Dotazovací jazyk SQL

Historický vývoj I

- IBM - 70. léta - prototyp relačního DBMS - System R
- 80. léta - základ 2 komerčních DBMS: SQL/DS, DB2

SQL jako standard

- ***Standardizační instituce***
  - ANSI: American National Standards Institute
  - ISO: International Organization for Standardization
  - IEC: International Electrotechnical Commission

- ***SQL86 (někdy též SQL87) - SQL 1***
  - 1986 ANSI X3.135-1986 *Database language SQL*
  - 1987 ISO 9075-1987 *Database language SQL*

  Stále nebyla standardizována referenční integrita, ta přišla až
  - 1989 ANSI X3.135-1989 *Database Language SQL With Integrity Enhancement*
  - 1989 ISO 9075-1989 *Database language SQL*

Historický vývoj II

- ***Embedded SQL***
  - 1989 ANSI X3.168-1989 Database Language Embedded SQL
  - neexistuje ISO standard pro embedde SQL

  Embedded SQL umožňuje klást SQL dotazy z prostředí hostitelského programovacího jazyka, typicky jazyka C

  Umožňuje zapisovat dotazy přímo do zdrojového kódu v jazyce C.

  Speciální preprocesor pak tyto dotazy nahradí voláním příslušných knihovních procedur.

  Počátkem 90. let hojně využíváno při vývoji databázových aplikací v jazyce C.

Historický vývoj III

- ***SQL92 – známý jako SQL2:***
  - 1992 ANSI X3.135-1992 *Database language SQL*
  - 1992 ISO/IEC 9075-1992 *Database language SQL*

  - Dnes nejběžněji používaný standard jazyka SQL
  - Hlavně tomuto standardu se budeme na přednáškách věnovat

Další vývoj

- ***SQL99 – známý jako SQL3***
  - Regulární výrazy
  - Rekurzivní dotazy
  - Non-scalar datové typy
  - a další
- ***2003: SQL2003***
  - Podpora XML
  - Standardizované sekvence a automaticky generované hodnoty
  - *http://www.wiscorp.com/sql_2003_standard.zip*
- ***2006: SQL2006***
  - Rozšířená podpora XML (XQuery)
  - Převody XML ↔ SQL
  - Export/import XML

SQL – datové typy I

Numerické datové typy ukládané přesně

INTEGER	Celá čísla	Rozsah implementačně závislý, obvykle -2147483648 až 214783647
SMALLINT	Celá čísla	Rozsah implementačně závislý, obvykle -32768 až 32767. Definice: rozsah není větší než u typu INTEGER.
NUMERIC **NUMERIC(p)** **NUMERIC(p,s)**	Číslo - může mít desetinnou část, ale uložené přesně. Dekadické číslo o *p* cifrách, z toho *s* za desetinnou čárkou. Např. DECIMAL(5,2) má 3 pozice před čárkou, 2 pozice za čárkou. Pokud *p* nebo *s* nevyjádřeno, vezme se implementačně závislý default. Číslo vždy uloženo v předepsané přesnosti (ledaže by se narazilo na implementačně dané maximum).	
DECIMAL **DECIMAL(p)** **DECIMAL(p,s)**	Obdoba jako NUMERIC, avšak číslo může být ukládáno přesněji než je požadavek (ledaže by se narazilo na implementačně dané maximum).	

Poznámka: NUMERIC zřejmě vždy ukládáno jako řetězec cifer - znaků, zatímco DECIMAL ve vnitřním formátu, pro repúrezentaci se použije tolik bytů, aby bylo dosaženo (alespoň) požadované přesnosti.

SQL – datové typy II

Numerické datové typy ukládané nepřesně

REAL	plovoucí řádová čárka jednoduchá přesnost	přesnost implementačně závislá, obvykle defaultní přesnost pro data s plovoucí řádovou čárku v jednoduché přesnosti pro danou hardwarovou platformu
DOUBLE PRECISON	plovoucí řádová čárka v dvojnás. přesnosti	přesnost implementačně závislá, obvykle defaultní přesnost pro data s plovoucí řádovou čárku v dvojnásobné přesnosti pro danou hardwarovou platformu. Definice: přesnost větší než u typu REAL.
FLOAT **FLOAT(p)**	Dovoluje definovat požadovanou přesnost. Přesnost uložení může být větší než požadovaná. Požadovaná přesnost *p* může být na jedné platformě realizována jednoduchou přesností, zatímco na jiné platformě dvojnásobnou přesností.	

Poznámka: Nepřesnost uložení je dána tím, že se jedná o plovoucí řádovou čárku. Číslo je vyjádřeno dvojicí <*mantisa, exponent*>

SQL – datové typy III

Znakové řetězce

CHARACTER **CHAR**ACTER**(x)**	Znakový řetězec o definované délce. Není-li x vyjádřeno, jedná se o CHAR(1).
CHARACTER **VAR**YING **VARCHAR** **CHAR**ACTER **VAR**YING**(x)** **VARCHAR(x)**	Znakové řetězce s proměnnou délkou (alokováno tolik byte, kolik je potřeba) – maximální délka stringu (počet znaků v řetězci) omezena číslem *x*. Maximální hodnota *x* stringu implementačně závislá.
NATIONAL **CHAR**ACTER **NCHAR** **NVARCHAR**	Pro potřeby národních abeced. UNICODE

SQL – datové typy IV

Datum a čas

DATE	Datum: rok 4 cifry, měsíc 2 cifry, den 2 cifry Délka definována na 10 znaků včetně oddělovačů **YYYY-MM-DD**
TIME **TIME(p)**	Čas: hodiny(2 cifry), minuty(2 cifry), vteřiny (2 cifry, navíc možno p desetinných míst) Délka 8 cifer pokud p=0, jinak 9+p Default 0 desetinných míst **HH:MM:SS**
TIMESTAMP **TIMESTAMP(p)**	Datum + čas. Délka 19 cifer pokud p=0, jinak 20+p Default 6 desetinných míst **YYYY-MM-DD HH:MM:SS**

CREATE TABLE I

CREATE TABLE *PACKAGE*
(*PACKID* **CHAR(***4***),**
PACKNAME **CHAR(***20***),**
PACKVER **DECIMAL(***3,2***),**
PACKTYPE **CHAR(***15***),**
PACKCOST **DECIMAL(***5,2***))**

Jméno tabulky, která má být vytvořena

Jmeno atributu

Typ atributu

Vytvoří prázdnou tabulku s 5 definovanými sloupci příslušných typů.

DROP TABLE *Computer*

Jméno tabulky, která má být zrušena

Zruší existující tabulku daného jména.

CREATE TABLE II (integritní omezení atributu)

```
CREATE TABLE COMPUTER
    (   COMPID      DECIMAL(2)  NOT NULL,
        MFGNAME     CHAR(15)    NOT NULL,
        MFGMODEL    CHAR(25),
        PROCTYPE    DECIMAL(7,2)    )
```

Integritní omezení (atributu), specifikující, že daný atribut musí mít povinně vyplněnou hodnotu

Vkládáme-li do tabulky nový řádek, nemusíme v obecném případě specifikovat hodnoty všech atributů (sloupců). Takový řádek pak bude mít ve sloupcích, pro něž jsme nezadali hodnotu, hodnotu uvedenu NULL.

