

Part 1. Introduction, data models and partitioned outer join

DB connection

HR schema:

- Login: HR
- Password: HR
- Server: evuakyisd0228
- Port: 1521
- Service name: orcl

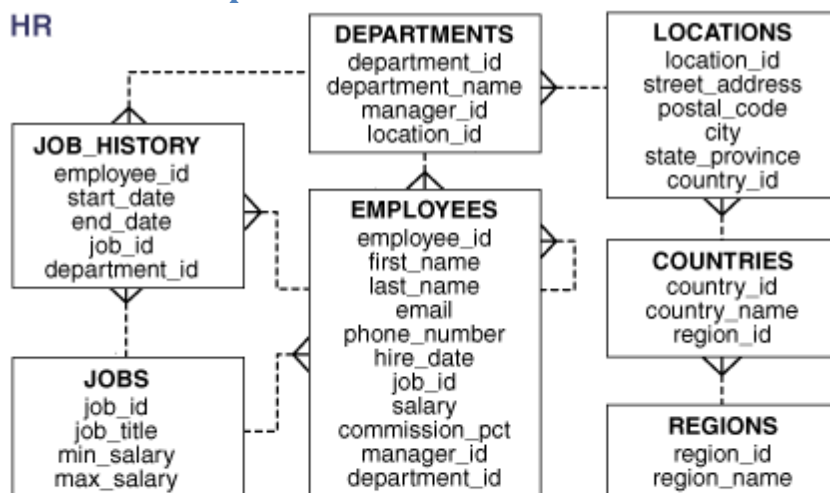
Connection string: HR/HR@evuakyisd0228:1521/orcl

SH schema:

- Login: SH
- Password: SH
- Server: evuakyisd0228
- Port: 1521
- Service name: orcl

Connection string: SH/SH@evuakyisd0228:1521/orcl

HR schema description



COUNTRIES table.

Column name	Null ?	Data type
COUNTRY_ID	NOT NULL	CHAR (2)
COUNTRY_NAME		VARCHAR2 (40)
REGION_ID		NUMBER

DEPARTMENTS table.

Column name	Null ?	Data type
-------------	--------	-----------

DEPARTMENT_ID	NOT NULL	NUMBER (4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2 (30)
MANAGER_ID		NUMBER (6)
LOCATION_ID		NUMBER (4)

EMPLOYEES table.

Column name	Null ?	Data type
EMPLOYEE_ID	NOT NULL	NUMBER (6)
FIRST_NAME		VARCHAR2 (20)
LAST_NAME	NOT NULL	VARCHAR2 (25)
EMAIL	NOT NULL	VARCHAR2 (20)
PHONE_NUMBER		VARCHAR2 (20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2 (10)
SALARY		NUMBER (8, 2)
COMMISSION_PCT		NUMBER (2, 2)
MANAGER_ID		NUMBER (6)
DEPARTMENT_ID		NUMBER (4)

JOBS table.

Column name	Null ?	Data type
JOB_ID	NOT NULL	VARCHAR2 (10)
JOB_TITLE	NOT NULL	VARCHAR2 (35)
MIN_SALARY		NUMBER (6)
MAX_SALARY		NUMBER (6)

JOB_HISTORY table.

Column name	Null ?	Data type
EMPLOYEE_ID	NOT NULL	NUMBER (6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2 (10)
DEPARTMENT_ID		NUMBER (4)

LOCATIONS table.

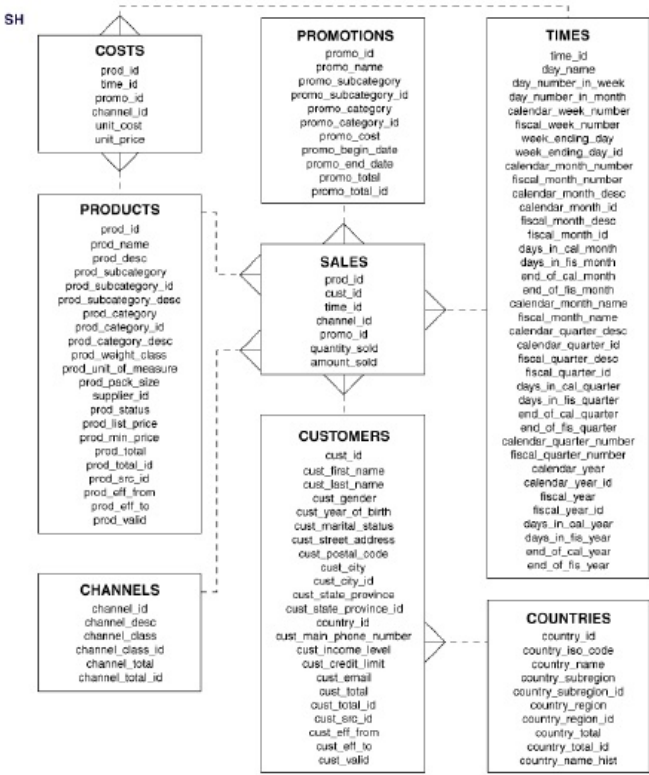
Column name	Null ?	Data type
LOCATION_ID	NOT NULL	NUMBER (4)
STREET_ADDRESS		VARCHAR2 (40)
POSTAL_CODE		VARCHAR2 (12)
CITY	NOT NULL	VARCHAR2 (30)
STATE_PROVINCE		VARCHAR2 (25)
COUNTRY_ID		CHAR (2)

REGIONS table.

Column name	Null ?	Data type
-------------	--------	-----------

REGION ID	NOT NULL	NUMBER
REGION NAME		VARCHAR2 (25)

SH schema description



SALES_VIEW view.

Column name	Data type
COUNTRY	VARCHAR2
PRODUCT	VARCHAR2
YEAR	NUMBER
SALES	NUMBER
CNT	NUMBER
BEST_YEAR	NUMBER
WORST_YEAR	NUMBER

Partitioned outer join

Main idea: convert sparse data into dense by executing separate outer join for each “partition” and then uniting the result.

Sparse data

DAY_NUMBER	PROD_NAME	CNT
1	Extension Cable ...	1171
2	Extension Cable ...	1183
3	Extension Cable ...	1109
4	Extension Cable ...	983
2	Y Box ...	1197
3	Y Box ...	934
4	Y Box ...	616
6	Y Box ...	1008
7	Y Box ...	1042

Dense data

DAY_NUMBER_IN_WEEK	PROD_NAME	CNT
1	Extension Cable ...	1171
2	Extension Cable ...	1183
3	Extension Cable ...	1109
4	Extension Cable ...	983
5	Extension Cable ...	0
6	Extension Cable ...	0
7	Extension Cable ...	0
1	Y Box ...	0
2	Y Box ...	1197
3	Y Box ...	934
4	Y Box ...	616
5	Y Box ...	0
6	Y Box ...	1008
7	Y Box ...	1042

```
SELECT select_expression
FROM   table_reference
      LEFT OUTER JOIN table_reference
      PARTITION BY (expr [,expr ]...)
```

```
SELECT select_expression
FROM   table_reference
      PARTITION BY (expr [, expr ]... )
      RIGHT OUTER JOIN table_reference
```

Useful links

- [Oracle 11g R2 docs](#)
- [SQL reference](#)
- [DWH Guide, part V](#)
- [Partitioned join in DWH Guide](#)
- [Oracle tutorial](#)

