Database systems

Structured query language SQL - II

OPP
A
PRA HA
PRA GUE
PRA GA
PRA G
EVROPSKÁ UNIE

INSERT INTO

Table name

List of values

INSERT INTO *EMPLOYEE* **VALUES (** *611, 'Dinh Melissa', 2963* **)**

List of columns not expressed – values will be assigend to columns in the order in which the columns have been defined in the CREATE TABLE statement.

Table name

List of columns

List of values

INSERT INTO *EMPLOYEE* **(** *EMPNUM, EMPNAME* **)**
VALUES (*611, 'Dinh Melissa'* **)**

The section VALUES specifies one or more (comma separated) tuples of values. The values will be assigend to columns in the order given by the list of columns.

Columns not introduced in the column list will get the NULL value. In this case EMPPHONE will get NULL.

SELECT I

SELECT <*list of columns or **>
    FROM <*relation definition*>
    WHERE <*selection condition*>
    GROUP BY <*list of columns*>
        HAVING <*group filtering condition*>
    ORDER BY <*list of column_defs*>

column_def ::= <column name> [<asc|desc>]

Logical operators:

=	equals
<=	less than or equal
<	les than
>=	greater than or equal
>	greater than
<>	not equal
!=	not equal

SELECT II

PACKAGE table

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 PACKID, PACKNAME, PACKCOST
       FROM PACKAGE
       WHERE   PACKCOST >= 200 AND
               PACKCOST <= 400
```

```
SELECT PACKID, PACKNAME, PACKCOST
       FROM PACKAGE
       WHERE PACKCOST
                  BETWEEN 200 AND 400
```

Result:

PACKID	PACKNAME	PACKCOST
DB32	Manta	380.00
SS11	Limitless View	217.95

SELECT III

PACKAGE table

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

boolean predicate LIKE
character % is a wildcard, i.e. matches with any charatcter (sub)string

SELECT *PACKID, PACKNAME*
FROM *PACKAGE*
WHERE *PACKNAME* ***LIKE*** *'%&%'*

Result:

PACKID	PACKNAME
WP08	Words & More

SELECT IV

Precate **"IS NULL"** is equal to **„true“** iff the respective coulmn has assigned no value

SELECT *EMPNUM, EMPNAME*
  FROM *EMPLOYEE*
  WHERE *EMPPHONE* **IS NULL**

TRUE for those rows in which *EMPPHONE* has an undefined value, i.e. NULL.

SELECT V (arithemetic operators)

PACKAGE table

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 PACKID, PACKNAME**, (.90** * PACKCOST **)**
FROM **PACKAGE**

Name of this column generated by DBMS client

Result:

i.e. 0.9 * 725.83
0.9 * 380.00
0.9 * 430.18

PACKID	PACKNAME	EXP1
AC01	Boise Accounting	635.25
DB32	Manta	342.00
DB33	Manta	387.16
SS11	Limitless View	196.16
WP08	Words & More	166.50
WP09	Freeware Processing	27.00

SELECT VI

PACKAGE table

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 PACKID, PACKNAME, PACKTYPE
FROM PACKAGE
WHERE PACKTYPE IN ('Database',
                   Spreadsheet',
                   'Word Processing')
```

```
SELECT PACKID, PACKNAME,
PACKTYPE
FROM PACKAGE
WHERE PACKTYPE = 'Database' OR
      PACKTYPE = Spreadsheet' OR
      PACKTYPE = 'Word Processing'
```

Result:

PACKID	PACKNAME	PACKTYPE
DB32	Manta	Database
DB33	Manta	Database
SS11	Limitless View	Spreadsheet
WP08	Words & More	Word Processing
WP09	Freeware Processing	Word Processing

SELECT VII (sorting)

PACKAGE table

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

Order of rows in the result of the query is undefined unless specified by OREDR clause:

SELECT *PACKID, PACKNAME, PACKTYPE, PACKCOST*
  FORM *PACKAGE*
  ORDER BY *PACKTYPE, PACKCOST* **DESC**

Rows will be oredered primarily by *PACKTYPE.* The order of rows with an equal value of *PACKTYPE* will be defined by *PACKCOST*.
DESC ... descending
ASC   ... ascending (default)

Result:

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

SELECT VIII

BUILT-IN aggregation functions

COUNT(***column*** **)** **COUNT(*)**	Number of rows matching WHERE condition. Independent of the column name, hence * is possible.
COUNT(DISTINCT ***column*** **)**	Number of **different** values of given column that appear in all rows matching WHERE condition.
SUM(***column*** **)**	Summ of values of the column over all rows matching WHERE condition.
AVG(***column*** **)**	Average of values of the column over all rows matching WHERE condition.
MAX(***column*** **)**	Maximal value in the the column over all rows matching WHERE condition.
MIN(***column*** **)**	Minimal value in the the column over all rows matching WHERE condition.