Pokud ovšem při vkládání řádku do tabulky neuvedeme hodnotu takového atributu, který má specifikováno integritní omezení NOT NULL, databázový engine odmítne takový řádek do tabulky vložit (chybová hláška nebo výjimka), protože by došlo k porušení příslušného integritního omezení.

CREATE TABLE III (integritní omezení atributu)

```
CREATE TABLE Films
    (   CODE CHAR(5)    CONSTRAINT Firstkey  PRIMARY KEY
        TITLE           VARCHAR(40)  NOT NULL,
        DateProd        DATE,
        KIND            VARCHAR(10),
        LEN             INTERVAL HOUR TO MINUTE
```

Integritní omezení může být (ale nemusí a obvykle nebývá) pojmenováno. Šedivý text tedy může být vynechán.

Toto integritní omezení říká, že atribut CODE je primárním klíčem. Musí mít tudíž povinně zadanou hodnotu a tato hodnota musí být unikátní přes všechny řádky dané tabulky.

CREATE TABLE IV (integritní omezení tabulky)

```
CREATE TABLE Films
    (   TITLE          VARCHAR(40)  NOT NULL,
        DateProd       DATE,
        KIND           VARCHAR(10),
        LEN            INTERVAL HOUR TO MINUTE  ),
        CONSTRAINT pk_const PRIMARY KEY (TITLE, DateProd)
    )
```

Integritní omezení tabulky

Nepovinné jméno integritního omezení tabulky. Integritní omezení je vhodné pojmenovávat, abychom je mohli popřípadě odtsranit, pokud nevyhovují:

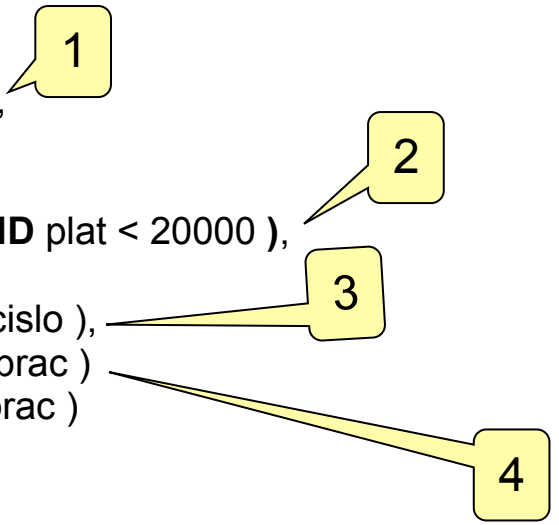
ALTER TABLE *Films* **DROP CONSTRAINT** pk_const;

V případě, že je prim
skutečnost vyjádřit in
na primárním klíči tot
primárním klíčem dvojice atributů (*TITLE*, *DateProd*), což vyjádříme uvedeným integritním omezením tabulky. Unikátní přes všechny řádky nemá být hodnota každého z atributů *TITLE, DateProd*, ale jejich kombinace.

PRIMARY KEY je jedním z možných integritních omezení tabulky.

CREATE TABLE V (integritní omezení)

```
CREATE TABLE osoby (
  os_cislo    NUMBER(5)    NOT NULL,
  rod_cis     VARCHAR2(30) NOT NULL UNIQUE,
  jmeno       VARCHAR2(30) NOT NULL,
  prijmeni    VARCHAR2(30) NOT NULL,
  plat        NUMBER(5)    CHECK ( plat > 5000 AND plat < 20000 ),
  cislo_prac  NUMBER(5)    NOT NULL,
  CONSTRAINT pk_osoby PRIMARY KEY ( os_cislo ),
  CONSTRAINT fk_prac  FOREIGN KEY ( cislo_prac )
                      REFERENCES pracoviste ( cislo_prac )
                      ON UPDATE CASCADE
                      ON DELETE CASCADE
)
```



1. Atribut může mít zadáno více integritních omezení současně – v tomto případě je hodnota atributu povinná (NOT NULL) a unikátní (UNIQUE) přes všechny řádky.
2. Integritní omezení může být zadáno i obecnou podmínkou, která musí být pro vkládaný řádek TRUE, jinak chyba.
3. Libovolné integritní omezení atributu může být rovněž vyjádřeno jako integritní omezení tabulky. V tomto případě jsme mohli skutečnost, že os_cislo je primárním klíčem, rovnocenně vyjádřit integritním omezením atributu rod_cis.
4. Toto je tzv. referenční integrita – bude probrána na samostatném slajdu.

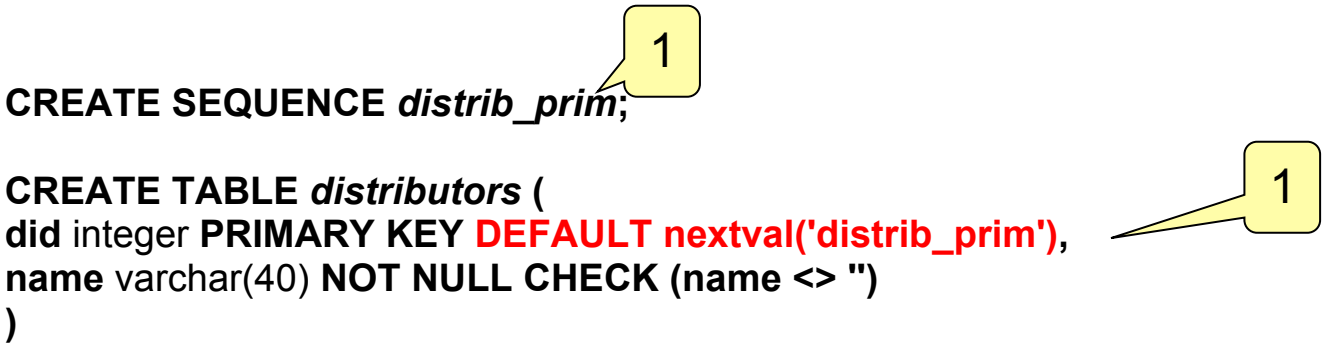
CREATE TABLE VI (integritní omezení)

```
CREATE TABLE PREDMETY (
  zkratka       VARCHAR2(10) PRIMARY KEY,
  nazev         VARCHAR2(30) NOT NULL,
  kredity       NUMBER(2) DEFAULT 2,
);
```

Pomocí svého druhu integritního omezení můžeme definovat i defaultní hodnotu atributu.

Budeme-li vkládat řádek do tabulky vytvořené výše uvedeným příkazem a neuvedeme-li přitom hodnotu sloupce kredity, nezústane tento sloupec nevyplněn (NULL), ale bude mít hodnotu 2.

