See also: OLAP.mth

First VALUE

```
SELECT DISTINCT REQUEST_ID
     , JE_HEADER_ID
     , JE_LINE_NUM
        , (FIRST_VALUE(JOIN_ID)  OVER (PARTITION BY JE_HEADER_ID, JE_LINE_NUM ORDER BY SPOSOB)
) JOIN_ID
  FROM XXGLMSZ_KGW009
ORDER BY JE_HEADER_ID, JE_LINE_NUM
```

Suma narastająco

```
SELECT accounted_dr, (SUM(accounted_dr) over (PARTITION BY '1' ORDER BY creation_date))
suma_narastajaco FROM gl_je_lines WHERE code_combination_id = 4388
ORDER BY creation_date
```

Min – max w poszczególnych grupach

```
SELECT department_id,
MIN(salary) KEEP (DENSE_RANK FIRST ORDER BY commission_pct) OVER(PARTITION department_id) "W",
MAX(salary) KEEP (DENSE_RANK LAST ORDER BY commission_pct) OVER(PARTITION department_id) "B"
   FROM employees
   GROUP BY department_id;
```

```
DEPARTMENT_ID      Worst       Best
------------- ---------- ----------
           10       4400       4400
           20       6000      13000
           30       2500      11000
           40       6500       6500
           50       2100       8200
```

Numeracja elementów w grupach

```
select zam_id, cena_euro,(rank() over (partition by zam_id order by cena_euro)) from
pozycjezamowien order by zam_id

select rownum, (row_number() over (partition by null order by sygnatura) ) ordered_number from
faktury
order by sygnatura
```

Numerowanie rekordów kolejno

```
select (row_number() OVER (partition by null order by null))  from fa_retirements
```

Data do = data od -1 z następnego rekordu

```
CREATE TABLE CWF_D_LOOKUP_VALUES
(
  ID                    NUMBER                  NOT NULL,
  LOOKUPV_CODE          VARCHAR2(255 BYTE)      NOT NULL,
  LOOKUP_VALUE_CODE     VARCHAR2(30 CHAR)       NOT NULL,
  LOOKUP_TYPE_CODE      VARCHAR2(20 BYTE)       NOT NULL,
  VALUE                 VARCHAR2(255 BYTE),
  DESCRIPTION           VARCHAR2(1000 BYTE),
  ATTRIBUTE1            VARCHAR2(1000 BYTE),
  ATTRIBUTE2            VARCHAR2(1000 BYTE),
  ATTRIBUTE3            VARCHAR2(1000 BYTE),
```

```
  ATTRIBUTE4            VARCHAR2(1000 BYTE),
  ATTRIBUTE5            VARCHAR2(1000 BYTE),
  START_DATE            DATE                NOT NULL,
  END_DATE              DATE,
  CREATED_BY_LOGIN      VARCHAR2(255 BYTE)  NOT NULL,
  CREATED_BY_IP         VARCHAR2(15 BYTE)   NOT NULL,
  CREATION_DATE         TIMESTAMP(9)        NOT NULL,
  LAST_UPDATED_BY_LOGIN VARCHAR2(255 BYTE)  NOT NULL,
  LAST_UPDATED_BY_IP    VARCHAR2(15 BYTE)   NOT NULL,
  LAST_UPDATE_DATE      TIMESTAMP(9)        NOT NULL,
  SEQNUM                NUMBER,
  ACTIVE_FLAG           VARCHAR2(1 BYTE)
;

Insert into FLEX_COL_USAGE  (START_DATE, LOOKUP_TYPE_CODE, LOOKUP_VALUE_CODE, DO_DNIA, DESCRIPTION) Values  (TO_DATE('03/01/2010 00:00:00', 'MM/DD/YYYY HH24:MI:SS'), 'BIK01',
'LOOKUP_VALUE_CODE', TO_DATE('04/30/2010 00:00:00', 'MM/DD/YYYY HH24:MI:SS'), 'WRROEKXHDVKMEYG');
Insert into FLEX_COL_USAGE  (START_DATE, LOOKUP_TYPE_CODE, LOOKUP_VALUE_CODE, DO_DNIA, DESCRIPTION) Values  (TO_DATE('05/01/2010 00:00:00', 'MM/DD/YYYY HH24:MI:SS'), 'BIK01',
'LOOKUP_VALUE_CODE', TO_DATE('06/30/2010 00:00:00', 'MM/DD/YYYY HH24:MI:SS'), 'NGHDPQOEHVULYXW');
Insert into FLEX_COL_USAGE  (START_DATE, LOOKUP_TYPE_CODE, LOOKUP_VALUE_CODE, DO_DNIA, DESCRIPTION) Values  (TO_DATE('07/01/2010 00:00:00', 'MM/DD/YYYY HH24:MI:SS'), 'BIK01',
'LOOKUP_VALUE_CODE', TO_DATE('12/31/4712 00:00:00', 'MM/DD/YYYY HH24:MI:SS'), 'BYIHLQAYOIELSPN');
COMMIT;


select start_date
     , nvl(lead(start_date-1)over(partition by lookup_type_code order by start_date),date'4712-12-31') end_date
     , lookup_type_code, lookup_value_code
     , description
  from cwf_d_lookup_values
 where 'BIK01' = lookup_type_code
   and 'LOOKUP_VALUE_CODE' = lookup_value_code
   order by start_date
```