Practice

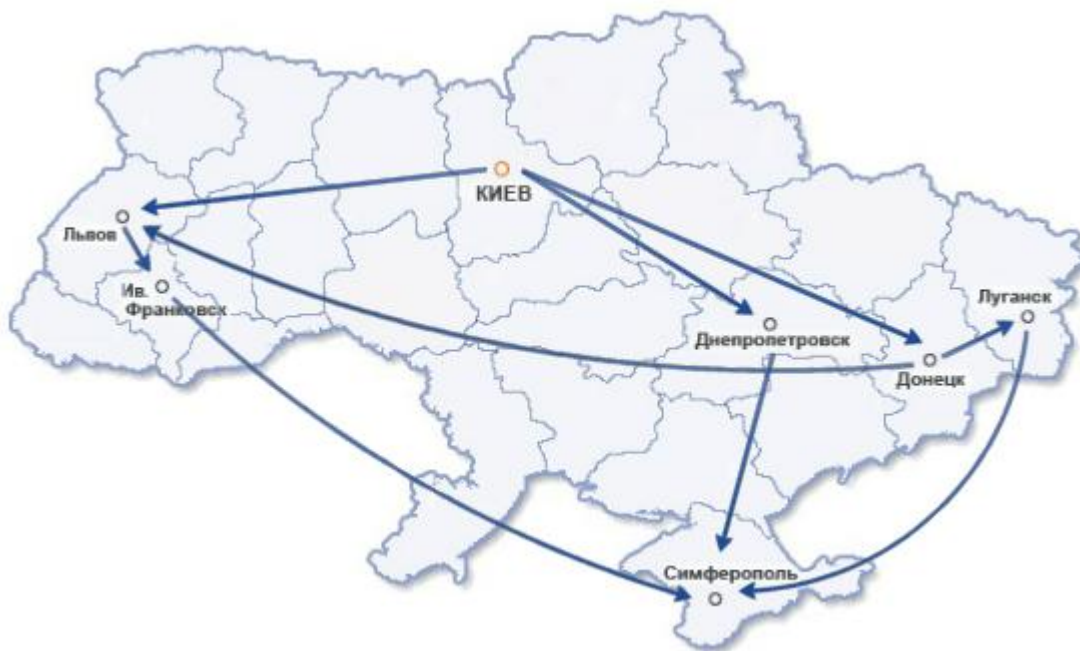
1. Create report in Sales History schema. For each calendar week of year 1998 show number of sold items of each product starting with "Multimedia". Report should be "dense". See example below.

PROD_NAME	CALENDAR_WEEK_NUMBER	CNT
Multimedia speakers- 3" cones ...	1	0
Multimedia speakers- 3" cones ...	2	10
Multimedia speakers- 3" cones ...	3	78
...
Multimedia speakers- 3" cones ...	52	30
Multimedia speakers- 3" cones ...	53	25
Multimedia speakers- 5" cones ...	1	0
Multimedia speakers- 5" cones ...	2	51
...
Multimedia speakers- 5" cones ...	51	9
Multimedia speakers- 5" cones ...	52	20
Multimedia speakers- 5" cones ...	53	22

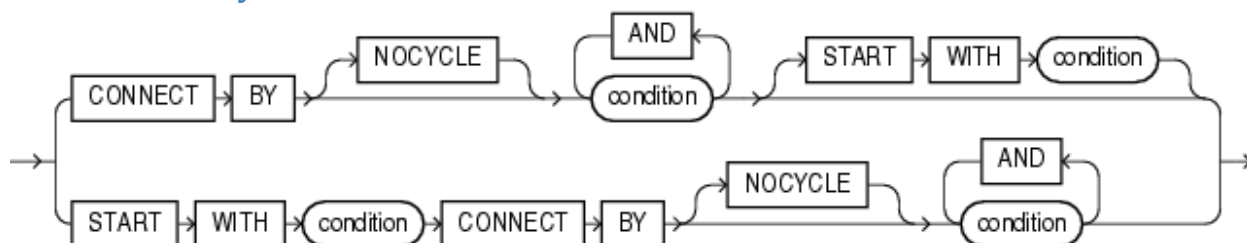
Part 2. Hierarchical queries with CONNECT BY

Railways data model

```
create table cities(  
    city          varchar2(50 char)  
    ,region_code  varchar2(10 char)  
);  
  
create table trains(  
    train_code     varchar2(10 char)      not null  
    ,origin        varchar2(50 char)      not null  
    ,destination   varchar2(50 char)      not null  
    ,duration_min  number(5)              not null  
);
```



CONNECT BY syntax



Order of hierarchical query execution

1. joins;
2. start with/connect by;
3. where;
4. group by and having;

5. order by.

Features

- **PRIOR** keyword can be used for previous level columns reference in CONNECT BY and SELECT clauses;
- **LEVEL** pseudocolumn returns hierarchy level, starting with 1;
- **SYS_CONNECT_BY_PATH** function concatenates string expressions at each level of hierarchy;
- **ORDER SIBLINGS BY** clause can be used to preserve tree structure and sort nodes at each level;
- **CONNECT BY NOCYCLE** keyword allows hierarchical queries with cycles in hierarchy;
- **CONNECT_BY_ISSYCLE** pseudocolumn is 1 for nodes having children that are their ancestor as well, 0 for others;
- **CONNECT_BY_ISLEAF** pseudocolumn is 1 for nodes having no children, 0 for others;
- **CONNECT_BY_ROOT** operator is like PRIOR, but references hierarchy root, not previous level.

Useful links

- [Hierarchical Queries](#)
- [Hierarchical Query Pseudocolumns](#)
- [Trees in SQL by Joe Celko](#)

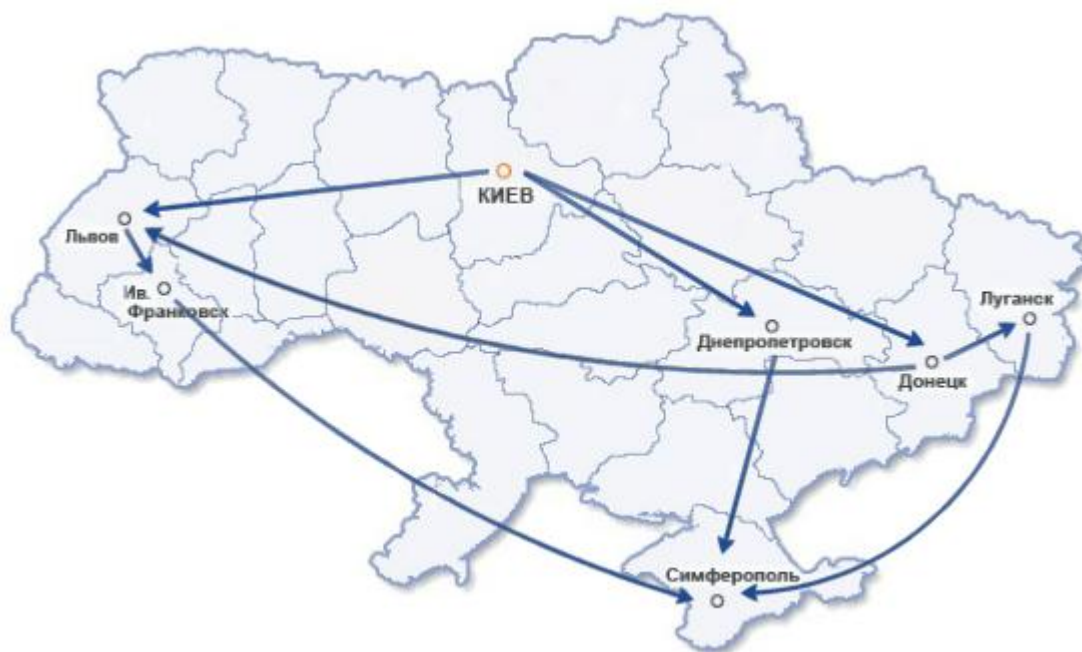
Practice (HR schema)

1. Select all subordinates of Steven King;
2. Select all managers of employee with phone_number = '650.505.2876'. Don't show that employee himself.
3. For each 'Programmer' show list of his managers from top to bottom (like '/King/..../TheProgrammer').
4. List all second-level subordinates of Steven King (direct subordinates are first-level subordinates).
5. For each employee show his salary and total salary of all his managers. Preserve tree structure in output.
6. Generate list of dates from sysdate to last day of the year.