CREATE TABLE VII (generování hodnot)



1. Nejprve definujeme tzv. sekvenci. V daném případě jsme ji pojmenovali *distrib_prim*.
2. Při vkládání nového řádku bude chtít integritní omezení DEFAULT přiřadit sloupci *did* vkládaného řádku hodnotu. Tuto hodnotu zjistí vyhodnocením funkce nextval(), jež ovšem vygeneruje nový (ještě neexistující) prvek sekvence *distrib_prim*. Jako výsledek bude mít každý řádek vygenerovanou unikátní hodnotu sloupce *did*.

Toto není SQL standard, ale syntax DB systému ProgreSQL. Generování hodnot bylo standardizováno až v SQL2006.

REFERENČNÍ INTEGRITA I

```
CREATE TABLE osoby (
  os_cislo   NUMBER(5)    PRIMARY KEY,
  rod_cis    VARCHAR2(30) NOT NULL UNIQUE,
  cislo_prac NUMBER(5)    NOT NULL,
  CONSTRAINT fk_prac  FOREIGN KEY ( cislo_prac )
                REFERENCES pracoviste ( cislo_prac )
                ON UPDATE CASCADE
                ON DELETE CASCADE
)
```

1. Integritní omezení *fk_prac* říká, že atribut *cislo_prac* je cizím klíčem, jehož hodnota odkazuje na takový řádek tabulky *pracoviste,* jehož hodnota primárního klíče se shoduje s hodnotou atributu *cislo_prac.*
2. Znamená to tedy, že řádky reprezentující všechny osoby z daného pracoviště se prostřednictvím atributu *cislo_prac* odkazují na stejnou řádku tabulky *pracoviste*, jež odpovídá jejich společnému pracovišti.
3. Co se stane, když někdo změní hodnotu primárního klíče jejich společného pracoviště? To řeší sekce *ON UPDATE ...* V žádném případě by nemělo dojít k tomu, že nějaký řádek tabulky *osoby* bude odkazovat na neexistující řádek tabulky *pracoviste*. V daném případě je specifikováno *ON UPDATE* ***CASCADE*****.** To znamená, že příkaz modifikující hodnotu primárního klíče bude proveden, ale současně budou změněny příslušným způsobem hodnoty cizího klíče (atributu *cislo_prac*) u všech řádků tabulky *osoby*, jež reprezentují osoby zařazené na pracoviště, jehož primární klíč jsme změnili.

REFERENČNÍ INTEGRITA II

```
CREATE TABLE osoby (
  os_cislo   NUMBER(5)    PRIMARY KEY,
  rod_cis    VARCHAR2(30) NOT NULL UNIQUE,
  cislo_prac NUMBER(5)    NOT NULL,
  CONSTRAINT fk_prac  FOREIGN KEY ( cislo_prac )
                REFERENCES pracoviste ( cislo_prac )
                ON UPDATE CASCADE
                ON DELETE CASCADE
)
```

4. Co se stane, když někdo zruší řádek tabulky pracoviště, který reprezentuje jejich společné pracoviště? To řeší sekce *ON DELETE ...* Opět by v žádném případě nemělo dojít k tomu, že nějaký řádek tabulky *osoby* bude odkazovat na neexistující řádek tabulky *pracoviste*.
V daném případě je specifikováno *ON DELETE* ***CASCADE*****.** To znamená, že příkaz rušící jejich společné pracoviště bude proveden, ale současně budou smazány i všechny řádky tabulky *osoby*, jež odpovídaly odsobám pracujícím na zrušeném pracovišti.

REFERENČNÍ INTEGRITA III

```
CREATE TABLE osoby (
  os_cislo   NUMBER(5)    PRIMARY KEY,
  rod_cis    VARCHAR2(30) NOT NULL UNIQUE,
  cislo_prac NUMBER(5)    NOT NULL,
  CONSTRAINT fk_prac  FOREIGN KEY ( cislo_prac )
                REFERENCES pracoviste ( cislo_prac )
                ON UPDATE CASCADE
                ON DELETE CASCADE
)
```

Jaké jsou jiné možnosti než ***CASCADE***?

1. RESTRICT – změna, která by porušila referenční integritu se neprovede. DB systém odmítne změnu primárního klíče (ON UPDATE) nebo rušení řádku (ON DELETE) provést – chybová hláška, výjimka.
2. SET NULL – změna se provede, ale řádky tabulky *osoby* odkazující svým cizím klíčem na modifikovaný (změna primárního klíče nebo zrušení řádku) řádek tabulky *pracoviste* dostanou ve sloupci *cislo_prac* hodnotu NULL (Nebudou tedy odkazovat na neexistující řádek).
3. SET DEFAULT – obdoba jako SET NULL určená pro případ, že cizí klíč má definovanou defaultní hodnotu.

Modifikátory *CASCADE, RSTRICT, SET NULL, SET DEFAULT* se v sekcích *ON UPDATE* a *ON DELETE* nastavují nezávisle.

SELECT VIII

Vestavěné (BUILT-IN) agregační funkce

COUNT(*sloupec*) **COUNT(*)**	Počet řádků vyhovujících podmínce WHERE. Protože výsledek nezávisí na jménu sloupce uvedeného v argumentu, lze místo sloupce uvést znak *.
COUNT(DISTINCT *sloupec*)	Počet **různých** hodnot uvedeného sloupce vyskytujících se ve všech řádcích vyhovujících podmínce WHERE.
SUM(*sloupec*)	Součet hodnot ve sloupci přes všechny řádky vyhovující podmínce WHERE. Sloupec musí být některého numericko typu.
AVG(*sloupec*)	Průměr hodnot ve sloupci přes všechny řádky splňující WHERE. Sloupec musí být některého numerického typu.
MAX(*sloupec*)	Největší hodnota ve sloupci přes všechny řádky splňující WHERE. U numerických číselně, u stringu podle abecedy, ...
MIN(*sloupec*)	Nejmenší hodnota ve sloupci přes všechny řádky splňující WHERE. U numerických číselně, u stringu podle abecedy, ...

SELECT IX

Tabulka PACKAGE

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

Count:

Počet řádků nezávisí na atributu uvedeném v argumentu funkce COUNT. Stejného výsledku tedy dosáhneme, uvedeme-li na místě argumentu jméno libovolného sloupce nebo znak '*'.

```
SELECT COUNT(*)
  FROM PACKAGE
  WHERE PACKTYPE = 'Database'
```

```
SELECT COUNT(PACKID)
  FROM PACKAGE
  WHERE PACKTYPE = 'Database'
```

Výsledek:

Název sloupce zvolil databázový systém sám.

COUNT1
2

SELECT X

Tabulka PACKAGE

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

SELECT COUNT(DISTINCT *PACKNAME* **)**
FROM *PACKAGE*
WHERE *PACKTYPE* **=** 'Database'

Výsledek:

COUNT1
1

Podmínce WHERE vyhovují dva řádky. Oba však mají atribut *PACKNAME* roven řetězci ‘Manta’. Výsledkem je tudíž číslo 1, neboť ve všech řádcích vyhovujících podmínce WHERE je pouze jedna rozdílná hodnota.