SELECT IX

PACKAGE table

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:

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

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

Result:

COUNT1
2

SELECT X

PACKAGE table:

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'
```

Result:

COUNT1
1

SELECT XI

PACKAGE table:

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*

Result:

COUNT1	**SUM2**
6	1968.96

SELECT XII

PACKAGE table:

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*

Result:

COUNT1	**AVG2**
6	**328.16**

SELECT XIII

PACKAGE table

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*

Result:

COUNT1	**MAX2**
6	725.83

SELECT XIV

PACKAGE table

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*

Result:

COUNT1	MIN2
6	30.00

SELECT XV

PC table

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

DISTINCT prohibits mulitple apearance of the same row in the result.

SELECT *EMPNUM*
FROM *PC*

Result:

EMPNUM
611
124
567
124
567

SELECT DISTINCT *EMPNUM*
FROM *PC*

Result:

EMPNUM
124
567
611

GROUP BY I

SOFTWARE table

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

Rows that have gone through WHERE condition, are grouped. Rows in one group have equal values of all columns specified in the GROUP BY clause - here *TAGNUM.*
The aggregation function is evaluated for each row separately.
ORDER BY just sorts the goups in the output.
Each goup represented by a single row in the output.
HAVING condition makes possible to filter out some of the groups

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

Result:

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
```

Result:

G_TAGNUM	SUM1
32808	1319.95
37691	607.50

GROUP BY II

SOFTWARE table

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
```

Result:

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
```

Result:

G_TAGNUM	**SUM1**
32808	1319.95
37691	607.50

The other 3 groups not present in the putput as they did not match the HAVING condition.

HAVING versus WHERE

- **WHERE** condition evalueated for each single row of the input relation.
- Only rows selected by **WHERE** condition go on to further processing by the query.
- Aggregation functions must not take part in the **WHERE** condition as it makes no sense to apply an aggregation function to a single row.
- **HAVING** condition is evaluated for for the whole group – one group by the other. So, it is not applied to a single row, but to a set of rows forming the group.
- Groups matching the **HAVING** condition are placed to the output of the query.
- As the **HAVING** condition is evaluated on multiple rows (whole group) simultaeously, it makes sense that an aggregation function may participate in the condition.
- Besides aggregation functions, **HAVING** condition may contain also those column names that are listed in **GROUP BY** clause.
- Other columns than those listed in **GROUP BY** must not participate in HAVING condition otherwise than arguments of an aggregation function. The reason is that they may have different values, i.e. their value is not property of the whole group.
- The same holds for attributes listed in SELECT clause.

JOIN I

EMPLOYEE:

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

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

We would like to query a relation that is defined as a join of these two tables.

JOIN II

Each row from PC joimned with each row from EMPLOYEE.
PC 5 rows, *EMPLOYEE* 3 rows => JOIN has 15 rows.

SELECT *
FROM *PC, EMPLOYEE*

Result:

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)

More frequently used (and more useful) is so called **equijoin.**
Only those rows tha „belong together“ are combined.

Typically, we would wish to combine rows that matches the value of a primary key of one table and a foreign key of the other one.

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

Result:

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)

Another example:

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

The equijoin condition may be followed by selection conditions in the WHERE clause.

Result:

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

JOIN V (equijoin)

USING clause contains a single list of columns (these have to have equal names in both tables), that define the **equi**-join.

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

INNER JOIN (inner is by default – may be omitted) – if the value of the matching columns is null in a row of one of those two tables, this row will not take part in the equi-join.

OUTER JOIN is the opposite – see later.

JOIN VI (equijoin)

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

NATURAL means that the equi-join is carried out over all columts that have equal names in both tables. Then **USING** is omitted.

JOIN VII (equijoin)

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

Most common form of equi-JOIN

JOIN VIII (OUTER JOIN)

As contrary to **INNER JOIN**, in case of **LEFT** (RIGHT/FULL) **OUTER JOIN** a row from the LEFT (RIGHT/BOTH) that have a NULL in the column that shall match with a column in the other table will be put to the result. Those columns that came from the other table (this row has no partner there) will get NULL (if no integrity constrain violation).