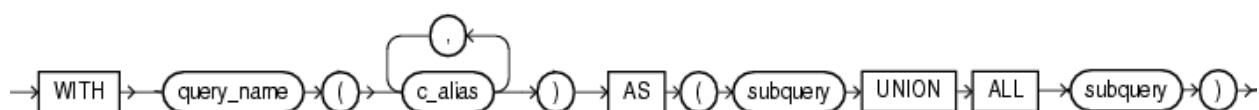
Part 3. Hierarchical queries and recursive WITH

Railways data model

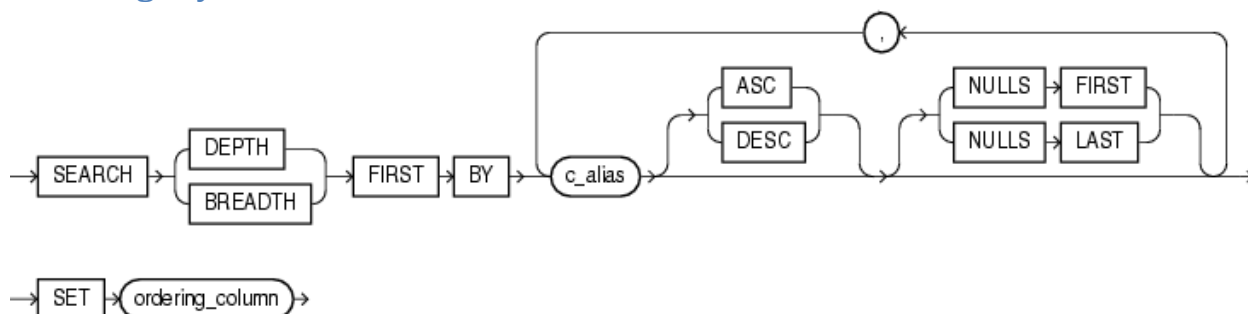
```
create table cities(  
    city          varchar2(50 char)  
    ,region_code  varchar2(10 char)  
);  
  
create table trains(  
    train_code    varchar2(10 char)      not null  
    ,origin       varchar2(50 char)      not null  
    ,destination  varchar2(50 char)      not null  
    ,duration_min number(5)              not null  
);
```



Basic syntax



Ordering syntax



Cycles processing syntax



Useful links

- [Documentation](#)
- [Examples](#)
- [Explanation by Владимир Пржиялковский](#)

Practice (HR schema)

1. Select all subordinates of Steven King;
2. Select all managers of employee with phone_number = '650.505.2876'. Don't show that employee himself.
3. For each 'Programmer' show list of his managers from top to bottom (like '/King/..../TheProgrammer').
4. List all second-level subordinates of Steven King (direct subordinates are first-level subordinates).
5. For each employee show his salary and summary salary of all his managers. Preserve tree structure in output.
6. Generate list of dates from sysdate to last day of the year.

Part 4. Analytic functions. Basic syntax.

Typical tasks

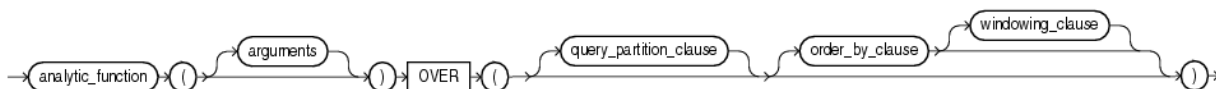
Compare these three types of Oracle functions:

- Single-row functions (like SUBSTR). Can use values from one row only.
- Aggregate functions (like SUM or COUNT). Can use values from group of rows, but “loss” individual rows.
- Analytic functions. For each individual row some group-based calculation can be done.

Typical tasks for analytic functions are:

- Convenient data output (individual value + aggregates in one row);
- Ranking;
- Cumulative totals;
- Sliding averages;
- Questions about previous or next row.
- etc.

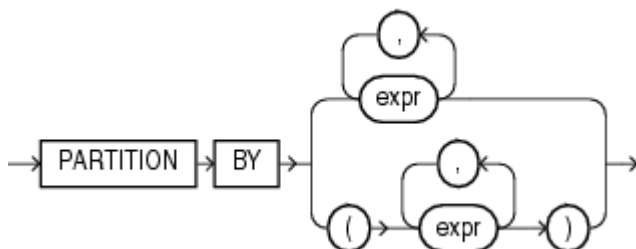
Common syntax



Partition clause

Allows to break data set into several groups. Analytic function will treat each partition is independent group and will not consider rows from other partitions.

Is not required.

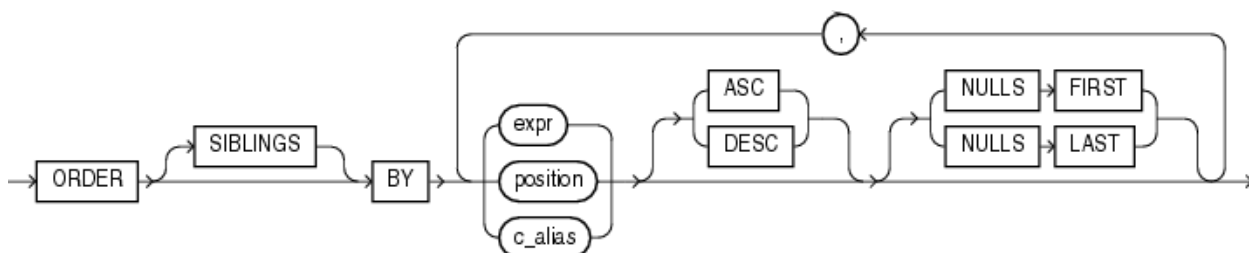


Order By clause

Allows to sort data set and:

- Answer questions about first/last/previous/next row in data set (or partition);
- Define sliding windows.

Is required is you use Windowing Clause. Can be used with or without Partition Clause.



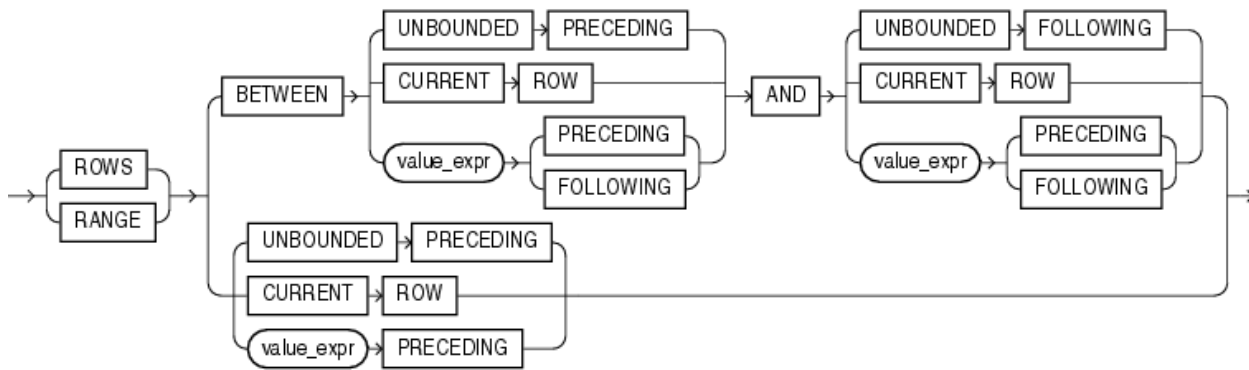
Window clause

Defines sliding window over ordered data set. Analytic function will “see” only rows inside the windows.