START_DATE	END_DATE	LOOKUP_TYPE_CODE	LOOKUP_VALUE_CODE	DESCRIPTION
2010-03-01	2010-04-30	BIK01	LOOKUP_VALUE_CODE	WRROEKXHDVKMEYG
2010-05-01	2010-06-30	BIK01	LOOKUP_VALUE_CODE	NGHDPQOEHVULYXW
2010-07-01	4712-12-31	BIK01	LOOKUP_VALUE_CODE	BYIHLQAYOIELSPN

Jest też funkcja LAG

Suma z ominięciem powtarzających się elementów

```
CREATE TABLE XXTESTANALYT
( ID   NUMBER,
  VAL  NUMBER
);

Insert into XXTESTANALYT  (ID, VAL) Values  (1, 2);
Insert into XXTESTANALYT  (ID, VAL) Values  (1, 2);
Insert into XXTESTANALYT  (ID, VAL) Values  (1, 2);
Insert into XXTESTANALYT  (ID, VAL) Values  (2, 3);
Insert into XXTESTANALYT  (ID, VAL) Values  (2, 3);
Insert into XXTESTANALYT  (ID, VAL) Values  (3, 4);
Insert into XXTESTANALYT  (ID, VAL) Values  (4, 1);
Insert into XXTESTANALYT  (ID, VAL) Values  (4, 1);
Insert into XXTESTANALYT  (ID, VAL) Values  (4, 1);
COMMIT;

select id, val
-- ostatnia wartosc (LAST_VALUE) z zakresu (OVER) miêdzy 1 a id-1, po
posortwaniu danych wg id
-- PARTITION by null mo¿na omin¹c
, last_value(val) OVER (PARTITION by null ORDER BY id RANGE BETWEEN id-1
PRECEDING AND 1 PRECEDING) analyt_alone
, val+nvl(last_value(val) OVER (ORDER BY id RANGE BETWEEN id-1 PRECEDING
AND 1 PRECEDING),0) analyt
from xxtestanalyt
```