SELECT XI

Tabulka PACKAGE

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

SELECT COUNT(*PACKID* **), SUM(** *PACKCOST* **)**
FROM *PACKAGE*

Výsledek:

COUNT1	SUM2
6	1968.96

SELECT XII

Tabulka PACKAGE

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

SELECT COUNT*(PACKID* **), AVG(** *PACKCOST* **)**
FROM *PACKAGE*

Výsledek:

COUNT1	AVG2
6	**328.16**

SELECT XIII

Tabulka PACKAGE

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

SELECT COUNT(*PACKID* **), MAX(** *PACKCOST* **)**
FROM *PACKAGE*

Výsledek:

COUNT1	MAX2
6	725.83

SELECT XIV

Tabulka PACKAGE

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

SELECT COUNT(*PACKID* **), MIN(** *PACKCOST* **)**
FROM *PACKAGE*

Výsledek:

COUNT1	MIN2
6	30.00

Poznámka:

1. Věty s NULL value v příslušném sloupci jsou u SUM, AVG, MAX, MIN ignorovány
2. Může nastat situace, kdy COUNT(*) a COUNT(atribut) vrátí rozdílné hodnoty a to tehdy, když pro některé věty má atribut **atribut** nepřiřazené hodnoty (NULL).

SELECT XV

Tabulka PC

TAGNUM	COMPID	EMPNUM	LOCATION
32808	M759	611	Accounting
37691	B121	124	Sales
57772	C007	567 I	Info Systems
59836	B221	124	Home
77740	M759	567	Home

Použitím klíčového slova **DISTINCT** v sekci SELECT zabráníme vícenásobnému uvedení téže věty ve výsledku dotazu.

SELECT *EMPNUM*
FROM *PC*

Výsledek:

EMPNUM
611
124
567
124
567

SELECT DISTINCT *EMPNUM*
FROM *PC*

Výsledek:

EMPNUM
124
567
611

GROUP BY I

Tabulka SOFTWARE

PACKID	TAGNUM	INSTDATE	SOFTCOST
AC01	32808	09/13/95	754.95
DB32	32808	12/03/95	380.00
DB32	37691	06/15/95	380.00
DB33	57772	05/27/95	412.77
WP08	32808	01/12/96	185.00
WP08	37691	06/15/95	227.50
WP08	57772	05/27/95	170.24
WP09	59836	10/30/95	35.00
WP09	77740	05/27/95	35.00

Věty, které projdou případnou podmínkou WHERE, se seskupí do skupin se stejnou hodnotou atributu *TAGNUM.* Pro každou takovou skupinu se určí suma hodnot atributu *SOFTCOST,* která se objeví ve výsledku – viz sloupec *SUM1* výsledku dotazu.
Bez uvedení sekce ORDER BY by nebylo pořadí skupin ve výsledku definováno.
Podmínka HAVING se vztahuje na celou skupinu a pomocí ní lze některé skupiny z výsledku dotazu vyfiltrovat.

```
SELECT TAGNUM, SUM( SOFTCOST )
  FROM SOFTWARE
  GROUP BY TAGNUM
  ORDER BY TAGNUM
```

Výsledek:

G_TAGNUM	SUM1
32808	1319.95
37691	607.50
57772	583.01
59836	35.00
77740	35.00

```
SELECT TAGNUM, SUM( SOFTCOST )
  FROM SOFTWARE
  GROUP BY TAGNUM
    HAVING SUM( SOFTCOST ) > 600
  ORDER BY TAGNUM
```

Výsledek:

G_TAGNUM	SUM1
32808	1319.95
37691	607.50

GROUP BY II

Tabulka SOFTWARE

PACKID	TAGNUM	INSTDATE	SOFTCOST
AC01	32808	09/13/95	754.95
DB32	32808	12/03/95	380.00
DB32	37691	06/15/95	380.00
DB33	57772	05/27/95	412.77
WP08	32808	01/12/96	185.00
WP08	37691	06/15/95	227.50
WP08	57772	05/27/95	170.24
WP09	59836	10/30/95	35.00
WP09	77740	05/27/95	35.00

```
SELECT TAGNUM, SUM( SOFTCOST )
   FROM SOFTWARE
   GROUP BY TAGNUM
   ORDER BY TAGNUM
```

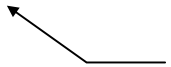
Výsledek:

G_TAGNUM	SUM1
32808	1319.95
37691	607.50
57772	583.01
59836	35.00
77740	35.00

```
SELECT TAGNUM, SUM( SOFTCOST )
   FROM SOFTWARE
   GROUP BY TAGNUM
      HAVING SUM( SOFTCOST ) > 600
   ORDER BY TAGNUM
```

Výsledek:

G_TAGNUM	SUM1
32808	1319.95
37691	607.50



Další tři skupiny se do výsledku (narozdíl od příkladu vlevo) nedostaly, neboť v jejich případě nebyla hodnota sloupce SUM1 větší než 600, jak požaduje podmínka HAVING.

HAVING versus WHERE

- Podmínka **WHERE** se vyhodnocuje pro jednotlivou větu a buď pro danou hodnotu nabývá hodnoty **true** nebo **false**.
- Věty, pro něž podmínka **WHERE** jsou vybrány do výstupu dotazu. Podmínka **WHERE** se tedy vyhodnocuje pro jednu větu a postupně se aplikuje na všechny věty vstupní tabulky (nebo joinu vstupních tabulek). V podmínce **WHERE** se tedy nemohou vyskytnout agregační funkce, protože aplikace agregační funkce na jedinou větu nemá smysl.
- Podmínka **HAVING** se vyhodnocuje pro všechny věty dané skupiny najednou. Pro danou skupinu vět nabývá hodnody **true** nabo **false**. Neaplikuje se tedy větu po větě, ale na všechny věty dané skupiny najednou.
- Skupiny s podmínkaou **HAVING** vyhodnocenou jako **true** jsou vybrány do výsledku dotazu.
- Protože se podmínka **HAVING** vyhodnocuje nad několika větami současně, má smysl, aby (narozdíl od podmínky WHERE) obsahovala agregační funkce.
- Kromě agregačních funkcí může obsahovat i atributy vyjmenované v sekci **GROUP BY**.
- Jiné atributy než ty, které jsou vyjmenovány v sekci **GROUP BY**, se nemouhou v podmínce **HAVING** vyskytnout (s výjimkou výskytu na místě argumentu nějaké agregační funkce), protože mohou mít pro různé věty téže skupiny různé hodnoty a nebylo by tudíž možné určit jednoznačnou hodnotu takového atributu pro celou skupinu.

JOIN I

Tabulka EMPLOYEE

EMPNUM	EMPNAME	EMPPHONE
124	Alvarez	1212
567	Feinstein	8716
611	Dinh	2963

Tabulka PC

TAGNUM	COMPID	EMPNUM	LOCATION
32808	M759	611	Accounting
37691	B121	124	Sales
57772	C007	567 I	Info Systems
59836	B221	124	Home
77740	M759	567	Home

Rádi bychom se dotazovali na relaci, jež vznikne spojením těchto dvou tabulek.