Can be used with Order By clause only.

Two types of windows are:

- Row-based. Such window set physical offset using number of rows. Like “2 previous and 2 next rows”;
- Range-based. Such windows set logical offset based on ordering criteria value. For data set ordered by salary in can be like “rows where salary is in the range “salary of current row +/- 100)



Useful links

- [Analytic functions](#)
- "Oracle для профессионалов" Тома Кайта, глава 12.
- "Pro Oracle SQL", глава 8.

Practice

1. For each employee show his salary and three average salaries: for all employees, for his job and for his department;
2. For each employee show number of people who had already been working in his department when he was hired.
3. For each employee show average salary of people hired in the same year;
4. For each employee show his hire_date, salary and average salary of five people: himself, two people hired right before him and two people hired right after;
5. For each employee show total salary of people hired 90 days before or after him.

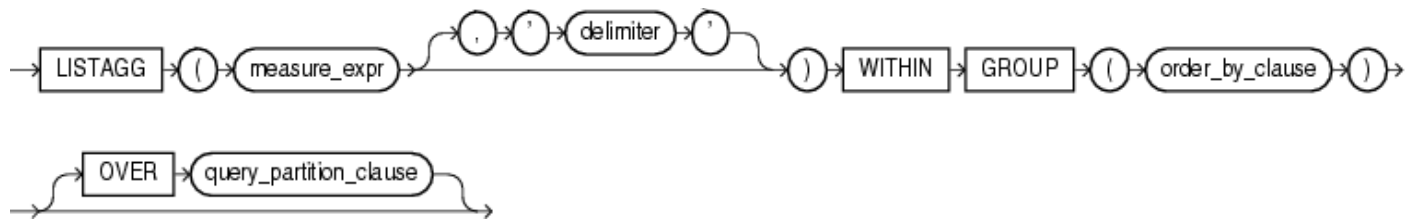
Part 5. Analytic functions. Functions.