zamiast PRECEDING można też napisać FOLLOWING

ID	VAL	ANALYT_ALONE	ANALYT
1	2		2
1	2		2
1	2		2
2	3	2	5
2	3	2	5
3	4	3	7
4	1	4	5
4	1	4	5
4	1	4	5

KLAUZULE GROUP BY, GROUP BY CUBE, GROUP BY ROLLUP

Klauzula	Opis	Zastosowanie
GROUP BY A,B,C	Grupuje dane po kolumnach (A,B,C)	Podsumowania
GROUP BY ROLLUP(A,B,C)	„zwija" wymiary od końca tj. grupuje dane wg: ABC, AB, A, NULL	Podsumowania po poszczególnych poziomach hierarchii, np.: Szczegółowo Wg komórek Wg pionu Ogółem
GROUP BY CUBE(A,B,C)	Grupuje dane wszystkich kombinacjach kolumn A,B,C tj. ABC, AB,AC,BC,A,B,C,NULL Aby sprawdzić, po których kolumnach następuje grupowanie, użyj funkcji GROUPING, np. GROUPING(A) = 1 OZNACZA, ZE ZESTAWIENIE JEST BEZ UZWGLEDNIANIA KOLUMNY A Zamiennie z GROUP BY CUBE można uzywać po prostu iloczynu kartezjańskiego kilku tabel po dodaniu do każdej tabeli wartości null. Składnie CUBE jest bardziej zwięzła. **W testach otrzymywałem ora-600 (Oracle R11R1) i wróciłem do zwykłego iloczynu kartezjańskiego.**	W planowaniu zajęć do wyznaczenia wszystkich kombinacji obiektów a następnie sprawdzenie uprawnień dla każdej kombinacji.

Przykład poniżej – kluczowe w przykładach są komórki oznaczone kolorem błękitnym.

```
CREATE TABLE XXCUBE ( LECTURER VARCHAR2(100), GROUP_NAME VARCHAR2(100), SUBJECT
VARCHAR2(100));
Insert into XXCUBE (LECTURER, GROUP_NAME, SUBJECT) Values   ('ABACKI', 'C11', 'MATH');
Insert into XXCUBE (LECTURER, GROUP_NAME, SUBJECT) Values   ('ABACKI', NULL, 'MATH');
Insert into XXCUBE (LECTURER, GROUP_NAME, SUBJECT) Values   (NULL, 'C11', 'MATH');
Insert into XXCUBE (LECTURER, GROUP_NAME, SUBJECT) Values   (NULL, NULL, 'MATH');
Insert into XXCUBE (LECTURER, GROUP_NAME, SUBJECT) Values   ('ABACKI', 'C11', 'PHIS');
Insert into XXCUBE (LECTURER, GROUP_NAME, SUBJECT) Values   ('ABACKI', NULL, 'PHIS');
Insert into XXCUBE (LECTURER, GROUP_NAME, SUBJECT) Values   (NULL, 'C11', 'PHIS');
Insert into XXCUBE (LECTURER, GROUP_NAME, SUBJECT) Values   (NULL, NULL, 'PHIS');
COMMIT;

SELECT * FROM XXCUBE
```

LECTURER	GROUP_NAME	SUBJECT

ABACKI	C11	MATH
ABACKI	NULL	MATH
NULL	C11	MATH
NULL	NULL	MATH
ABACKI	C11	PHIS
ABACKI	NULL	PHIS
NULL	C11	PHIS
NULL	NULL	PHIS

```
SELECT LECTURER, GROUP_NAME, SUBJECT, COUNT(*), GROUPING(LECTURER), GROUPING(GROUP_NAME),
GROUPING(SUBJECT)
FROM XXCUBE
GROUP BY LECTURER, GROUP_NAME, SUBJECT
ORDER BY COUNT(*)
```

LECTURER	GROUP_NAME	SUBJECT	COUNT(*)	GROUPING(LECTURER)	GROUPING(GROUP_NAME)	GROUPING(SUBJECT)
null	null	MATH	1	0	0	0
null	null	PHIS	1	0	0	0
null	C11	MATH	1	0	0	0
null	C11	PHIS	1	0	0	0
ABACKI	null	MATH	1	0	0	0
ABACKI	null	PHIS	1	0	0	0
ABACKI	C11	MATH	1	0	0	0
ABACKI	C11	PHIS	1	0	0	0

```
SELECT LECTURER, GROUP_NAME, SUBJECT, COUNT(*), GROUPING(LECTURER), GROUPING(GROUP_NAME),
GROUPING(SUBJECT)
FROM XXCUBE
GROUP BY CUBE(LECTURER, GROUP_NAME, SUBJECT)
ORDER BY GROUPING(LECTURER) *4 + GROUPING(GROUP_NAME) *2 + GROUPING(SUBJECT) *1, COUNT(*)
```

