Information Integration

Chapter 2. Federated Databases (Integration and Analytical Model)

SIA & SDBIS

Integration Model with OLAP Views

SQL ROLAP from Data Source Views

Case Study Architecture (Review)

- Data Sources
  - SQL: Oracle DB Database 12c/18c/19c/21c,
  - SQL: PostgreSQL 9/10/12,
  - CSV/XLSx: Local FileSystem, Virtual File System (FTP)
  - XML: Local FileSystem or (Web)REST Data Services
  - JSON: Local FileSystem or (Web)REST Data Services from NoSQL databases

- Data source Access Model
  - External Tables
  - Remote Views
    - SQL Remote Views
    - XML Remote Views
    - REST Remote Views
  - Local Tables (ETL)

- Integration Model
  - Consolidation Views
  - Analytical Views

- Web Model:
  - ORDS REST Views,
  - APEX Reports and Charts

2.2 Architecture and components FDB

- 2.2.1 Federated Database System Concept
- 2.2.2 Data Source Model and Access Components
- **2.2.3 Integration and Analytical Model**
- 2.2.4 Integration Web Model

2.2.3 Integration and Analytical Model
Oracle Federated System: Integration Views

- SQL Analytics:
    - Hierarchical queries, PIVOT, UNPIVOT
    - SQL ROLAP Advanced Extensions:
        - ROLLUP, CUBE, GROUPING SETS
        - Analytical Window Functions
- Case Study: Multidimensional SQL schema. Integration Views:
    - ROLAP Fact Views
    - ROLAP Dimensional Views
    - ROLAP Analytical Views

1. SQL Analytics

- Hierarchies and hierarchical levels
  - START WITH...CONNECT BY
- Pivoting Clauses
  - PIVOT
  - UNPIVOT
- OLAP Aggregations and ROLLUP
- Examples
  - 30_Analytical_Integration_Examples.sql

SQL Hierarchical Processing

- Oracle Database offers advanced support for hierarchical (tree-based) queries:
  - Hierarchical SQL clauses
    - START WITH …
    - CONNECT BY PRIOR col_id = col_parent_id
  - Pseudocolumns within hierarchical processing:
    - LEVEL
    - CONNECT_BY_ROOT
    - CONNECT_BY_ISLEAF
  - Iterative processing function to traverse hierarchical levels:
    - SYS_CONNECT_BY_PATH(expr_col)

SQL Hierarchical

```
SELECT category_name, category_code, LEVEL AS nivel,
  LPAD(' ', 3*(LEVEL-1), '-') || category_name AS hierarchy_name,
  SYS_CONNECT_BY_PATH(category_code, '\') AS hierarchical_path,
  CONNECT_BY_ROOT category_name AS root_category,
  CONNECT_BY_ISLEAF last_level
FROM CTG_HIERARCHY p
     START WITH parent_category IS NULL
     CONNECT BY PRIOR category_code = parent_category
;
```

SQL PIVOT Clause

- Use PIVOT clause to convert rows into columns.
- Use UNPIVOT clause to convert columns into rows.

PIVOT Operator Syntax

- PIVOT(
    - Aggregation_Function(value_col)
    - FOR pivot_column
        - IN (

          pivot_col_key_1_value AS “pivot_col_key1_name”,

          pivot_col_key_2_value AS “pivot_col_key2_name”,

          …)
    -)

SQL PIVOT SELECT Query

```
SELECT * FROM (
  SELECT p.prod_name,
    f.sales_amount AS Total_Sales
  FROM OLAP_FACTS_SALES_AMOUNT f
    INNER JOIN PRODUCTS_VIEW p ON f.product_code = p.product_code
  WHERE EXTRACT (YEAR FROM invoice_date) = 2016 AND
    EXTRACT (MONTH FROM invoice_date) IN (1, 7)
  ORDER BY 1
 ) V
PIVOT (
  SUM(Total_Sales)
  FOR prod_name IN (
    'Prod A' AS "Produs A",
    'Prod B' AS "Produs B",
    'Prod C' AS "Produs C",
    'Prod D' AS "Produs D"))
ORDER BY 1;
```

UNPIVOT Operator Syntax

- UNPIVOT(
    - generated_value_column_name
    - FOR generated_label_column_name_from_header
        - IN (
            - initial_column1_name AS ‘label1’,
            - initial_column2_name AS ‘label2’,
            - …)
    -)

SQL UNPIVOT SELECT Query

```
SELECT cust_id, detail_label, detail_value
    FROM customers_details_view
UNPIVOT INCLUDE NULLS
    (detail_value
    FOR detail_label IN
        (
            CREDIT_RATING as 'CREDIT_RATING',
            INDUSTRY as 'INDUSTRY',
            COMP_TYPE as 'COMP_TYPE')
    );
```

SQL ROLAP: Subtotals

- Sub-totals' problem - solution strategies:
  - (s1) each sub-total level produced by specific query and UNION all datasets;
  - (s2) use OLAP specific clauses as ROLAP.

ROLLUP

- Syntax
  - SELECT ... FROM ...
  - GROUP BY **ROLLUP**(gr_col1, gr_col2)
- **ROLLUP** Aggregation Processing Rules
  - Creates subtotals/aggregations “rolling up” from the most detailed level to the most synthetic one: produces *n+1* aggregations levels (subtotals included), where *n* is the column number from ROLLUP clause.