Aggregation

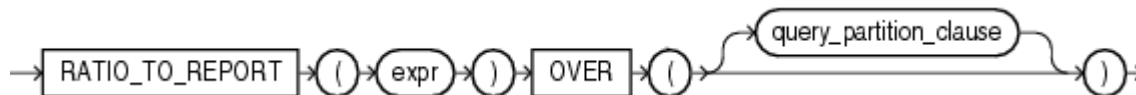
COUNT, SUM, AVG, MAX, MIN



LISTAGG

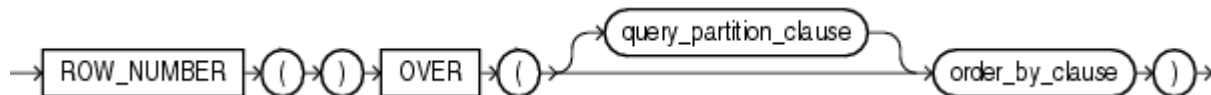


RATIO_TO_REPORT

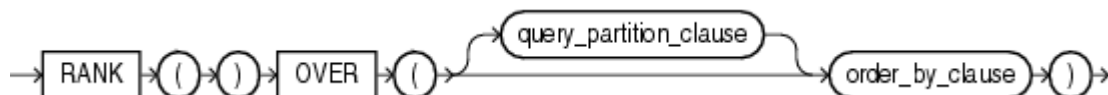


Ranking

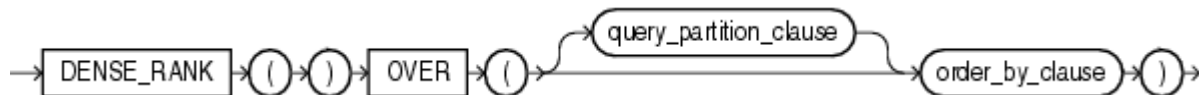
ROW_NUMBER



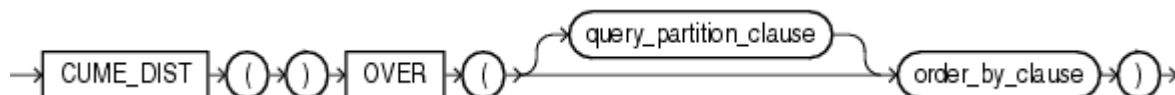
RANK



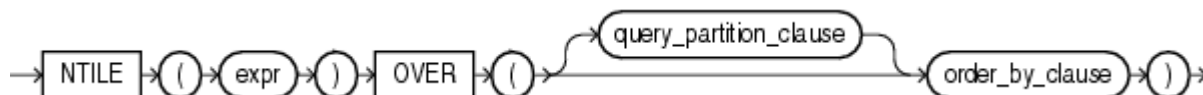
DENSE_RANK



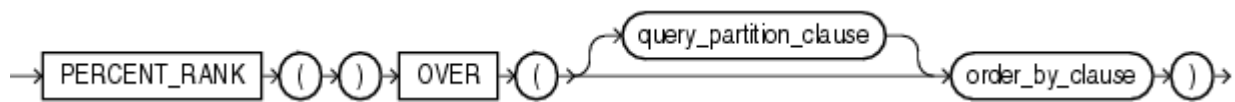
CUME_DIST



NTILE

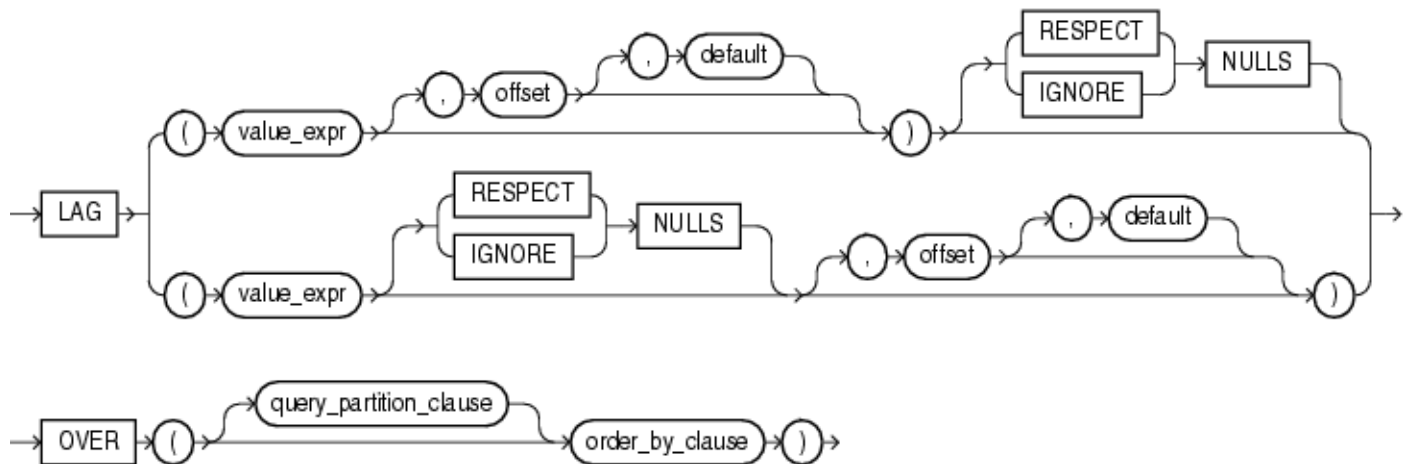


PERCENT_RANK

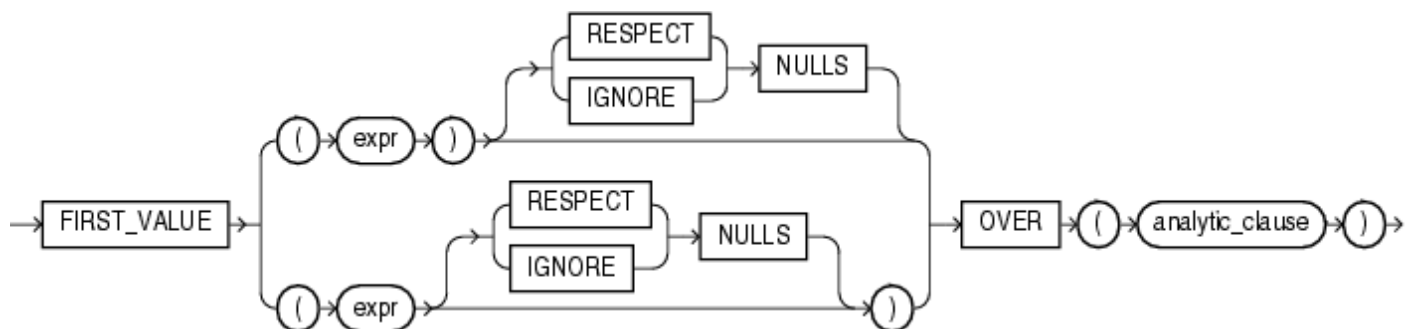


Offset

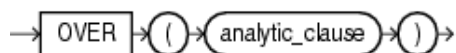
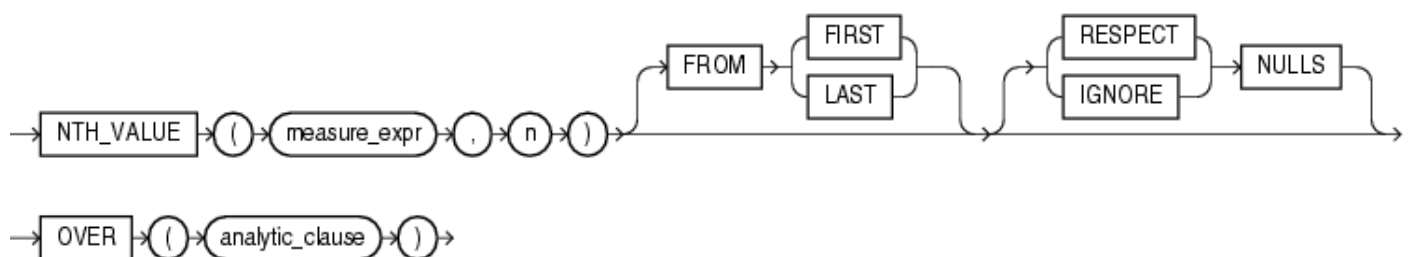
LAG and LEAD



FIRST_VALUE and LAST_VALUE



NTH_VALUE



Useful links

- [Analytic functions](#)

Practice

1. Typical top-N query can be understood as:
 - a. return exactly N rows. If there's a "tail" - ignore it;
 - b. return all records with one of first N values;
 - c. return N rows + tail if it exists;
 - d. return not more than N rows (if last value has tail - don't get rows with last value);Implement each of these approaches (EMP table).
2. For each employee from EMPLOYEES table show last name of the first employees hired in the same year and last employee, hired in the same year.
3. For each employee, show comma-separated list of people managed by the same manager.
4. Select 20% of least-paid people from EMPLOYEES table.

Part 6. Analytic functions. Practice.

Task 1

Table TUMBLER stores "turn on" and "turn off" events for some machine:

- EVENT_TIME - date and time of event
- EVENT_TYPE - "ON" of "OFF"

Calculate how much time (in days) machine were ON and OFF during period represented in TUMBLER table

Note: "ON" event can be followed by "OFF" event only and vice versa.

Initial data set

EVENT_TIME	EVENT_TYPE
01.01.2012	ON
17.01.2012	OFF
19.01.2012	ON
28.02.2012	OFF
02.03.2012	ON

Desired result

EVENT_TYPE	DURATION
ON	56
OFF	5

Task 2

Each record in VISITS table represents one visit to client. Company policy requires to visit each client not later than in 30 days after previous visit.

Find all visits that break this rule. Show CLIENT_ID, VISIT_DATE and number of days passed since previous visit.

Initial data set

CLIENT_ID	VISIT_DATE
1	01.01.2012
1	27.01.2012
1	28.02.2012
1	10.03.2012
2	10.02.2012
2	02.03.2012
2	20.05.2012
3	12.04.2012
3	28.04.2012

Desired result

CLIENT_ID	VISIT_DATE	DIFF
1	28.02.2012	32
2	20.05.2012	79

Task 3

Database table called CVs contains info about employees' job history:

- LAST_NAME - employee last name. Unique.
- START_DATE - date of job start.
- END_DATE - date of job end.
- COMPANY - name of company employee worked for.

You should find all periods when employee was working for two companies at the same time.

For each such period show employee last_name, period start and end date and names of both companies

Note: it's known that nobody have worked for three companies at the same time.

Initial data set

LAST_NAME	START_DATE	END_DATE	COMPANY
Baker	01.01.2001	31.07.2004	Sugar candies
Baker	01.03.2002	31.12.2004	McDonalds
Baker	01.01.2005	29.09.2012	Sportlife
Jeeves	01.01.1990	31.12.1993	Mr. Woodster
Jeeves	05.05.1991	31.12.1992	Some TV show
Jeeves	01.01.2004	29.09.2012	Princeton-Plainsboro
Smith	01.04.2002	12.08.2003	Student's heaven
Smith	13.08.2003	20.08.2003	The big mistake
Smith	01.02.2004	30.09.2008	Next step corp
Smith	01.10.2008	29.09.2012	Smith and co

Desired result

LAST_NAME	START_DATE	END_DATE	CURRENT_COMP	PREV_COMP
Baker	01.03.2002	31.07.2004	McDonalds	Sugar candies
Jeeves	05.05.1991	31.12.1992	Some TV show	Mr. Woodster

Task 4

Income tax rate in some Banana Republic is changed very often. These changes are reflected in TAX_RATE table:

- CHANGE_DATE - date when new tax rate is set.
- RATE - tax rate in percent.

Using TAX_RATE table, report rate value for each day between min(change_date) and max(change_date).

Initial data set

CHANGE_DATE	RATE
01.01.2012	10
04.01.2012	13
05.01.2012	15
10.01.2012	13

Desired result

THE DAY	RATE
01.01.2012	10
02.01.2012	10
03.01.2012	10
04.01.2012	13
05.01.2012	15
06.01.2012	15
07.01.2012	15
08.01.2012	15
09.01.2012	15
10.01.2012	13

Task 5

Information about blank passport forms are stored in database table FORMS, one row per form. Due to paper saving initiative, we should show available forms in continuous ranges. Write a query to produce desired report.

Initial data set

SERIA	NUMB
AA	100000
AA	100001
AA	100005
AA	100006
BB	100004
BB	100010
BB	100011
BB	100012

Desired result

SERIA	RANGE_START	RANGE_END
AA	100000	100001
AA	100005	100006
BB	100004	100004
BB	100010	100012

Task 6

During the call mobile phone initiates a connection to cell and keeps it active until call end. If calling person is moving during the call, connection can be transferred from one cell to another to ensure high transmission quality.

Table CONN contains information about phone-to-cell connections:

- PHONE_NO - phone number;
- CELL_ID - cell identifier;
- START_TIME - date and time of connection start;
- DURATION - connection duration in seconds.

Two connections are considered to be parts of one call if difference between start_date1 + duration1 and start_date2 is less than one second. Using CONN table, report PHONE_NUMBER and START_TIME of each call.