LECTURER	GROUP_NAME	SUBJECT	COUNT(*)	GROUPING(LECTURER)	GROUPING(GROUP_NAME)	GROUPING(SUBJECT)
null	null	MATH	1	0	0	0
null	null	PHIS	1	0	0	0
null	C11	MATH	1	0	0	0
ABACKI	null	MATH	1	0	0	0
ABACKI	null	PHIS	1	0	0	0
ABACKI	C11	PHIS	1	0	0	0
ABACKI	C11	MATH	1	0	0	0
null	C11	PHIS	1	0	0	0
null	null	null	2	0	0	1
ABACKI	null	null	2	0	0	1
ABACKI	C11	null	2	0	0	1
null	C11	null	2	0	0	1
null	null	MATH	2	0	1	0
null	null	PHIS	2	0	1	0
ABACKI	null	MATH	2	0	1	0
ABACKI	null	PHIS	2	0	1	0
null	null	null	4	0	1	1
ABACKI	null	null	4	0	1	1
null	null	MATH	2	1	0	0
null	C11	PHIS	2	1	0	0
null	null	PHIS	2	1	0	0
null	C11	MATH	2	1	0	0
null	null	null	4	1	0	1
null	C11	null	4	1	0	1

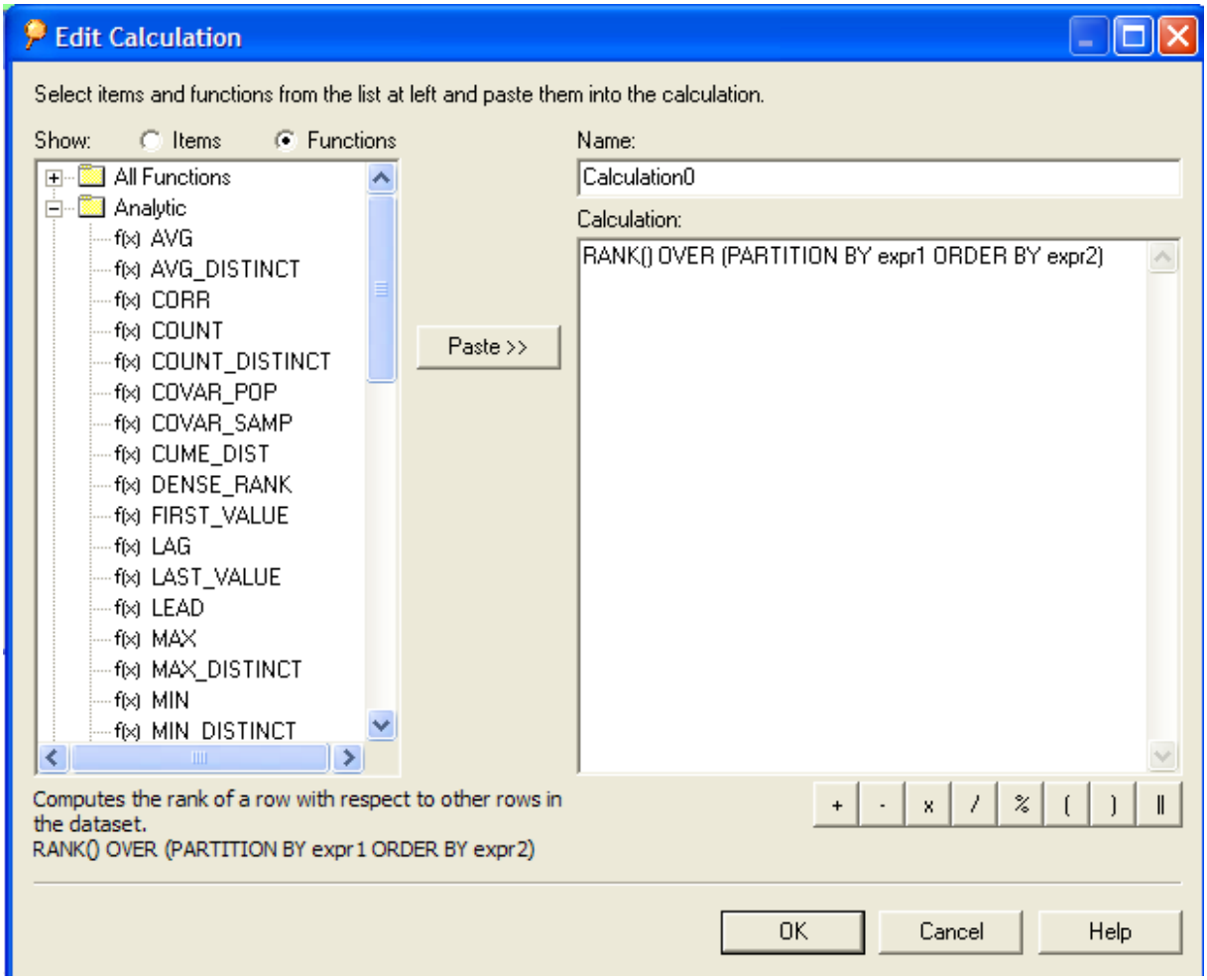
null	null	MATH	4	1	1	0
null	null	PHIS	4	1	1	0
null	null	null	8	1	1	1

```
SELECT LECTURER, GROUP_NAME, SUBJECT, COUNT(*), GROUPING(LECTURER), GROUPING(GROUP_NAME),
GROUPING(SUBJECT)
FROM XXCUBE
GROUP BY ROLLUP(LECTURER, GROUP_NAME, SUBJECT)
ORDER BY GROUPING(LECTURER) *4 + GROUPING(GROUP_NAME) *2 + GROUPING(SUBJECT) *1, COUNT(*)
```

