty ANY, ALL, SOME*

- >SOME, <SOME, <>SOME (⇔ NOT IN), =SOME (⇔ IN)). ANY je synonymum pro SOME.
- > ALL vyjadřuje: "větší než všechny prvky ze specifikované množiny" (+ další porovnání)
  - jméno_atributu Θ ALL(∅) vrací TRUE,
  - jméno_atributu Θ ALL(ℵ) vrací UNKNOWN,
  - jméno_atributu Θ ANY(∅) vrací FALSE,
  - jméno_atributu Θ ANY(ℵ) vrací UNKNOWN.

2.7. Predikáty ANY, ALL, SOME

D22. Najdi zaměstnance, kteří mají plat vyšší než všichni zaměstnanci z Prahy.

```
SELECT osobní_č, jméno
FROM Zaměstnanci
WHERE plat >  ALL(SELECT Z.plat
                  FROM Zaměstnanci Z
                  WHERE Z.adresa LIKE '%Praha%');
```

2.8 Kvantifikace v SQL

Př. "Pro všechny filmy platí, že mají režiséra".

Logika: univerzální (∀) a existenční (∃) kvantifikátor jsou spolu svázány transformací:

$\forall x\,(p(x)) \cong \neg \exists x\,(\neg p(x))$

Ekvivalentní vyjádření: "Neexistuje film takový, že není pravda, že tento film má režiséra".

Jednodušeji: "Každý film má režiséra " je ekvivalentní tvrzení "Neexistuje film bez režiséra".

- EXISTS

simuluje ∃ (test na neprázdnost množiny)

[NOT] EXISTS poddotaz

2.8 Kvantifikace v SQL

```
SELECT Jméno
FROM Zákazníci Z
WHERE EXISTS (SELECT *  FROM Rezervace
                        WHERE rod_č = Z. rod_č);
```

D23. Najdi jména zákazníků, kteří mají rezervovaný nějaký film.

D23'. Najdi jména zákazníků takových, že existuje film, který mají rezervovaný.

Sémantika:

- výraz se vyhodnotí jako TRUE, je-li množina daná poddotazem neprázdná. V opačném případě nabývá hodnoty FALSE.
- vyhodnocení je podle dvouhodnotové logiky.

2.8 Kvantifikace v SQL

- D23' lze i pomocí IN.
- IN a EXISTS nelze vždy jednoduše navzájem alternovat.

D24. Najdi kina, která nic nehrají.

D24'. Najdi kina taková, že neexistuje film, který by hráli.

```
SELECT název_k
FROM Kina K
WHERE NOT EXISTS
        (SELECT *  FROM Představení P
        WHERE K.název_k = P.název_k);
```

Poznámka: lze i pomocí COUNT

2.9 Množinové operace

```
výraz_dotazu UNION  [ALL] výraz_dotazu [ORDER BY
                                    specifikace_třídění]
```

- UNION
- INTERSECT
- EXCEPT.
  - + složitější výrazy, např. (množinově) $(X \cap Y) \cup Z$, kde X, Y, Z jsou dány poddotazy nebo TABLE T
  - eliminují duplikáty
  - lze simulovat pomocí LEFT OUTER JOIN a testu IS NULL

```
D24.    (SELECT název_k FROM Kina)
        EXCEPT
        (SELECT název_k FROM Představení);
```

2.9 Množinové operace

CORRESPONDING [BY (seznam_sloupců)]

- CORRESPONDING
  - lze zadat přes které společné sloupce se množinová operace provádí
  - bez výčtu sloupců se ve výsledku objeví pouze sloupce společné pro oba operandy.
  - přidáním BY (seznam_sloupců) lze dokonce vybrat jen některé ze společných sloupců.

```
TABLE Zákazníci UNION CORRESPONDING
TABLE Zaměstnanci
```

$\Leftrightarrow$ ZÁKAZNÍCI[JM, ADRESA] $\cup$ ZAMĚSTNANCI[JM, ADRESA]

2.10 Použití NULL – slabiny SQL

```
SELECT osobní_č, jméno
FROM Zaměstnanci
WHERE plat >  ALL(SELECT Z.plat
                  FROM Zaměstnanci Z
                  WHERE Z.adresa LIKE '%Praha%');
```

Je-li ALL(∅), pak > dává TRUE a v odpovědi budou všichni zaměstnanci z tabulky Zaměstnanci.