JOIN II

SELECT *
FROM *PC, EMPLOYEE*

Join je spojení tabulek metodou každý s každým, t.j. každá věta z „levé“ tabulky se spáří s každou větou z „pravé“ tabulky.
To znamená, má-li tabulka *PC* 5 řádků a tabulka *EMPLOYEE* 3 řádky, má JOIN obou tabulek 15 řádků – viz níže.

Výsledek:

TAGNUM	COMPID	EMPNUM	LOCATION	EMPNUM	EMPNAME	EMPPHONE
32808	M759	611	Accounting	124	Alvarez	1212
32808	M759	611	Accounting	567	Feinstein	8716
32808	M759	611	Accounting	611	Dinh	2963
37691	B121	124	Sales	124	Alvarez	1212
37691	B121	124	Sales	567	Feinstein	8716
37691	B121	124	Sales	611	Dinh	2963
57772	C007	567	Info Systems	124	Alvarez	1212
57772	C007	567	Info Systems	567	Feinstein	8716
57772	C007	567	Info Systems	611	Dinh	2963
59836	B221	124	Home	124	Alvarez	1212
59836	B221	124	Home	567	Feinstein	8716
59836	B221	124	Home	611	Dinh	2963
77740	M759	567	Home	124	Alvarez	1212
77740	M759	567	Home	567	Feinstein	8716
77740	M759	567	Home	611	Dinh	2963

JOIN III (equijoin)

Častější operací je tzv. **equijoin**, t.j. výsledkem je join pouze těch řádků z „levé“ a „pravé“ tabulky, které se shodují v některém sloupci. Například equijoin tabulek *PC* a *EMPLOYEE*, který je definován shodou hodnoty sloupce *EMPNUM* tabulky *PC* se sloupcem *EMPNUM* tabulky *EMPLOYEE*, se realizuje příkazem:

SELECT *TAGNUM, COMPID, EMPLOYEE.EMPNUM, EMPNAME*
FROM *PC, EMPLOYEE*
WHERE ***PC.EMPNUM = EMPLOYEE.EMPNUM***

Výsledek:

TAGNUM	COMPID	EMPLOYEE.EMPNUM	EMPNAME
32808	M759	611	Dinh
37691	B121	124	Alvarez
57772	C007	567	Feinstein
59836	B221	124	Alvarez
77740	M759	567	Feinstein

JOIN IV (equijoin)

Další příklad:

SELECT *TAGNUM, COMPID, EMPLOYEE.EMPNUM, EMPNAME*
FROM *PC, EMPLOYEE*
WHERE ***PC.EMPNUM = EMPLOYEE.EMPNUM*** **AND** ***LOCATION*** **= 'Home'**

Podmínka pro equijoin může být doplněna v sekci WHERE o další selekční podmínky.

Výsledek:

TAGNUM	COMPID	EMPLOYEE.EMPNUM	EMPNAME
59836	B221	124	Alvarez
77740	M759	567	Feinstein

JOIN V (equijoin)

V sekci USING je seznam atributů (musí mít stejná jména v obou tabulkách), přes které se provádí equi-join.

SELECT *TAGNUM, COMPID, EMPNUM, EMPNAME*
FROM *PC* **INNER JOIN** *EMPLOYEES* USING (*EMPNUM)*

INNER JOIN (vnitřní join) - když se k větě z první tabulky nenajde v druhé tabulce věta splňující podmínku joinu, ona věta z první tabulky se do výsledku nepromítne.

Slovo INNER se může vynechat, je default. Opakem je OUTER JOIN.

JOIN VI (equijoin)

SELECT *TAGNUM, COMPID, EMPNUM, EMPNAME*
FROM *PC* **NATURAL JOIN** *EMPLOYEES*

Klíčové slovo **NATURAL** znamená, že se equ-join provádí přes všechny stejnojmenné atributy v obou tabulkách. Pak se nemusí uvádět sekce **USING**.

JOIN VII (equijoin)

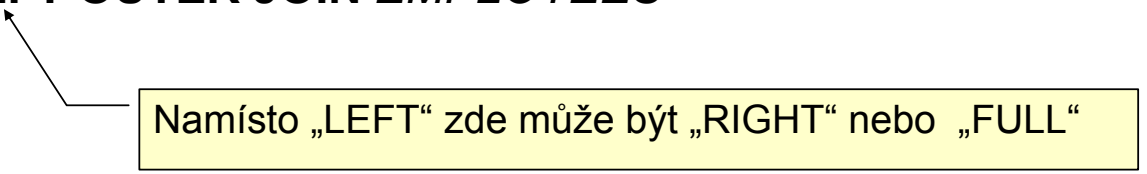
SELECT TAGNUM, COMPID, EMPNUM, EMPNAME
FROM PC **JOIN** EMPLOYEES **ON** *PC.EMPNUM* **=** *EMPLOYEES.EMPNUM*

Pokud nemají atributy, přes které se dělá equi-join, v obou tabulkách stejná jména, mohu podmínku equi-joinu vyjádřit v sekci **ON**.

JOIN VIII (OUTER JOIN)

Narozdíl od INNER JOIN se v případě OUTER JOIN do výsledku promítne věta z levé (**LEFT OUTER JOIN**), respektive z pravé **(RIGHT OUTER JOIN**), respektive z obou tabulek (**FULL OUTER JOIN**), i v případě, že nemá v druhé tabulce partnerskou větu. Atributy odpovídající chybějící partnerské větě dostanou hodnotu **NULL.**

SELECT *TAGNUM, COMPID, EMPNUM, EMPNAME*
FROM *PC* **LEFT OUTER JOIN** *EMPLOYEES*



UNION

```
SELECT COMPID, MFGNAME
    FROM COMPUTER
    WHERE PROCTYPE = '486DX'
UNION
SELECT COMPUTER.COMPID, MFGNAME
    FROM COMPUTER, PC
    WHERE COMPUTER.COMPID = PC.COMPID
          AND LOCATION = 'Home'
```

INTERSECTION

```
SELECT COMPID, MFGNAME
    FROM COMPUTER
    WHERE PROCTYPE = '486DX'
INTERSECT
SELECT COMPUTER.COMPID, MFGNAME
    FROM COMPUTER, PC
    WHERE COMPUTER.COMPID = PC.COMPID
          AND LOCATION = 'Home'
```

DIFFERENCE

```
SELECT COMPID, MFGNAME
    FROM COMPUTER
    WHERE PROCTYPE = '486DX'
EXCEPT
SELECT COMPUTER.COMPID, MFGNAME
    FROM COMPUTER, PC
    WHERE COMPUTER.COMPID = PC.COMPID
          AND LOCATION = 'Home'
```