Initial data set

PHONE_NO	CELL_ID	START_TIME	DURATION
111-11-11	1	01.10.2012 06:01:00	18
111-11-11	38	01.10.2012 09:38:40	124
222-22-22	1	01.10.2012 06:01:19	71
222-22-22	2	01.10.2012 06:02:30	193
222-22-22	2	01.10.2012 18:40:43	17
222-22-22	1	01.10.2012 18:41:00	89
333-33-33	7	01.10.2012 14:15:01	10
333-33-33	17	01.10.2012 14:15:11	48
333-33-33	9	01.10.2012 14:15:59	156

Desired result

PHONE_NO	START_TIME
111-11-11	01.10.2012 06:01:00
111-11-11	01.10.2012 09:38:40
222-22-22	01.10.2012 06:01:19
222-22-22	01.10.2012 18:40:43
333-33-33	01.10.2012 14:15:01

Part 7. GROUP BY clause extensions

ROLLUP

ROLLUP operator is convenient for various “subtotals and totals” report.

GROUP BY ROLLUP (expr1,...,exprN-1,exprN) is equivalent to:

- GROUP BY expr1,...,exprN-1,exprN union all
- GROUP BY expr1,...,exprN-1 union all
- ...
- GROUP BY expr1 union all
- query without GROUP BY

For example, GROUP BY ROLLUP (country, year) is equivalent to:

- GROUP BY country, year union all
- GROUP BY country union all
- query without GROUP BY

CUBE

Similar to ROLLUP, but produces all possible combination of GROUP BY expressions (2^N combinations for N expressions).

For the same example, result of GROUP BY CUBE(country, year) will be equivalent to:

- GROUP BY country, year union all
- GROUP BY country union all
- GROUP BY year union all
- query without GROUP BY

GROUPING SETS

Grouping sets is the most flexible construction. It allows to define several GROUP BY criteria explicitly.

For example, GROUP BY GROUPING SETS(country, year) is equivalent to:

- GROUP BY country union all
- GROUP BY year.

Another example. GROUP BY GROUPING SETS ((country,year), country, year, ()) is equivalent to:

- GROUP BY country, year union all
- GROUP BY country union all
- GROUP BY year union all
- query without GROUP BY.

(In fact, CUBE is reproduced in this example)

GROUPING function

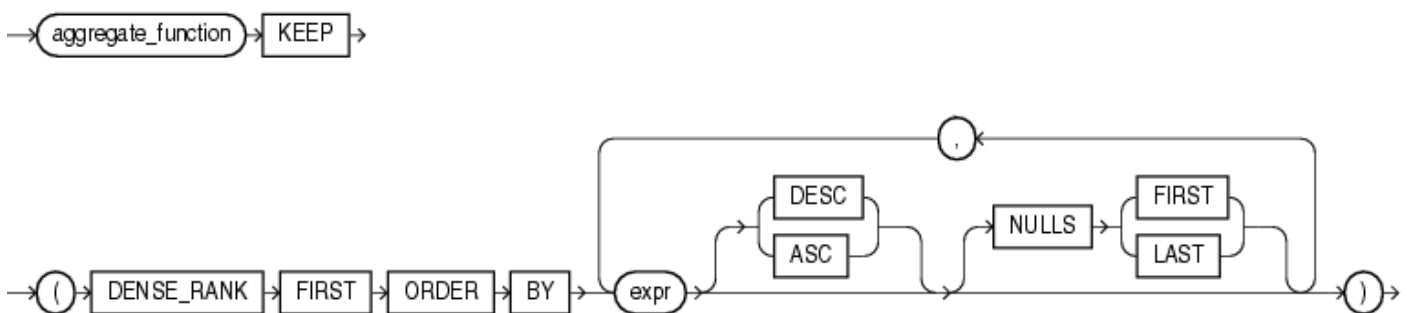
Common problem for listed GROUP BY extensions is to distinguish between “natural” NULLs and “artificial” NULLs (those produced by grouping). This problem can be solved with GROUPING(<expr>) function, which:

- Returns 1, if NULL in <expr> is produced by grouping;
- Returns 0 otherwise.

FIRST/LAST functions

FIRST and LAST are two similar functions (“function extensions” may be better definition) that allows other aggregate functions to operate over “only first” or “only last” rows in data set.

Syntax (for FIRST, version for LAST is similar):



where:

- aggregate_function – one of MIN, MAX, SUM, AVG, COUNT, VARIANCE, or STDDEV functions.
- KEEP – just a keyword saying that <aggregate_function> will operate on first of last rows, not on the whole data set.
- DENSE_RANK FIRST / DENSE_RANK LAST – defines FIRST or LAST rows will be processed.
- ORDER BY ... - defines ordering criteria necessary to identify first or last rows.

Useful links

- [GROUP BY clause](#)
- [GROUPING function](#)
- [FIRST/LAST functions](#)

Practice

1. For EMPLOYEES table, produce a report that will show:
 - a. Number of employees in each department;
 - b. Total number of employees;
2. For EMPLOYEES table, produce a report that will show:
 - a. Total salary for each department;
 - b. Total salary for each job_id;
 - c. Grand total salary;

Note: show departments first, than job_ids, than grand total.

3. Using SALES_VIEW (SH schema), produce the following report for Argentina:
 - a. Total sales per year (total sales = sum(sales));
 - b. For each year – most and least profitable products. Identify profitability by SALES field.

Part 8. PIVOT/UNPIVOT operation

PIVOT idea

Take some column(s) and convert its values into separate columns. Do some aggregation also.

COUNTRY	PRODUCT	YEAR	SALES
Australia	1.44MB External 3.5" Diskette	1998	2535.75
Australia	1.44MB External 3.5" Diskette	1999	2345.04
Australia	1.44MB External 3.5" Diskette	2001	732.17
Australia	1.44MB External 3.5" Diskette	2000	3021.24
Brazil	1.44MB External 3.5" Diskette	2000	9.35
Canada	1.44MB External 3.5" Diskette	2000	1852.16
Canada	1.44MB External 3.5" Diskette	1998	1452.72



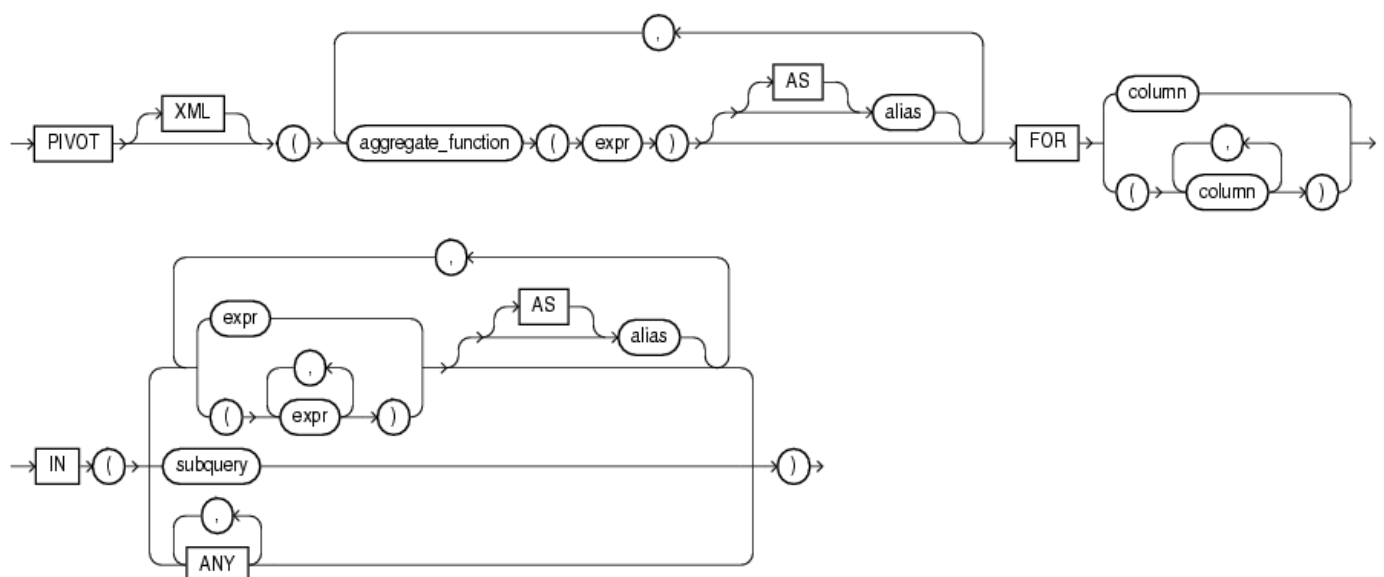
COUNTRY	1998	1999	2000	2001
Argentina	5235.49	5344.48	3363.98	702.69
Australia	1044201.52	856929.35	896291.71	1164870.57
Brazil	12390.99	5083.96	13379.04	5197.89
Canada	578619.96	597684.83	650987.09	859218.21
China	1267.91	1821.45	109.94	628.89
Denmark	497039.93	445738.33	436663.52	598323.01
France	1005833.57	859035.11	886244.61	1025156.84
Germany	2203519.32	2126328.26	2275388.56	2604893.08