Alternativa

```
SELECT osobní_č, jméno
FROM Zaměstnanci
WHERE plat > (SELECT MAX (Z.plat)
              FROM Zaměstnanci Z
              WHERE Z.adresa LIKE '%Praha%')
```

MAX(∅) = NULL a > nedá TRUE pro žádnou hodnotu platu. Odpovědi bude ∅.

2.10 Průnik vs. jednoduchá selekce

D25. Kteří zákazníci s bankou jsou v obou tabulkách?

R	
Zákazník	Kód_banky
1	808
2	NULL
NULL	312
NULL	NULL
3	156

S	
Zákazník	Kód_banky
3	156
NULL	808
2	NULL
NULL	NULL

```
SELECT Zákazník, Kód_banky FROM R
INTERSECTION
SELECT Zákazník, Kód_banky FROM S
```

Výsledek	
Zákazník	Kód_banky
3	156
2	NULL
NULL	NULL

2.10 Průnik vs. jednoduchá selekce

R	
Zákazník	Kód_banky
1	808
2	NULL
NULL	312
NULL	NULL
3	156

S	
Zákazník	Kód_banky
3	156
NULL	808
2	NULL
NULL	NULL

```
SELECT R.Zákazník, S.Kód_banky
FROM R, S
WHERE R. Zákazník=S.Zákazník AND
R.Kód_banky= S.Kód_banky
```

Výsledek	
Zákazník	Kód_banky
3	156

2.10 NOT IN vs. NOT EXISTS

D26. Najdi banky, které nemají bankomat na Žižkově

Banky	
Kód_banky	Jméno
156	Reiff
312	KB
808	ČS

Bankomaty		
Bankomat	Čtvrť	Kód_banky
B1	MS	156
B2	Karlín	312
B3	Žižkov	NULL
B4	NULL	312
B5	Smíchov	808

```
SELECT X.Jméno FROM Banky X
WHERE X.Kód_banky NOT IN (SELECT Y.Kód_banky
                          FROM Bankomaty Y
                          WHERE Čtvrť = 'Žižkov')
```

Výsledek
Jméno

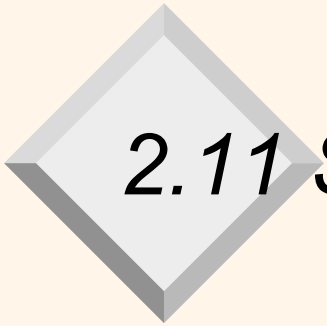
2.10 NOT IN vs. NOT EXISTS

Banky	
Kód_banky	Jméno
156	Reiff
312	KB
808	ČS

Bankomaty		
Bankomat	Čtvrť	Kód_banky
B1	MS	156
B2	Karlín	312
B3	Žižkov	NULL
B4	NULL	312
B5	Smíchov	808

```
SELECT X.Jméno FROM Banky X
WHERE NOT EXISTS (SELECT *
        FROM  Bankomaty Y
                WHERE Čtvrť = 'Žižkov'
                        AND X. Kód_banky = Y. Kód_banky
```

Výsledek
Jméno
Reiff
KB
ČS



2.11 Spojení tabulek

- přirozené spojení,
- spojení křížem,
- spojení přes podmínku,
- spojení přes vyjmenované sloupce,
- vnitřní spojení,
- vnější spojení,
- spojení sjednocením.

