

# Dotazovací jazyk SQL II

# SELECT VIII

## Vestavěné (BUILT-IN) agregační funkce

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

COUNT1
2

Název sloupce zvolil  
databázový systém sám.  
↖

# 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** nepřřazené hodnoty (NULL).



# SELECT XV

Tabulka PC

TAGNUM	COMPID	EMPNUM	LOCATION
32808	M759	611	Accounting
37691	B121	124	Sales
57772	C007	567	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
```

```
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
57772	583.01
59836	35.00
77740	35.00

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 případech 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 hodnoty **true** nebo **false**. Neaplikuje se tedy větu po větě, ale na všechny věty dané skupiny najednou.
- Skupiny s podmínkou **HAVING** vyhodnocenou jako **true** jsou vybrány do výsledku dotazu.
- Protože se podmínka **HAVING** vyhodnocuje nad několika větami současně, má smysl, aby (na rozdíl 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 nemohou 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**

<b>EMPNUM</b>	<b>EMPNAME</b>	<b>EMPPHONE</b>
124	Alvarez	1212
567	Feinstein	8716
611	Dinh	2963

**Tabulka PC**

<b>TAGNUM</b>	<b>COMPID</b>	<b>EMPNUM</b>	<b>LOCATION</b>
32808	M759	611	Accounting
37691	B121	124	Sales
57772	C007	567	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)

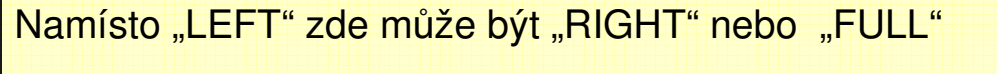
```
SELECT TAGNUM, COMPID, EMPNUM, EMPNAME  
FROM PC JOIN EMPLOYEES ON PC.EMPNUM =  
EMPLOYEES.EMPNUM
```

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
```



Namísto „LEFT“ zde může být „RIGHT“ nebo „FULL“

# 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

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



# 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 použijí 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 vnořného 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:

<i>PACKID</i>	<i>TAGNUM</i>	<i>INSTDATE</i>	<i>SOFTCOST</i>
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
<b>DB32</b>	<b>Manta</b>	1.50	Database	<b>380.00</b>
<b>DB33</b>	<b>Manta</b>	2.10	Database	<b>430.18</b>
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 klíče '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

**ALTER TABLE *EMPLOYEE*  
ADD *EMPTYTYPE* CHAR(1)**

Přidání sloupce do existující tabulky.  
Nově přidávaný atribut musí akceptovat NULL (nepřiřazenou hodnotu), ta se totiž stane hodnotou atributu *EMPTYTYPE* (nově přidaného) ve všech dosud existujících větách.

**ALTER TABLE *EMPLOYEE*  
ADD *EMPTYTYPE* CHAR(1) INIT = 'H'**

Přidání sloupce do existující tabulky.  
Nově přidávanému atributu *EMPTYTYPE* ve všech dosud existujících větách bude přiřazena hodnota 'H'.

**ALTER TABLE *PACKAGE*  
DELETE *PACKVER***

Změna struktury tabulky *PACKAGE* spočívající v odstranění atributu (sloupce) *PACKVER*.

**ALTER TABLE *EMPLOYEE*  
CHANGE COLUMN *EMPNAME* TO CHAR(30)**

Změna struktury tabulky *PACKAGE* spočívající ve změně typu atributu *EMPNAME*.  
**Pozor na ztrátu stávajících dat nebo jejich přesnosti !**

**DROP TABLE *COMPUTER***

Zrušení celé tabulky *COMPUTER*.

# 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 cokoli (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 efektivity vyhledávání (záleží na kvalitě optimalizace 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 dispozici 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 jednoznačnosti některého atributu je funkční záležitostí).

# Embedded SQL I

```
int cust_assist() {  
char title[51], year[11], result[102], star_name[51];  
  
EXEC SQL DECLARE CURSOR movie_cursor FOR  
    SELECT title, CAST (year_released AS CHARACTER(10))  
        FROM movie_titles;  
  
EXEC SQL DECLARE CURSOR star_cursor FOR  
    SELECT actor_last_name), actor_first_name  
        FROM movie_stars  
        WHERE movie_title = :title AND  
            year_released = CAST(:year AS DATE);  
  
EXEC SQL OPEN movie_cursor;
```

# Embedded SQL II

```
while (continue1 == 1) /* cyklus pres jednotlivy filemy */
{
    EXEC SQL FETCH NEXT FROM movie_cursor INTO :title, :year ;
    ...

    EXEC SQL OPEN star_cursor;

    while (continue2 == 1) /* cyklus pres herce daneho filmu */
    {
        EXEC SQL FETCH NEXT FROM star_cursor INTO :name ;
        ...
        EXEC SQL CLOSE star_cursor ;
        ...
    }

    EXEC SQL CLOSE movie_cursor
}
```

- Dodělat:
  - Integritní omezení