Integritní omezení I

Povinný údaj	**NOT NULL**
Unikátnost hodnoty	**UNIQUE**
Přípustné hodnoty:	**CHECK (***PC.LOCATION IN ('Accounting', 'Sales', 'Info Systems', 'Home'))* nebo ekvivalentně **CHECK (** *PC.LOCATION = 'Accounting'* **OR** *PC.LOCATION = 'Sales'* **OR** *PC.LOCATION = 'Info Systems'* **OR** *PC.LOCATIONS = 'Home'* **)**
Primární klíče:	**PRIMARY KEY (***TAGNUM***)** **PRIMARY KEY (***PACKID, TAGNUM***)**
Cizí klíče:	**FOREIGN KEY (***COMPID***) REFERENCES** *COMPUTER*

Integritní omezení II

Příklad:

CREATE TABLE *PC*
(*TAGNUM* **CHAR(***5***),**
COMPID **CHAR(***4***),**
EMPNUM **DECIMAL(***3***),**
LOCATION **CHAR(***12***) CHECK (** *PC.LOCATION IN ('Accounting', 'Sales','Info Systems', 'Home'))*
PRIMARY KEY (*TAGNUM***)**
FOREIGN KEY (*COMPID***) REFERENCES** *COMPUTER*
FOREIGN KEY (*EMPNUM***) REFERENCES** *EMPLOYEE* **)**

Integritní omezení III

```
CREATE ASSERTION A1 CHECK
    ( NOT EXISTS
        ( SELECT *
            FROM PACKAGE
            WHERE PACKCOST <
                ( SELECT MAX (SOFTCOST)
                    FROM SOFTWARE
                    WHERE PACKAGE.PACKID = SOFTWARE.PACKID
    )    )       )
```

ztratilo-li toto integritní omezení smysl, lze je odstranit:

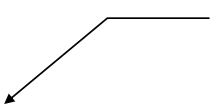
```
DROP ASSERTION A1
```

Domény – uživatelsky definované datové typy

CREATE DOMAIN ***LOCATIONS*** **CHAR(***12***)**
CHECK (VALUE = *'Accounting'* **OR**
VALUE = *'Sales'* **OR**
VALUE = *'Info Systems'* **OR**
VALUE = *'Home'* **)**

... potom použiji takto:

CREATE TABLE *PC*
(...
...
LOCATION LOCATIONS
...
...)



Deklarace atributu *LOCATION* pomocí domény *LOCATIONS* .

Vnořené dotazy, subquery I

```
SELECT PACKID, PACKNAME
  FROM PACKAGE
  WHERE PACKCOST >
                    ( SELECT AVG( PACKCOST )
                      FROM PACKAGE
                      WHERE PACKTYPE = 'Database' )
```

Komentář: Nejprve se vyhodnotí subquery, její výsledek se uloží do dočasné tabulky (v tomto případě 1 sloupec, 1 řádek), pak se vyhodnotí vnější query.

Výsledek vnořeného dotazu byl:

AVG1
405.09

Výsledek celého dotazu:

PACKID	PACKNAME
AC01	Boise Accounting
DB33	Manta

Vnořené dotazy, subquery II

Stejného výsledku, jako dává equijoin, lze dosáhnout i jinými prostředky – viz níže.
Měla by se však dávat přednost equijoinu před použitím zanořených dotazů.

```
SELECT PACKNAME
FROM PACKAGE
WHERE PACKID IN
   ( SELECT PACKID
     FROM SOFTWARE
     WHERE TAGNUM = '32808')
```

Výsledek:

PACKNAME
Boise Accounting
Manta

```
SELECT PACKNAME
  FROM SOFTWARE JOIN PACKAGE
  WHERE TAGNUM = '32808'
```

Výsledek:

PACKNAME
Boise Accounting
Manta

Vnořené dotazy, subquery III

IN versus EXISTS

```
SELECT TAGNUM, COMPID
FROM PC
WHERE EXISTS
 ( SELECT *
   FROM SOFTWARE
   WHERE PC.TAGNUM  = SOFTWARE.TAGNUM
         AND PACKID = 'WP08')
```

```
SELECT TAGNUM, COMPID
FROM PC
WHERE TAGNUM   IN
        ( SELECT TAGNUM
          FROM SOFTWARE
          WHERE PACKID = 'WP08')
```

Korelovaný poddotaz (correlated subquery):
Vnořený dotaz se vyhodnocuje (provádí) pro každou řádku vnějšího dotazu znovu, neboť hodnota atributu PC.TAGNUM pro momentálně vyhodnocovaný daný řádek vnějšího dotazu je vlastně parametrem dotazu vnořeného.
Použití korelovaných poddotazů bychom se měli vyhnout, neboť je mimořádně neefektivní.

Výsledek:

TAGNUM	COMPID
32808	M759
37691	B121
57772	C007

Výsledek:

TAGNUM	COMPID
32808	M759
37691	B121
57772	C007

Vnořený select: kam všude

```
SELECT (SELECT ...)
    FROM (SELECT ...) tname
    WHERE   abc > (SELECT ...)
                or abc IN (SELECT ...)
    GROUP BY ...
        HAVING ... (SELECT ...)
```

- často musí vracet jeden sloupec
- někdy i jedinou řádku (u aritmetického porovnávání)

Vnořený select: kam všude

Vybrané názory z diskusních fór:

> Has anyone ever used a statement like this:
>
> Select column1, column 2, (select columnA from table2) from table1 where....?
>
> This is contrary to anything I have ever seen in SQL (especially ANSI/ISO SQL), and I would appreciate your comments. Thanks
> SK

> No I haven't, not once in in 17 years. More to the point I have no intention of ever doing so.
>
> That would give you a cartesian product if tyhe parse didn't get really upset. Table1 * Table2 records.
>
> Same as Select Column1,Column2,ColumnA From Table1,Table2

Kvantifikátor ALL

Slovní formulace dotazu:
Najdi instalaci software, jejíž pořizovací cena byla větší než současná katalogová cena **libovolného** produktu.

SOFTWARE

PACKID	TAGNUM	INSTDATE	SOFTCOST
AC01	32808	09/13/95	754.95
DB32	32808	12/03/95	380.00
DB32	37691	06/15/95	380.00
DB33	57772	05/27/95	412.77
WP08	32808	01/12/96	185.00
WP08	37691	06/15/95	227.50
WP08	57772	05/27/95	170.24
WP09	59836	10/30/95	35.00
WP09	77740	05/27/95	35.00

SELECT *PACKID, TAGNUM, INSTDATE, SOFTCOST*
FROM *SOFTWARE*
WHERE *SOFTCOST* **> ALL**
(SELECT *PACKCOST*
FROM *PACKAGE* **)**

Výsledek:

PACKID	*TAGNUM*	*INSTDATE*	*SOFTCOST*
AC01	32808	09/13/95	754.95

Kvantifikátor ANY

Slovní formulace dotazu:
Najdi instalaci software, jejíž pořizovací cena byla větší než současná katalogová cena některého produktu.