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

LEFT, RIGHT or 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'
```

Integrity contstraints I

Required value	**NOT NULL**
Unique value	**UNIQUE**
Acceptable values:	**CHECK (***PC.LOCATION IN ('Accounting', 'Sales', 'Info Systems', 'Home'))* Is eqivalent to **CHECK (** *PC.LOCATION = 'Accounting'* **OR** *PC.LOCATION = 'Sales'* **OR** *PC.LOCATION = 'Info Systems'* **OR** *PC.LOCATIONS = 'Home'* **)**
Primary key:	**PRIMARY KEY (***TAGNUM***)** **PRIMARY KEY (***PACKID, TAGNUM***)**
Foreign key:	**FOREIGN KEY (***COMPID***) REFERENCES** *COMPUTER*

Integrity constraints II

Example:

```
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 )
```

Integrity constraints 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*

Domains – user defined data types

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

... will be used as follows:

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

Declaration of the *LOCATION* column by means of the *LOCATIONS* domain.

Nested queries, subquery I

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

Comment: First, the subquery (aka inner query) will be evaluated. Its result will be used in the outer query.

The result of the inner query is:

AVG1
405.09

The result of the whole query is:

PACKID	PACKNAME
AC01	Boise Accounting
DB33	Manta

Nested queries, subquery II

The same result can be achieved by using an equijoin. Equijoint should be prefered before using nested queries

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

Result:

PACKNAME
Boise Accounting
Manta

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

Result:

PACKNAME
Boise Accounting
Manta

Nested queries, subquery III

IN versus EXISTS

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

Correlated subquery:

The inner query is executed for each row evaluated by the outer query again. The reason is that the value of PC.TAGNUM column is a parameter of the nested query.

Computationally extremely expensive. Should be avoided if possible.

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

This is not a correlated query. The inner query will be executed once only.

It gives the same result as the (correlated) query on the left.

Better than the correlated query, but replacing with an equijoin would be even better.

Result:

TAGNUM	COMPID
32808	M759
37691	B121
57772	C007

Result:

TAGNUM	COMPID
32808	M759
37691	B121
57772	C007

Nested select: where it can be nested

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

ALL quantifier

Textual formulation of the query:
Find an instalation of a software product that was bought for a price that is higher than current catalogue price of **any** software product.

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 )
```

Result:

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

ANY quantifier

Textual formulation of the query:
Find an instalation of a software product that was bought for a price that is higher than current catalogue price of **some** software product.

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

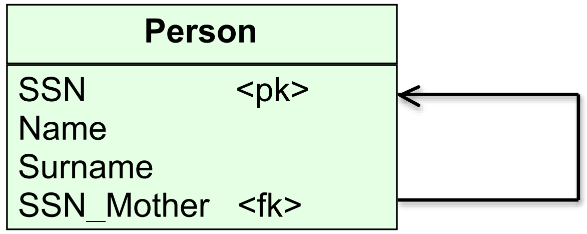
```
SELECT PACKID, TAGNUM, INSTDATE, SOFTCOST
FROM SOFTWARE
WHERE SOFTCOST > ANY
          ( SELECT PACKCOST
            FROM PACKAGE )
```

Result:

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

Usage of ALIAS

Find the name and surname of John Smith’s mother:



PERSON table needs to be opened twice. Once for the child (John Smith) and once for his potential mothers.

```
SELECT M.NAME, M.SURNAME
  FROM PERSON M JOIN PERSON CH ON (M.SSN = CH.SSN_MOTHER)
  WHERE CH.NAME = “John“ AND CH.SURNAME = “Smith“
```

Creating a copy of an existing table 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'
```

The target table *DBPACK* has to have cloumns of the same names as the source table. The corersponding columns of the source and target tables have to be compatible.

Creating a copy of an existing table 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
```

The columns of the target tabel have to be compatible with the respective columns of the source table.

VIEW I

View can be understood as a table that does not contain explicite data. This “table” is a view on another table or a relation defined as a join of multiple tables.

View is aimed at (i) reading and/or (ii) modifying data from the coresponding table(s).

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

VIEW can be

- materialized – exists independently on existence of a database connection,
- non-materialized – its existence ends on closing the database connection.

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

The cells with yellow background will form the contents of the view named *DATABASE*.

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

We can use a view similarly as a table.
In this case, the result will be the only one row:

PACKID	PACKNAME	PACKCOST
DB33	Manta	430.18

VIEW III

Columns of a view can have names that are different from the column names of the source tables.

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

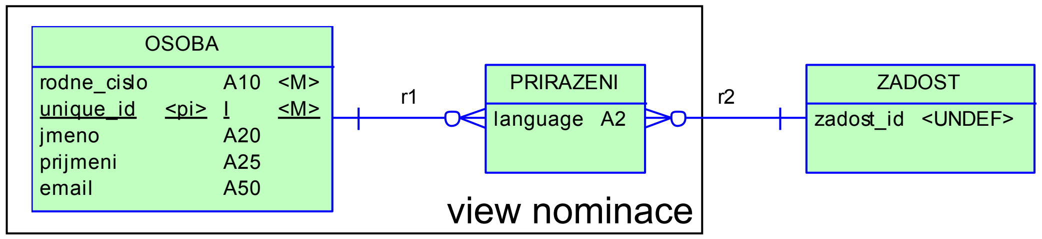
Meaning of a view:

1. Data independence.
   Modification of the source table structure that does not affect the columns participating in the view does not affect the work with the view.
2. Different views on the same data. We can hide what the user does not need to see.