PIVOT syntax

Simple example:

```
with t as (select country, year, sales from sales_view)
select *
from t pivot (sum(sales) for year in (1998,1999,2000,2001));
```

Complete syntax:



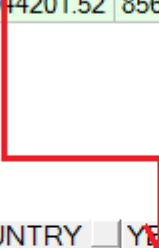
Useful PIVOT facts

- Pivot is executed before any joins or where clauses.
- All columns not mentioned in aggregation clause or in for clause are used as GROUP BY criteria. So use should carefully choose columns you will pass to PIVOT.
- Column names are composed as <value alias>_<aggregate alias>. So, only one among several aggregates may have no value.
- You can't pivot for expressions. So, calculate all necessary expressions before PIVOT.
- You can use "XML" keyword in pair with "ANY" keyword or subquery to dynamically define values for pivoting. However, all pivoting results will be returned in one XML column.

UNPIVOT

Almost "reverse PIVOT" operation. It converts several columns into values of one column. However, it can't undone aggregation, of course.

COUNTRY	1998	1999	2000	2001
Argentina	5235.49	5344.48	3363.98	702.69
Australia	1044201.52	856929.35	896291.71	1164870.57



COUNTRY	YEAR	SALES
Argentina	1998	5235.49
Argentina	1999	5344.48
Argentina	2000	3363.98
Argentina	2001	702.69
Australia	1998	1044201.52
Australia	1999	856929.35
Australia	2000	896291.71
Australia	2001	1164870.57

Syntax example (for tables above):

```
select * from unpivot_me
UNPIVOT(sales for year in ("1998","1999","2000","2001"))
```

Useful links

- [PIVOT clause](#)
- [UNPIVOT clause](#)
- [DWH Guide. Pivoting examples.](#)

Practice

1. SALES_VIEW view. For each product show world-wide sales (in items!) for year 1998,1999,2000. Like this:

PRODUCT	1998	1999	2000
1.44MB External 3.5" Diskette	6602	6586	7568
128MB Memory Card		1851	3983
17" LCD w/built-in HDTV Tuner	2278	1387	786
18" Flat Panel Graphics Monitor	1479	804	1669

2. SALES_VIEW view. For each country show number of DIFFERENT products bought there in 2000 and 2001 years. Like this:

COUNTRY	2000	2001
Argentina	15	12
Australia	71	70
Brazil	20	35
Canada	71	69

3. In HR schema for each department having employees show number of people working as 'SA_REP','ST_CLERK','IT_PROG' as well as total salary of such people. Like this:

DEPARTMENT_NAME	'SA_REP'_SAL	'SA_REP'_EMP	'ST_CLERK'_SAL	'ST_CLERK'_EMP	'IT_PROG'_SAL	'IT_PROG'_EMP
Administration		0		0		0
Accounting		0		0		0
Purchasing		0		0		0
Human Resources		0		0		0
IT		0		0	28800	5
Public Relations		0		0		0
Executive		0		0		0
Shipping		0	55700	20		0
Sales	243500	29		0		0
Finance		0		0		0
Marketing		0		0		0

4. In HR schema show number of hired people for each year and each month. Take into account currently working people only. Like this:

YEAR	1	2	3	4	5	6	7	8	9	10	11	12
2001	1	0	0	0	0	0	0	0	0	0	0	0
2002	0	0	0	0	0	4	0	2	0	0	0	1
2003	0	0	0	0	2	1	1	0	1	1	0	0
2004	2	2	1	0	1	1	1	1	0	1	0	0
2005	3	2	6	1	0	2	2	4	3	3	1	2
2006	3	3	5	3	1	1	3	1	1	0	2	1
2007	1	3	3	1	2	2	0	1	0	1	2	3
2008	4	3	2	2	0	0	0	0	0	0	0	0

Part 9. MODEL clause, part 1.

Model interprets data set as multi-dimensional array optionally divided into partitions.

Model clause column types

- Partitions. Isolated groups of data, like in analytic functions.
- Dimensions. Like array indexes. Combination of dimensions forms address of cell inside partition.
- Measures. Array cells with data. They are addressed with dimension values.

All column types listed are needed to write rules – assignment formulas that modify or create cell values.

Model clause simplified syntax

```
SELECT...
MODEL
  [PARTITION BY (<part expr 1>,...) ]
  DIMENSION BY (<dimension expr 1>[,...])
  MEASURES (<measure 1>[,...])
  RULES [UPDATE | UPSERT | UPSERT ALL] (
    <cell reference 1> = <expression 1>
    [,...]
  );
```

Cell reference

- Positional. Cell is specified with constant or constant expression for each dimension.
- Symbolic. Cell(s) is specified by some Boolean expression for each dimension.

Symbolic references may be multi-cell references.

Multi-cell references

- On the right side of the rule – with aggregate function only.
- On the left side of the rule – enforce rule execution for each cell in multi-cell reference.
- CV([dimension]) functions is useful in the right side of the rule if multi-cell reference is used in the left side.
- ORDER BY expression is useful in the left side to enforce strict cells processing order.

Rules modes

Rules can operate in one of three modes:

- UPDATE. Rule in update mode can only change existing cell and cannot create new cell. Update rules with non-existent left side are ignored.
- UPSERT. The default. Rule in upsert mode works as UPDATE for symbolic references. For positional references it changes or creates cell.
- UPSERT ALL. Changes or creates cells for both positional and symbolic references.

Useful links

- [SQL for modeling](#)
- [MODEL clause](#)

Practice

1. For each country in SALES_VIEW, calculate:
 - a. total sales;
 - b. total sales in 20th century;
 - c. total sales of products starting with 'C' letterReturn calculated rows only.
2. Create a report that will show:
 - a. country;
 - b. product;
 - c. year;
 - d. sales;
 - e. product total: total sales of product for all year in this country;
 - f. year total: total sales of all product during this year in this country.
3. You are trying to calculate how much would you earn if you had pushed additional product called 'Magic box' to Argentina and Australia markets.

Analytics state that in Argentina 'Magic Box' sales would be 30% more than '3 1/2" Bulk diskettes, Box of 50' sales and in Australia - 10% less than total Australia sales.