LECTURER	GROUP_NAME	SUBJECT	COUNT(*)	GROUPING(LECTURER)	GROUPING(GROUP_NAME)	GROUPING(SUBJECT)
null	null	MATH	1	0	0	0
null	null	PHIS	1	0	0	0
null	C11	MATH	1	0	0	0
null	C11	PHIS	1	0	0	0
ABACKI	null	MATH	1	0	0	0
ABACKI	null	PHIS	1	0	0	0
ABACKI	C11	PHIS	1	0	0	0
ABACKI	C11	MATH	1	0	0	0
null	null	null	2	0	0	1
null	C11	null	2	0	0	1
ABACKI	C11	null	2	0	0	1
ABACKI	null	null	2	0	0	1
null	null	null	4	0	1	1
ABACKI	null	null	4	0	1	1
null	null	null	8	1	1	1

MATERIAŁ ROBOCZY

Screen z discoverera



sumy zamówień

SELECT ZAM_ID, SUM(CENA_EURO) OVER (PARTITION BY ZAM_ID ORDER BY ZAM_ID)
FROM

POZYCJE_ZAMOWIEN

```
select zam_id, sum(cena_euro) OVER (PARTITION BY zam_id ORDER BY zam_id) from pozycjezamowien
```

	ZAM_ID	SUM(CENA_EURO)OVER(PARTITIONBY
1	3761	185,95
2	3761	185,95
3	3761	185,95
4	3761	185,95
5	3761	185,95
6	3761	185,95
7	3761	185,95
8	3761	185,95
9	3761	185,95
10	3761	185,95
11	3761	185,95
12	3761	185,95
13	3761	185,95
14	3761	185,95
15	3761	185,95
16	3761	185,95
17	3761	185,95
18	3761	185,95
19	3761	185,95
20	3761	185,95
21	3761	185,95
22	3761	185,95
23	3763	14,38
24	3763	14,38
25	3763	14,38
26	3764	50,4
27	3764	50,4

ANALITIC FUNCTIONS - EXAMPLE OF USE

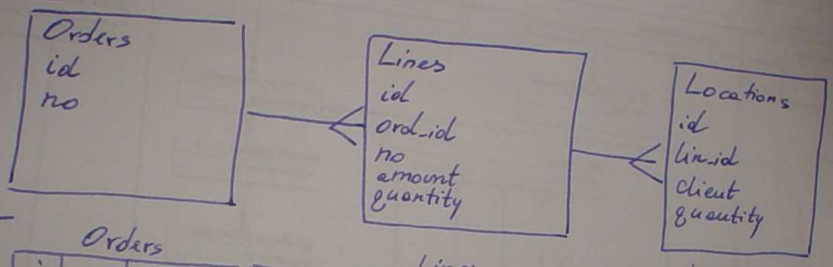
Zaremowane fragmenty dotyczą pułapki wachlarzowej – do tego przykładu nic nowego to nie wnosi.