2.11 Spojení tabulek

- *přirozené spojení*

```
SELECT *
FROM Filmy NATURAL JOIN Představení;
```

- *spojení křížem*

```
SELECT *
FROM R CROSS JOIN S;
```

- *spojení přes podmínku*

```
SELECT *
FROM R JOIN S ON A≤B;
```

- *spojení přes vyjmenované sloupce*

```
SELECT *
FROM U JOIN V USING (Z, Y);
```

2.11 Spojení tabulek

- *vnitřní spojení*
- *vnější spojení* (LEFT, RIGHT a FULL)

Spojovat lze opět přirozeně nebo s ON.

```
SELECT *
FROM Představení NATURAL RIGHT OUTER JOIN Filmy;
```

obdržíme tabulku, kde budou i ty filmy, které se nikde nedávají.

- *spojení sjednocením*

```
SELECT *
FROM U UNION JOIN V;
```

Každý řádek z levého, resp. pravého operandu je ve výsledku doplněn zprava, resp. zleva hodnotami NULL.

Není podporován od SQL:2003!

2.11 Spojení tabulek

Za FROM mohou být odvozené tabulky zadané pomocí SELECT (⇔ CROSS JOIN)

```
D12. (SQL)
        SELECT AVG(T.minim_c)
        FROM (SELECT  MIN(cena)
                FROM Výpůjčky
                GROUP BY rod_č) AS T(minim_c);
```

Výraz dotazu je kolekce termů spojených pomocí UNION, INTERSECT, EXCEPT. Každý term je buď specifikací dotazu (SELECT) nebo konstantní řádek či tabulka dané příslušnými konstruktory.

3. Aktualizace v SQL

```
DELETE FROM Filmy
WHERE jméno_f = ‘Puška’;
```

Co se bude dít, má-li film kopie, nebo je rezervován?

```
UPDATE Zákazníci  SET jméno = ‘Götzová’
                        WHERE rod_č = '4655292130';
```

```
UPDATE Zákazníci SET jméno = ‘Müller’
                        WHERE jméno = ‘Muller’;
```

```
ALTER TABLE Zákazníci
Add Počet_půjček  Number;

UPDATE Zákazníci Z
SET Počet_půjček = (SELECT count(*) from Výpůjčky V
                        WHERE V.rod_č = Z. rod_č);
```

3. Aktualizace v SQL

sloupec adresa bude mít hodnotu default, nebo NULL

Co se stane při pokusu vložit již zapsané rod_č?

```
INSERT INTO Zákazníci (rod_č, jméno)
VALUES ('4804230160',Novák');
```

```
CREATE TABLE Kolik_kopií (rod_č CHAR(10),
                          počet SMALLINT);
INSERT INTO TABLE Kolik_kopií
        SELECT rod_č, COUNT(č_kopie) FROM Výpůjčky
        GROUP BY rod_č;
```

```
CREATE TABLE Kolik_kopií (rod_č CHAR(10),
                          počet SMALLINT)
AS      SELECT rod_č, COUNT(č_kopie) FROM Výpůjčky
        GROUP BY rod_č;
```

4. Pohledy

chceme-li aktualizaci

```
CREATE VIEW jméno-pohledu [(v-jméno-atr1[,v-jméno-atr2]...)]
AS dotaz
[WITH [{CASCADE | LOCAL} CHECK OPTION]
```

```
CREATE VIEW Pražáci AS  SELECT č_zak, jméno, adresa
FROM Zákazníci WHERE  adresa LIKE '%PRAHA%';
```

```
DROP VIEW Pražáci;
```

```
CREATE VIEW Kolik_kopií (rod_č, počet_půjček) AS
        SELECT rod_č, COUNT(č-kopie) FROM Výpůčky
        GROUP BY rod_č;
```

4. Pohledy

- pohled nelze indexovat

Aktualizace pohledu vede na aktualizaci základní tabulky, na které je pohled založen.