Basing on these predictions, show sales of 'Magic box' in Argentina and Australia for 1998-2001 period.
4. For each product in each country build sales forecast for 2002 and 2003 years.

Assume that sales in 2002 will be sum of 2000 and 2001 sales, sales in 2002 - twice more than in 2002.

Part 10. MODEL clause, part 2.

NULLs and missing cells

By default, missing cells (references to cells with non-existent combination of dimensions) are treated as NULLs. Necessity to use NVL in calculations with NULLs can be inconvenient for large models with lots of rules.

You can use KEEP NAV or IGNORE NAV to define defaults for NULLs and missing cells:

- **KEEP NAV** – default behavior, NULLs and missing cells are NULLs;
- **IGNORE NAV** – NULLs and missing cells are replaced with:
 - 0 for numeric data;
 - 01-Jan-2001 for dates.

Syntax: MODEL ... MEASURES(...) [**KEEP NAV | IGNORE NAV**]

You can also use:

- **IS PRESENT** predicate – to define if cell was present in the **initial** (pre-model) data set or not.
- **PresentV**(CELL, EXPR1, EXPR2) function. It returns EXPR1 if CELL is present in initial data set. EXPR2 otherwise.
- **PresentNNV**(CELL, EXPR1, EXPR2) function. It returns EXPR1 if CELL is present in initial data set and is not null. EXPR2 otherwise.

FOR loops

For loop can be used in cell reference to make **multi-cell positional reference**. It provides more capabilities to add new cells than UPSERT ALL rule mode.

If “sales” is a measure and “year” is a dimension, the following loop constructions can be used:

- Sales[FOR year in (2000,2001,2002)] – positional reference to three cells ;
- Sales[FOR year FROM 2000 TO 2010 INCREMENT 2] – positional reference to years 2000, 2002,2004, 2006, 2008 and 2010.
- Sales[FOR year in (select distinct year from sales_view)] – positional reference to all year values returned by subquery. Note: subquery can't be correlated in FOR loops.

Iterative models

Using iterative models, you can execute the whole set of rules several times.

Syntax: MODEL...RULES [UPDATE | UPSERT | UPSERT ALL] [**ITERATE** (<number>) [**UNTIL** <condition>]]

If only number of iterations is set, model is executed given number of times. If both number of iterations and UNTIL condition are set, model stops execution after given number of iterations or when UNTIL condition become true, whichever is faster.

Additional features:

- **ITERATION_NUMBER** variable returns number of current iteration. Starts with 0.
- **PREVIOUS**(cell) function lets you to get a value cell had at previous iteration. Useful if you change the same cell at each iteration.

Useful links

- [NULLs and missing cells](#)
- [FOR loops](#)
- [Iterative models](#)

Practice

1. SH schema. For each existing combination of country and product create sales forecast for years 2002-2005. Sales forecast for some year should be equal to sum of sales (real or forecasted) during all previous years.
2. HR schema. For each department having employees show:
 - a. list of employees with salary, commission_pct and total income (salary + salary*commission_pct) for each.
 - b. Sum of salary and sum of income for department;
 - c. Sum of summary salary and sum income for the whole company.

DEPARTMENT_NAME	LAST_NAME	SALARY	COMMISSION	INCOME
Accounting	Gietz	9000		9000
Accounting	Higgins	16000		16000
Accounting	DEPARTMENT TOTAL	25000		25000
Administration	Whalen	6000		6000
Administration	DEPARTMENT TOTAL	6000		6000
...
Sales	Tucker	12008	0.3	15610.4
Sales	Bernstein	12008	0.25	15010
Sales	Hall	12008	0.25	15010
Sales	Olsen	12008	0.2	14409.6
Sales	Cambrault	12008	0.2	14409.6
Sales	DEPARTMENT TOTAL	448632		552601.2
...
Shipping	Vollman	8500		8500
Shipping	Kaufling	8500		8500
Shipping	Fripp	8500		8500
Shipping	Weiss	8500		8500
Shipping	Taylor	5500		5500
Shipping	DEPARTMENT TOTAL	252500		252500
TOTAL		1039132		1143101.2

3. SH schema. Using SALES_VIEW view, create "dense" sales report that will show sales for each POSSIBLE (even not existent) combination of country, product and year. If combination does not exist, show 0 sales.

Part 11. Common practice.

Task 1. Fibonacci series.

Write SQL query that will generate first 50 numbers in Fibonacci series.

Fibonacci series is defined with the following rules:

- $F(0) = F(1) = 1$;
- $F(n) = F(n-1) + F(n-2)$.

Desired result:

N	VAL
0	1
1	1
2	2
3	3
4	5
5	8
6	13
7	21
8	34
9	55
10	89
...	...

Task 2. String parsing

Table **TheString** contains one row and one field called **str**. This field stores some arithmetical expression with numbers, arithmetic signs and parenthesis's. For example, " $2*((5+7)+2*(2+3))*8)+9$ ".

2a. Write a query that will return each symbol from the string as a separate row.

Desired result:

C
2
*
(
(
5
...

2b. Return position of first parenthesis that breaks correct parenthesis's sequence (for the given example – character #20, 3rd from the end).

Task 3. Deposits

Some person has a deposit account. From time to time he/she adds money to that account. Those payments are recorded in **DEPOSITS** table:

- TheDate – date of payment. Just date, without time.
- Payment – sum of payment.

Daily interest rate is 1% (yes, 365% per year!). Interest is paid each day. Moreover, bank uses compound interest - i.e. sum of interest is added to account and used for calculations next day. For example, if initial sum is 1000, account dynamic would be:

- Day 1: 1000.00
- Day 2: 1010.00 = $1000 + 0.01 * 1000$
- Day 3: 1020.10 = $1010 + 0.01 * 1010$
- Etc.

Initial data set:

THEDATE	PAYMENT
01.01.2012	1000
01.02.2012	1000
01.03.2012	500
01.04.2012	1000

Using payments information from DEPOSITS table, calculate:

3a. Account balance on the June 01, 2012. Desired result: **10955.02** (rounded).

3b. Account balance for each day between account opening and June 01, 2012. Desired result:

THEDATE	BALANCE
01.01.2012	1000
02.01.2012	1010
03.01.2012	1020.1
04.01.2012	1030.301
05.01.2012	1040.60401
...	...
29.05.2012	10632.8303292085
30.05.2012	10739.1586325006
31.05.2012	10846.5502188256
01.06.2012	10955.0157210138

Task 4. Store dynamics

Table **STOCK** contains some store IN (receiving goods) and OUT (shipping goods) operations:

- OP_ID – unique operation identifier. Can be used for operations ordering – i.e. operation with greater OP_ID is later operation.
- OP_TYPE – 'IN' or 'OUT';
- OP_COUNT – number if items received or shipped. Positive of both IN and OUT operations.

Initial data set:

OP_ID	OP_TYPE	OP_COUNT
1	IN	10
2	IN	15
3	OUT	5
4	OUT	10
5	IN	5
6	OUT	12

4a. For each operation, calculate store balance before operation and after it. Desired result:

OP_ID	OP_TYPE	BALANCE_BEFORE	OP_COUNT	BALANCE_AFTER
1	IN	0	10	10
2	IN	10	15	25
3	OUT	25	-5	20
4	OUT	20	-10	10
5	IN	10	5	15
6	OUT	15	-12	3

4b. Store implements FIFO (first in – first out) write-off method. For each shipping (OUT) operation define how many items from which IN operation has to be written-off. Desired result:

OP_ID	OP_COUNT	FIFO
3	5	5 items from op.# 1 ...
4	10	5 items from op.# 1 , 5 items from op.# 2 ...
6	12	10 items from op.# 2 , 2 items from op.# 5 ...