

# Database systems

## Structured query language SQL - II



# SELECT VIII

## BUILT-IN aggregation functions

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

# SELECT IX

**PACKAGE table**

<b>PACKID</b>	<b>PACKNAME</b>	<b>PACKVER</b>	<b>PACKTYPE</b>	<b>PACKCOST</b>
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:**

<b>COUNT1</b>
2

# SELECT X

**PACKAGE table:**

<b>PACKID</b>	<b>PACKNAME</b>	<b>PACKVER</b>	<b>PACKTYPE</b>	<b>PACKCOST</b>
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:**

<b>COUNT1</b>
1

# SELECT XI

**PACKAGE** table:

<b>PACKID</b>	<b>PACKNAME</b>	<b>PACKVER</b>	<b>PACKTYPE</b>	<b>PACKCOST</b>
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:**

<b>COUNT1</b>	<b>SUM2</b>
6	1968.96

# SELECT XII

**PACKAGE table:**

<b>PACKID</b>	<b>PACKNAME</b>	<b>PACKVER</b>	<b>PACKTYPE</b>	<b>PACKCOST</b>
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:**

<b>COUNT1</b>	<b>AVG2</b>
<b>6</b>	<b>328.16</b>

# SELECT XIII

**PACKAGE** table

<b>PACKID</b>	<b>PACKNAME</b>	<b>PACKVER</b>	<b>PACKTYPE</b>	<b>PACKCOST</b>
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:**

<b>COUNT1</b>	<b>MAX2</b>
6	725.83

# SELECT XIV

**PACKAGE** table

<b>PACKID</b>	<b>PACKNAME</b>	<b>PACKVER</b>	<b>PACKTYPE</b>	<b>PACKCOST</b>
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:**

<b>COUNT1</b>	<b>MIN2</b>
6	30.00



# SELECT XV

PC table

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

**DISTINCT** prohibits multiple appearance 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	INSDATE	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 groups in the output.  
 Each group 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
```

```
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
57772	583.01
59836	35.00
77740	35.00

**Result:**

G_TAGNUM	SUM1
32808	1319.95
37691	607.50

# GROUP BY II

**SOFTWARE table**

PACKID	TAGNUM	INSDATE	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 evaluated 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) simultaneously, 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	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

**SELECT \***  
**FROM PC, EMPLOYEE**

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

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 that „belong together“ are combined.

Typically, we would wish to combine rows that match 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 columns that have equal names in both tables. Then **USING** is omitted.

# JOIN VII (equijoin)

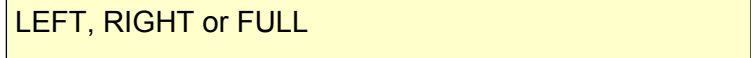
```
SELECT TAGNUM, COMPID, EMPNUM, EMPNAME  
FROM PC JOIN EMPLOYEES ON PC.EMPNUM =  
EMPLOYEES.EMPNUM
```

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 constraints I

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



# 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

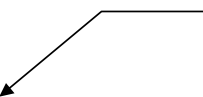
```
CREATE DOMAIN LOCATIONS CHAR(12)  
CHECK ( VALUE = 'Accounting' OR  
        VALUE = 'Sales' OR  
        VALUE = 'Info Systems' OR  
        VALUE = 'Home' )
```



... 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. Equijoin should be preferred 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.**

Result:

TAGNUM	COMPID
32808	M759
37691	B121
57772	C007

```
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

## 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 installation 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:

<i>PACKID</i>	<i>TAGNUM</i>	<i>INSTDATE</i>	<i>SOFTCOST</i>
AC01	32808	09/13/95	754.95



# ANY quantifier

## Textual formulation of the query:

Find an installation 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

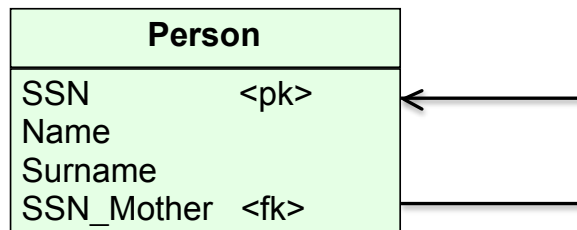
```
SELECT PACKID, TAGNUM, INSTDATE, SOFTCOST
FROM SOFTWARE
WHERE SOFTCOST > ANY
( SELECT PACKCOST
  FROM PACKAGE )
```

## 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 columns of the same names as the source table. The corresponding 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 table 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 explicit 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 corresponding 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
<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

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'
```

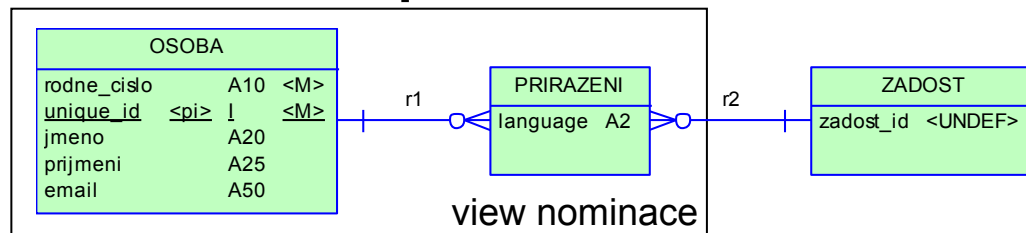
## Meaning of a view:

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 constraints 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 with 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 their names !



# Modification of table's data

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

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

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

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

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

# Modification of the database structure

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

## Adding a column to an existing table.

If there is at least one record 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 EMPLOYEE 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 attention!** 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:**
- increases requirements on the media capacity
  - each update of a table -> update of the index (slowing 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 descending 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 application(s).