- pohled daný spojením více tabulek nebývá aktualizovatelný,
- pohled nad jednou tabulkou je neaktualizovatelný,
  - obsahuje-li sloupec s odvozenou hodnotu,
  - odstiňuje-li projekcí sloupec, na který je uvaleno NOT NULL omezení (zejména PRIMARY KEY)

4. Pohledy

- u pohledu, jehož definice obsahuje selekci, je nutné reagovat na pokus o aktualizaci, která je v rozporu s definicí pohledu, např.

  INSERT INTO Pražáci VALUES
  (1234, ’Novák Jiří’,’Pražská 3, Kolín 5’)
- klauzule WITH CHECK OPTION nabádá DB stroj, aby odmítl takovou aktualizaci

  CASCADE/LOCAL - určuje hloubku kontroly

4. Pohledy - použití

- utajení dat (lze předložit je některé sloupce a řádky)
- ukrytí složitosti (složitý dotaz skrytý v definici pohledu je navržen pouze jednou)
- optimalizace (např. je-li použito hledání společných podvýrazů)

4. Pohledy - materializace

- Materializované pohledy nejsou virtuální, ale skutečné tabulky.
- Mohou být automaticky udržované (inkrementálně nebo přepočítáním celé tabulky pohledu)
- Podpora: Oracle, DB2

5. Integritní omezení

- CREATE DOMAIN

```
CREATE DOMAIN
    LETOS IS DATE DEFAULT ‘2001-12-31’
    CHECK (VALUE >= ‘2001-01-01’ AND VALUE <= ‘2001-12-31’)
    NOT NULL;
CREATE TABLE VÝPŮJČKY
(č_kopie CHAR(3) UNIQUE NOT NULL,
č_zak CHARACTER(6) NOT NULL,
cena DECIMAL(5,2) CHECK (cena >= 100),
rod_č CHARACTER(10) NOT NULL,
datum_v LETOS)
PRIMARY KEY (č_zak);
```

5. Integritní omezení

```
cena DECIMAL(5,2)
        CONSTRAINT VĚTŠÍ100 CHECK (cena >= 100)
```

– pojmenovaná IO, odkazy na jiné sloupce, tabulky

IO: "Nebude se dávat žádný film režírovaný Troškou"

pro atribut JMÉNO_F v tabulce PŘEDSTAVENÍ.

```
CHECK (jméno_f <> ANY (SELECT jméno_f FROM FILMY
                       WHERE režisér = 'Troška') )
```

– IO tabulek

```
CONSTRAINT Troška_ne ...
```

5. Integritní omezení

Problém: IO tabulek jsou splněna i pro ∅.

IO: „Vždy se dává nějaký film“.

```
CONSTRAINT PŘEDSTAVENÍ_VŽDY
        CHECK (SELECT COUNT(*) FROM PŘEDSTAVENÍ) > 0
```

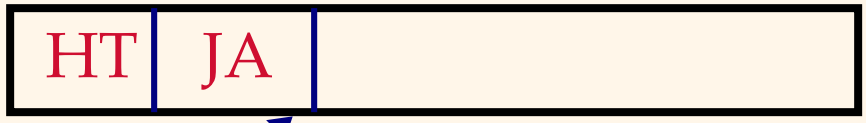
- tvrzení - jsou definována mimo tabulky

- CREATE ASSERTION

pojmenované IO formulované za svým jménem pomocí CHECK, nenabývá automaticky TRUE na prázdné tabulce!

5.2 Referenční integrita

hlavní tabulka (HT)
master
parent



závislá tabulka (ZT)
detail
child

CK cizí klíč, hodnota může být NULL,
doména je dána aktuální doménou jednoznačného atributu JA (např. primární klíč či UNIQUE NOT NULL)

Pz.: prázdné hodnoty souvisí m.j. s potřebou vyjádřit nepovinné členství s kardinalitou 1:M v E-R modelu

Pokus narušit ref. integritu vedl podle SQL89 pouze k hlášení chyby.