SOFTWARE

PACKID	TAGNUM	INST DATE	SOFT COST
AC01	32808	09/13/95	754.95
DB32	32808	12/03/95	380.00
DB32	37691	06/15/95	380.00
DB33	57772	05/27/95	412.77
WP08	32808	01/12/96	185.00
WP08	37691	06/15/95	227.50
WP08	57772	05/27/95	170.24
WP09	59836	10/30/95	35.00
WP09	77740	05/27/95	35.00

```
SELECT PACKID, TAGNUM, INSTDATE, SOFTCOST
FROM SOFTWARE
WHERE SOFTCOST > ANY
                ( SELECT PACKCOST
                  FROM PACKAGE )
```

Výsledek:

PACKID	TAGNUM	INSTDATE	SOFTCOST
AC01	32808	09/13/95	754.95
DB32	32808	12/03/95	380.00
DB32	37691	06/15/95	380.00
DB33	57772	05/27/95	412.77
WP08	32808	01/12/96	185.00
WP08	37691	06/15/95	227.50
WP08	57772	05/27/95	170.24
WP09	59836	10/30/95	35.00
WP09	77740	05/27/95	35.00

Význam použití ALIASu

Slovní formulace dotazu:
Najdi všechny dvojice produktů mající tentýž název.

PACKAGE

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

Tabulku *PACKAGE* otevírám 2x – jednou k ní budu přistupovat pod jménem *FIRST*, podruhé pod jménem *SECOND*. Zajímají mne tedy všechny kombinace vět z tabulek *FIRST* a *SECOND*, které se shodují v hodnotě atributu *PACKNAME*.

SELECT *FIRST.PACKID, FIRST.PACKNAME, SECOND.PACKID, SECOND.PACKNAME*
FROM *PACKAGE FIRST, PACKAGE SECOND*
WHERE *FIRST.PACKNAME* **=** *SECOND.PACKNAME* **AND** *FIRST.PACKID* **<** *SECOND.PACKID*

Výsledek:

PACKID	PACKNAME	PACKID	PACKANME
DB32	Manta	DB33	Manta

Vytvoření kopie existující tabulky I

```
CREATE TABLE DBPACK
    (   PACKID      CHAR(4),
        PACKNAME    CHAR(20),
        PACKVER     NUMERIC(4,2),
        PACKCOST    NUMERIC(5,2) )

INSERT INTO DBPACK
    SELECT *
        FROM PACKAGE
        WHERE PACKTYPE = 'Database'
```

V tomto případě má cílová tabulka *DBPACK* stejnou strukturu jako vzorová tabulka *PACKAGE*,

Vytvoření kopie existující tabulky II

```
CREATE TABLE WPPACK
    (   PACKID      CHAR(4),
        PACKNAME    CHAR(20),
        PACKTYPE    CHAR(15) )

INSERT INTO DBPACK
    SELECT PACKID, PACKNAME, PACKTYPE
        FROM PACKAGE
        WHERE PACKTYPE = 'Word Processing'
        ORDER BY PACKNAME
```

V tomto případě bude množina atributů cílové tabulky podmnožinou atributů vzorové tabulky.

Stejně tak množina řádků cílové tabulky bude podmnožinou množiny řádků vzorové tabulky.

VIEW I

View lze chápat jako tabulku, jež neobsahuje explicitně zadaná data. Tato tabulka je „pohledem“ na jinou tabulku nebo join tabulek.

View přitom slouží nejen k získání dat z databáze ale i k jejich modifikaci.

```
CREATE VIEW DATABASE AS
    SELECT PACKID, PACKNAME, PACKCOST
    FROM PACKAGE
    WHERE PACKTYPE = 'Database'
```

VIEW nemusí být materializováno – zaniká spolu s databázovým spojením (session).

Materializované VIEW existuje nezávisle na databázovém spojení (session).

VIEW II

PACKAGE

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	**Manta**	1.50	Database	**380.00**
DB33	**Manta**	2.10	Database	**430.18**
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

Obsahem View *DATABASE* budou buňky se žlutým pozadím.

Při definici view můžeme omezit jeho přístup pouze k vyjmenovaným atributům.

```
CREATE VIEW DATABASE ( PACKID, PACKNAME, PACKCOST ) AS
   SELECT PACKID, PACKNAME, PACKCOST
   FROM PACKAGE
   WHERE PACKTYPE = 'Database'
```

Na view se lze obracet stejně jako na tabulku.

V tomto případě bude výsledkem jediná řádka:

PACKID	PACKNAME	PACKCOST
DB33	Manta	430.18

VIEW III

Atributy view mohou mít jiná jména než atributy zdrojové tabulky.

CREATE VIEW *DATABASE* **(** ***PKID*****,** ***NAME*****,** ***COST*** **) AS**
SELECT ***PACKID*****,** ***PACKNAME*****,** ***PACKCOST***
FROM *PACKAGE*
WHERE *PACKTYPE* **=** 'Database'

Význam view:

1. Datová nezávislost.
   Změna struktury databáze u atributů neúčastnících se view neovlivní práci s view.
2. Různé pohledy na tatáž data. Uživatel nevidí, co nemá.

Problémy při update view:

- pokud view nezahrnuje všechny sloupce p;vodní tabulky a přidáme větu do view, jaká hodnota se v původní tabulce přiřadí atributům neúčastnícím se view ? NULL !
- pokus o přidání řádku ('AC01','DATAQUICK',250.00) musí selhat, protože v tabulce *PACKAGE* již věta s primárním klíce 'AC01' existuje. To ovšem může uživatele view překvapit, protože on vidí jen věty obsažené ve view.