Updating a view:

- When inserting to a view, modifying a view or deleting records from the view, **integrity constrains of the source tables** are checked.
- A trial to add the row ('AC01','DATAQUICK',250.00) to the *DATABASE* view has to fail, as the *PACKAGE* table already contains a row with primary key 'AC01'. It may be a surprise for the user, as he sees only rows of the view and it does not contain a row witjh primary key 'AC01'.

VIEW IV – Updatable view in PosgreSQL



View definition:

```
CREATE OR REPLACE VIEW nominace AS
SELECT osoba.rodne_cislo, osoba.jmeno, osoba.prijmeni, osoba.email, prirazeni.language, prirazeni.zadost_id
FROM osoba JOIN prirazeni ON osoba.unique_id = prirazeni.osoba_unique_id;
```

1. rule:

```
CREATE OR REPLACE RULE "_INSERT_A_FIRST" AS
ON INSERT TO nominace
WHERE
NOT (EXISTS (SELECT 1 FROM osoba WHERE osoba.rodne_cislo = new.rodne_cislo))
DO INSTEAD
INSERT INTO osoba (rodne_cislo, jmeno, prijmeni, email)
        VALUES (new.rodne_cislo, new.jmeno, new.prijmeni, new.email);
```

2. rule:

```
CREATE OR REPLACE RULE "_INSERT_Z_LAST" AS
ON INSERT TO nominace
DO INSTEAD
INSERT INTO prirazeni (zadost_is, language, opponent_unique_id)
        VALUES (new.zadost_is, new.language, ( SELECT osoba.unique_id
                                               FROM osoba
                                               WHERE osoba.rodne_cislo = new.rodne_cislo)));
```

Rules are applied in the ALPHABETIC order of tehir names !

Modification of table’s data

```
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
```

Increase the value of the *PACKCOST* column of the *PACKAGE* table by 2 percent in all records meeting the WHERE condition.

Remove the value in the *EMPPHONE* column of the *EMPLOYEE* table in all recordsmeeting the WHERE condition.

Modification of the database structure

```
ALTER TABLE EMPLOYEE
   ADD EMPTYPE CHAR(1)
```

Adding a column to an existing table.

If there is at least one reord in the table already, the new attribute has to accept NULL, as its value in the already existing rows will be NULL.

```
ALTER TABLE EMPLOYEE
   ADD EMPTYPE CHAR(1) INIT = 'H'
```

Adding a column to an existing table.

The new attribute will get the value **‘H‘** in all rows existing so far.

```
ALTER TABLE PACKAGE
   DELETE PACKVER
```

Removal of a column.

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

Change of column’s data type.

Pay attantion! Data may be lost if the „new length“ is less than the „old one“.

```
DROP TABLE COMPUTER
```

Removal of the whole table named *COMPUTER*.

Granting rights

User *JONES* will be allowed to read data from *EMPLOYEE* table.

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

Any user will be allowed to read columns *PACKID, PACKNAME a PACKTYPE* of *PACKAGE* table.

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

Users SMITH and BROWN will be allowed to insert rows to *PACKAGE* table.

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

User ANDERSON will be allowed to modify values of *EMPNAME* and *EMPHONE* of the *EMPLOYEE* table.

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

User MARTIN will be allowed to delete rows of *SOFTWARE* table.

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

User ROBERTS will be allowed to create indices for the *COMPUTER* table.

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

User THOMAS will be allowed to change structure of the *EMPLOYEE* table.

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

User WILSON will be allowed to do anything (see above) with tables *COMPUTER* and *EMPLOYEE*.

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

Revolking the access right

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

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

Indices I

advantage:	• shortening the response time (depends on the quality of the query optimizer)
	• sorting
disadvantage:	• incerases requirements on the media capacity
	• each update of a table -> update of the index (slowiing down insert and update)

Index expression = set of columns

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

Creates an index named *CUSTIND2* for the table *EMPLOYEE.* The index expression will be the singleton { *COMPID* }.

Indexy II

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

The index expression may be a set of multiple columns.

Indexy II

CREATE INDEX SOFTIND ON SOFTWARE (PACKID, TAGNUM)

The index expression may be a set of multiple columns.

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

Index may have assigned an ascending or descenting order.

Indexy III

Removal of a (not needed) index:

DROP INDEX *PACKIND*

Indexy IV

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

The index management will not allow for adding a row to the respective table if there already is a row with the respective value of the index expression in the table.

You should not rely on the uniqueness of indexes. **The index shall influence just the performance not the functionality of the database application.**

CORRECT: If a (set of) column(s) shall be unique, the respective integrity constraint shall be added to the definition of the respective table.

The reason is that the index can be created/removed by the database administrator, who does not know, whether its uniqueness is important for the correct functionality of respective databases aplication(s).