5.2 Referenční integrita

- v definici IO sloupce
- v definici IO tabulky

```
FOREIGN KEY (č_kopie) REFERENCES Kopie,
FOREIGN KEY (rod_č) REFERENCES Zákazníci)
```

- operační chování

DELETE (řádku z hlavní tabulky)

- kaskádové odstranění řádků (ON DELETE CASCADE)
- nahrazení cizího klíče prázdnou hodnotou (SET NULL)
- nahrazení cizího klíče implicitní hodnotou (SET DEFAULT)
- neodstranění řádku s upozorněním (NO ACTION)

Zadání: ON DELETE akce, nebo ON UPDATE akce

5.2 Příklad

```
. . .
DROP TABLE KINA CASCADE CONSTRAINTS;
CREATE TABLE KINA . . .
ON DELETE CASCADE

CREATE TABLE PŘEDSTAVENÍ
 (NAZEV_K   Char_Varying(20)          NOT NULL,
  JMENO_F   Char_Varying(20)          NOT NULL,
  DATUM     Date                      NOT NULL,
  PRIMARY KEY (NAZEV_K, JMENO_F),
  FOREIGN KEY (NAZEV_K) REFERENCES KINA,
  FOREIGN KEY (JMENO_F) REFERENCES FILMY);
```

5.2 Definice tabulky - shrnutí

CREATE TABLE *jméno_tabulky* (
{ *název_sloupce datový_typ* [NOT NULL] [UNIQUE]
[DEFAULT *hodnota*] [CHECK **(** *výběrová_podmínka* **)**
[**,** *název_sloupce …*]}
[PRIMARY KEY **(** *seznam_názvů_sloupců* **),**]
{ **[** FOREIGN KEY **(** *seznam_názvů_sloupců_tvořící_cizí_klíč* **)**
REFERENCES *název_nadřazené_tabulky* [(*seznam_názvů_sloupců*)] ,
[MATCH { PARTIAL | FULL }]
[ON UPDATE referenční akce]
[ON DELETE referenční akce]]
[, …] }
{ [CHECK (výběrová_podmínka) [, …] }
)

5.3 Další možnosti IO

WITH CHECK OPTION poskytuje další možnost pro vyjádření IO nad základní tabulkou pohledu.

```
CREATE VIEW v_kopie
AS
SELECT * FROM Kopie K
WHERE K.Jmeno_f IN (SELECT Jmeno_f FROM Film)
WITH CHECK OPTION
```

Pohled vyjadřuje referenční integritu a může být alternativou k jejímu deklarativnímu vyjádření u strojů SQL, které ji nepodporují

6. Systémový katalog

Př.: ORACLE

Tab(TName,TabType, ClusterID)

- jméno tabulky (relace nebo pohledu)
- typ tabulky (relace nebo pohled)
- ve kterém klastru je tabulka uložena

SysCatalog … více informací o tabulkách

SysColumns(CName, TName, Creator, ColNo, ColType,…)

SysUserlist(userId, UserName, TimeStamp,...

6. Systémový katalog

SysIndexes(IName, ICreator, TName, Creator, .)

SysViews(ViewName, VCreator, …)

Dotazy nad katalogem pomocí SQL

SELECT * FROM Tab

7. Ochrana dat

Příklady:

- ALTER
- DELETE
- EXECUTE
- INDEX
- INSERT
- REFERENCES
- SELECT
- UPDATE

Danému uživateli/uživatelské roli lze přidělit právo provádět dané akce nad daným objektem

```
REVOKE ALL PRIVILEGES ON
Filmy FROM PUBLIC;
```

- odstraň práva přístupu
- PUBLIC odkazuje k implicitně definované skupině rolí

```
GRANT ALL PRIVILEGES ON
V_filmy TO PUBLIC;
```

8. Standardizace SQL

SEQUEL: vývoj od počátku 70. let

Standardizace:

- SQL86
- SQL89 (opravy, úpravy SQL86)
- SQL92
  - vstupní (drobné úpravy SQL89)
  - prostřední (asi polovina všech funkcí)
  - úplný

8. Standardizace SQL

- SQL99 (objektové rozšíření, rekurze, triggery, …)
  - Všechny rysy jazyka jsou očíslovány a označeny za povinné nebo výběrové.
  - Systémy vyhovující standardu musí splňovat všechny povinné vlastnosti.
- SQL:2003
  - Objevuje se něco z XML.
  - Dokončeno pět částí SQL/MM (Multimedia and Application Packages)

8. Standardizace SQL

- SQL:2006
  - úplná integrace XML do SQL včetně XQuery
- SQL/MM (Multimedia and Application Packages)
  - Část 1: Základy (Framework),
  - Část 2: Úplné texty (Full Text),
  - Část 3: Prostorové objekty (Spatial),
  - Část 5: Nepohyblivé obrazy (Still Image)
  - Část 6: Dolování dat
  - Část 7: Historie (draft z r. 2011), nyní ISO/IEC TS 13249-7
  - Část 8: Registr metadat (draft z r. 2011), nyní ISO/IEC 11179

8. Standardizace SQL

- SQL:2006
  - úplná integrace XML do SQL včetně XQuery
- SQL/MM (Multimedia and Application Packages)
  - Část 1: Základy (Framework),
  - Část 2: Úplné texty (Full Text),
  - Část 3: Prostorové objekty (Spatial),
  - Část 5: Nepohyblivé obrazy (Still Image)
  - Část 6: Dolování dat
  - Část 7: Historie (draft z r. 2011), nyní ISO/IEC TS 13249-7
  - Část 8: Registr metadat (draft z r. 2011), nyní ISO/IEC 11179

8. Standardizace SQL

- SQL:2008
  - část 1: Framework (SQL/Framework)
  - část 2: Foundation (SQL/Foundation) 1100 s.
  - část 3: Call-Level Interface (SQL/CLI*)
  - část 4: Persistent Stored Modules (SQL/PSM**)
  - část 9: Management of External Data (SQL/MED)
  - část 10: Object Language Bindings (SQL/OLB)
  - část 11: Information and Definition Schemas (SQL/Schemata)
  - část 13: SQL Routines and Types Using the Java TM PL (SQL/JRT)
  - část 14: XML-Related Specifications (SQL/XML)

* alternativa k volání SQL z aplikačních programů (implementace: ODBC)

** procedurální jazyk pro psaní transakcí (alternativy: IBM: SQL PL, Microsoft/Sybase: T- SQL, MySQL: MySQL, Oracle: PL/SQL, PostgreSQL: PL/pgSQL

8. Standardizace SQL

- Části 5, 6, 8 neexistují

Dočasně pozastaveno:

- část 7 – SQL/Temporal (částečně implementován v ORACLE 11g, IBM DB2 pro z/OS, Teradata 13.10)

Zrušeno:

- část 12 – SQL/Replication

- SQL:2011
  - příkaz pro „vypnutí“ IO,
  - obsahuje podporu temporálních databází – liší se od přístupu zrušené části 7

8. Standardizace SQL

- SQL:2016 (má více než 4300 stran)
  - rozpoznání vzorů řádků – vzor je zadán regulárním výrazem (vhodné pro hledání vzorů v časových řadách)
  - podpora typu JSON (nikoliv nativně – viz XML, ale využívá řetězce znaků)
  - polymorfické funkce
- SQL:2019
  - multimediální pole (typ MDarray a odpovídající operátory)

Standardizační organizace:

- ANSI a ISO (International Organization of Standardization, ale též z řečtiny „stejný“ (isos - ίδιος))

9. Závěr

- SQL je především jazykem komunikace
- aplikovatelnost vs. monstrózní rozsah