Example data
==========

```
drop table orders;
drop table lines;
drop table locations;
--drop table invoices;

--create table invoices ( id number, no varchar2(10), ord_id number);

create table orders ( id number, no varchar2(10));

create table lines     ( ord_id number, id number, no varchar2(10), amount number, quantity
number);

create table locations ( lin_id number, id number, client varchar2(10), quantity number);


insert into orders (id, no) values ( 1, 'z1');

insert into lines  ( id, ord_id,  no, amount, quantity) values (1,1,1,10,2);
insert into locations ( id, lin_id, client, quantity) values ( 1,1,'a',1);
insert into locations ( id, lin_id, client, quantity) values ( 2,1,'b',1);

insert into lines  ( id, ord_id,  no, amount, quantity) values (2,1,2,20,2);
insert into locations ( id, lin_id, client, quantity) values ( 3,2,'c',1);
insert into locations ( id, lin_id, client, quantity) values ( 4,2,'d',1);

insert into lines  ( id, ord_id,  no, amount, quantity) values (3,1,3,30,2);
insert into locations ( id, lin_id, client, quantity) values ( 5,3,'e',1);
insert into locations ( id, lin_id, client, quantity) values ( 6,3,'f',1);

insert into orders (id, no) values ( 2, 'z2');
```

```
insert into lines  ( id, ord_id,  no, amount, quantity) values (4,2,1,40,2);
insert into locations ( id, lin_id, client, quantity) values ( 7,4,'g',1);
insert into locations ( id, lin_id, client, quantity) values ( 8,4,'h',1);

insert into lines  ( id, ord_id,  no, amount, quantity) values (5,2,2,50,2);
insert into locations ( id, lin_id, client, quantity) values ( 9,5,'i',1);
insert into locations ( id, lin_id, client, quantity) values (10,5,'j',1);

insert into lines  ( id, ord_id,  no, amount, quantity) values (6,2,3,60,2);
insert into locations ( id, lin_id, client, quantity) values (11,6,'k',1);
insert into locations ( id, lin_id, client, quantity) values (12,6,'l',1);

--insert into invoices ( id, ord_id, no ) values (1,1, '1');
--insert into invoices ( id, ord_id, no ) values (2,1, '2');

--insert into invoices ( id, ord_id, no ) values (3,2, '3');
--insert into invoices ( id, ord_id, no ) values (4,2, '4');

QUERIES
========

-- to zapytanie zwróci 24 w kolumnie sum (lin.QUANTITY) , co jest nieprawda, bo powinno wyjsc
12  ( rekordy podrzedne podwoily ilosc )
select sum (lin.QUANTITY), sum (loc.QUANTITY)
  from orders    ord
,      lines     lin
,      locations loc
--,      invoices  inv
where  lin.ord_id = ord.id
  and  loc.lin_id = lin.id
--  and  inv.ORD_ID = ord.id

-- to zapytanie zachowa sie prawidlowo - zwroci 12 w kolumnie sum (lin.QUANTITY)
select  ord.no
,      sum (lin.quantity) over (partition by lin.ord_id)
,      sum (loc.quantity) over (partition by lin.ord_id)
  from orders    ord
,      lines     lin
,      locations loc
--,      invoices  inv
where  lin.ord_id = ord.id
  and  loc.lin_id = lin.id
--  and  inv.ORD_ID = ord.id
```

Model



Data

Orders

id	no
1	Z1
2	Z2

Lines

id	amount	ord_id	no amount	quant
1	10	1	1	2
2	20	1	2	2
3	30	1	3	2
4	40	2	1	2
5	50	2	2	2
6	60	2	3	2

Locations

id	lin_id	client	quant
1	1	a	1
2	1	b	1
3	2	c	1
4	2	d	1
5	3	e	1
6	3	f	1
7	4	g	1
8	4	h	1
9	5	i	1
10	5	j	1
11	6	k	1
12	6	l	1