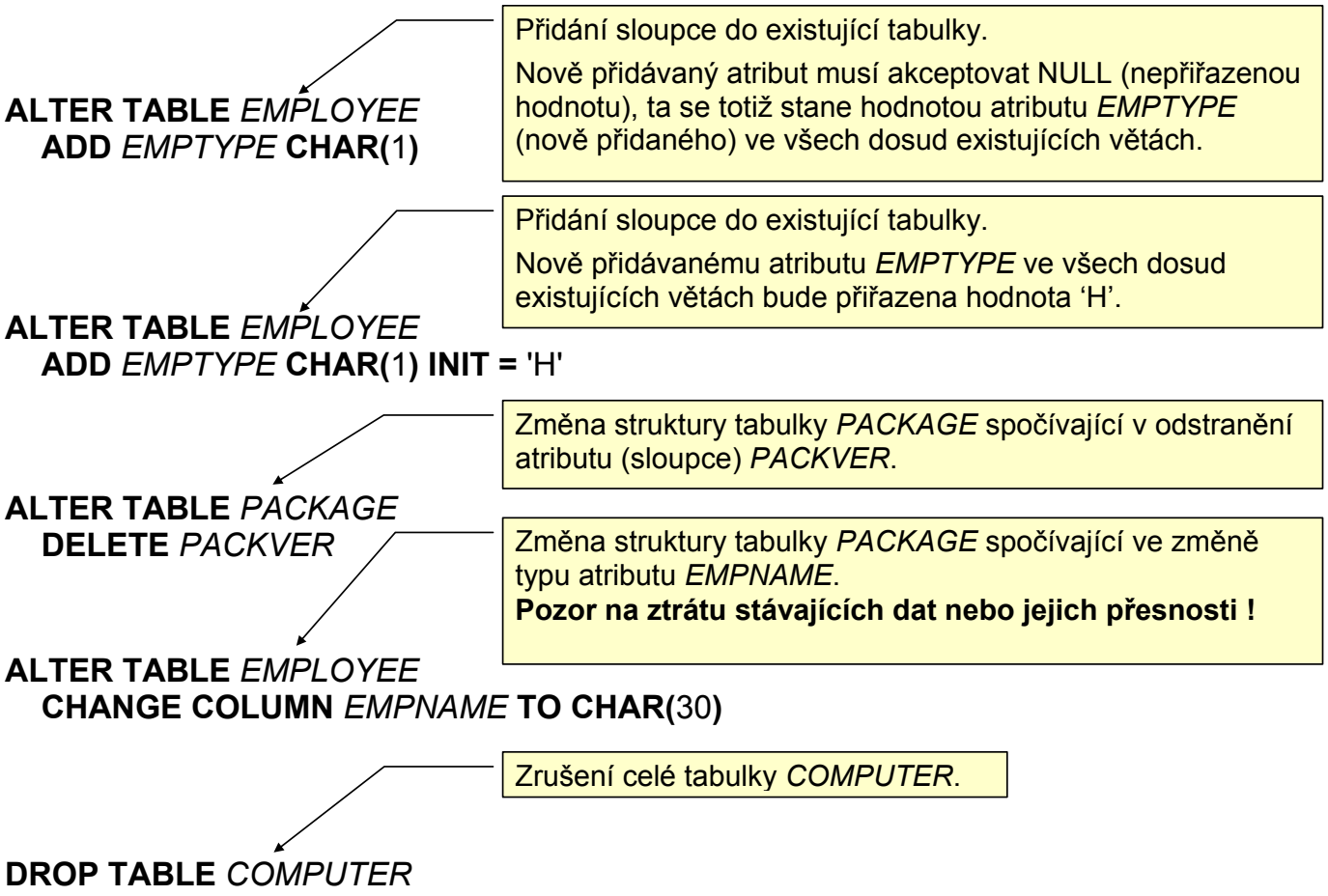
Změna obsahu databáze

```
UPDATE PACKAGE
   SET PACKNAME = 'Manta II'
   WHERE PACKID = 'DB33'

UPDATE PACKAGE
   SET PACKCOST = PACKCOST * 1.02
   WHERE PACKTYPE = 'Database'
         AND PACKCOST > 400

UPDATE EMPLOYEE
   SET EMPPHONE = NULL
   WHERE EMPNUM = 124
```

Zvětši hodnotu atributu *PACKCOST* tabulky *PACKAGE* o dvě procenta ve všech větách, pro něž je splněna podmínka WHERE.

Odstraň hodnotu atributu *EMPPHONE* tabulky *EMPLOYEE* ve všech větách, pro něž je splněna podmínka WHERE.

Změna struktury databáze



Udělení práva k provádění akcí

Uživatel *JONES* bude smět číst data z tabulky *EMPLOYEE*.

GRANT SELECT ON *EMPLOYEE* **TO** JONES

Libovolný uživatel bude smět číst atributy *PACKID, PACKNAME a PACKTYPE* tabulky *PACKAGE*.

GRANT SELECT ON *PACKAGE* **(** *PACKID, PACKNAME, PACKTYPE* **) TO PUBLIC**

Uživatelé SMITH a BROWN budou smět vkládat řádky do tabulky *PACKAGE*.

GRANT INSERT ON *PACKAGE* **TO** SMITH, BROWN

Uživatel ANDERSON bude smět měnit hodnoty atributů tabulky *EMPLOYEE*.

GRANT UPDATE ON *EMPLOYEE* **(** *EMPNAME, EMPPHONE* **) TO** ANDERSON

Uživatel MARTIN bude smět rušit řádky tabulky *SOFTWARE*.

GRANT DELETE ON *SOFTWARE* **TO** MARTIN

Uživatel ROBERTS bude smět vytvářet indexy pro tabulku *COMPUTER*.

GRANT INDEX ON *COMPUTER* **TO** ROBERTS

Uživatel THOMAS bude smět měnit strukturu tabulky *EMPLOYEE*.

GRANT ALTER ON *EMPLOYEE* **TO** THOMAS

Uživatel WILSON bude smět dělat cokoliv (viz výše) s tabulkami *COMPUTER* a *EMPLOYEE*.

GRANT ALL ON *COMPUTER*, *EMPLOYEE*, **PC TO** WILSON

Odejmutí přístupového práva

REVOKE **SELECT** ON *EMPLOYEE* FROM *JONES*

Příkazy GRANT a REVOKE jsou aplikovatelné jak na tabulky tak i na view.

Indexy I

výhody:
- zvýšení efektivnosti vyhledávání
  (záleží na kvalitě optimalizce dotazu)
- třídění

nevýhody:
- nároky na kapacitu média
- index musí být updatován při každém update databáze

Ačkoliv standard SQL92 nedefinuje následující příkazy pro vytvoření a odstranění indexu, ve většině aplikací jsou k disposici v téže syntaktické podobě.

CREATE **INDEX** *CUSTIND2* ON *EMPLOYEE* (*COMPID*)

Vytvoří index pojmenovaný *CUSTIND2* pro tabulku *EMPLOYEE.* Indexačním výrazem bude jednoprvková množina sloupců { *COMPID* }.

Indexy II

CREATE INDEX *SOFTIND* ON *SOFTWARE* (*PACKID, TAGNUM*)

Index může být vytvořen nad více než jedním atributem (sloupcem).

Indexy II

CREATE INDEX *SOFTIND* ON *SOFTWARE* (*PACKID, TAGNUM*)

Index může být vytvořen nad více než jedním atributem (sloupcem).

CREATE INDEX *PACKIND3* ON *PACKAGE* (*PACKNAME, PACKVER* **DESC**)

Indexu nastavit vzestupné nebo sestupné třídění.

Indexy III

Odstranění nepotřebného klíče:

DROP INDEX *PACKIND*

Indexy IV

CREATE **UNIQUE** INDEX *PACKIND* ON *PACKAGE* (*PACKID*)

Správa indexu nedovolí, aby se v tabulce vyskytlo více vět s touž hodnotou atributu(ů) nad nímž (nimiž) je postaven daný index.

SQL92 umožňuje předepsat tuto jednoznačnost v definici tabulky popř. dalšími prostředky pro definici integritních omezení.

To je správnější, neboť tak se tato důležitá podmínka stává součástí definice schematu (fyzického datového modelu) a není ponechána na vůli správce databáze, zda vytvoří vhodný index.

(Ne)Existence indexu má mít vliv pouze na efektivitu (výkon) databáze a nikoliv na její funkci (a definice jednozncnosti některého atributu je funkční záležitostí).