Aggregation Levels: ROLLUP(a,b,c,d)					
Detail Level Base aggregation level a,b,c,d	a	b	c	d	∑
(Sub) Total Agregation level a,b,c [detail{abcd} –> {d}] Aggregating oOne step from left to the right	a	b	c		∑d
(Sub) Total Agregation level a,b [previous_level{abc} –> {c}]	a	b			∑c
(Sub) Total Agregation level [previous_level{ab} –> {b}]	a				∑b
Overall Total External Aggregation (+1)					∑a

SQL ROLLUP example

```
SELECT
CASE
    WHEN GROUPING(D.Departament_Name) = 1 THEN '{Total General}'
    ELSE D.Departament_Name END AS Departament_Name,
  CASE
    WHEN GROUPING(D.Departament_Name) = 1 THEN ' '
    WHEN GROUPING(D.City_Name) = 1 THEN 'subtotal Departament ' || D.Departament_Name
    ELSE D.City_Name END AS City_Name,
  CASE
    WHEN GROUPING(D.Departament_Name) = 1 THEN ' '
    WHEN GROUPING(D.City_Name) = 1 THEN ' '
    WHEN GROUPING(D.Customer_Name) = 1 THEN 'subtotal city ' || d.City_Name
    ELSE to_char(D.Customer_Name) END AS Customer_Name,
  SUM(NVL(f.SALES_AMOUNT, 0)) as SALES_AMOUNT
FROM OLAP_DIM_CUSTS_CITIES_DEPTS D
    INNER JOIN OLAP_FACTS_SALES_AMOUNT F ON D.customer_id = F.Cust_Id
GROUP BY ROLLUP (d.Departament_Name, d.City_Name, d.Customer_Name)
ORDER BY d.Departament_Name, d.City_Name, d.Customer_Name;
```

Advanced **ROLAP**

- ROLAP Aggregation SQL Clauses:
  - ROLLUP
  - CUBE
  - GROUPING SETS.
- Analytical Functions and aggregations:
  - RANKing functions
  - ROW window functions
  - Other functions: FIRST_VALUE, RATIO_TO_REPORT.

CUBE

- As the ROLLUP operator, CUBE extension of GROUP BY clause aims to produce subsequent **sub-totals** (alongside with base aggregation level set by the “classical” GROUP BY clause).
- Unlike ROLLUP operator, that set a single aggregation direction to produce subsequent sub-totals by processing grouping columns only from right to left, CUBE operator multiplies aggregation directions to any possible direction.
- CUBE represents a *generalization form of ROLLUP operator.*

CUBE example

```
SELECT
  d.Departament_Name,
  d.City_Name,
  d.Customer_Name,
  SUM(NVL(f.SALES_AMOUNT, 0)) as SALES_AMOUNT
FROM OLAP_DIM_CUSTS_CITIES_DEPTS D
    INNER JOIN OLAP_FACTS_SALES_AMOUNT F ON D.customer_id = F.Cust_Id
GROUP BY CUBE (
     d.Departament_Name,
     d.City_Name,
     d.Customer_Name
     )
ORDER BY
     d.Departament_Name,
     d.City_Name,
     d.Customer_Name;
```

Partial ROLLUP and partial CUBE

- Partial ROLLUP and partial CUBE represents extensions of the simple classical operators: they will generate, beside the base aggregation level, those aggregation levels that will result from column combinations covered internal (with simple parenthesis) by ROLLUP or CUBE clauses.

Partial CUBE example

```
SELECT
      Departament_Name,
      City_Name,
      Customer_Name,
      SUM(NVL(f.SALES_AMOUNT, 0)) as SALES_AMOUNT
FROM OLAP_DIM_CUSTS_CITIES_DEPTS D
    INNER JOIN OLAP_FACTS_SALES_AMOUNT F ON D.customer_id = F.Cust_Id
GROUP BY
      d.Departament_Name,
      CUBE(
           d.City_Name,
           d.Customer_Name)
ORDER BY
      d.Departament_Name,
      d.City_Name,
      d.Customer_Name;
```

GROUPING SETS

- GROUPING SETS extension of GROUP BY clause will impact to the aggregation granularity.

- While the CUBE operator will multiply the number of aggregation levels, the GROUPING SETS operator could reduce this number.

GROUPING SETS

- The aggregation levels will be set by GROUPING SETS operator using simple columns or column groups.
- GROUPING SETS (a,b,c) will result in only 3 aggregation levels – subtotals for a, b, şi c.
- GROUPING SETS (a,b,(b,c)) will result in only 3 aggregation levels:
  - Subtotal a;
  - Subtotal b;
  - Subtotal of b,c column group;
- GROUPING SETS (a,b,(b,c), ()) will insert also an overall subtotal.

GROUPING SETS Example

```
SELECT
CASE
    WHEN GROUPING(D.Departament_Name) = 1 THEN '{Total General}'
    ELSE D.Departament_Name END AS Departament_Name,
  CASE
    WHEN GROUPING(D.Departament_Name) = 1 THEN ' '
    WHEN GROUPING(D.City_Name) = 1 THEN 'subtotal Departament ' || D.Departament_Name
    ELSE D.City_Name END AS City_Name,
  CASE
    WHEN GROUPING(D.Departament_Name) = 1 THEN ' '
    WHEN GROUPING(D.City_Name) = 1 THEN ' '
    WHEN GROUPING(D.Customer_Name) = 1 THEN 'subtotal city ' || d.City_Name
    ELSE to_char(D.Customer_Name) END AS Customer_Name,
  SUM(NVL(f.SALES_AMOUNT, 0)) as SALES_AMOUNT
FROM OLAP_DIM_CUSTS_CITIES_DEPTS D
    INNER JOIN OLAP_FACTS_SALES_AMOUNT F ON D.customer_id = F.Cust_Id
GROUP BY GROUPING SETS(Departament_Name, (Departament_Name, City_Name, Customer_Name), () )
ORDER BY d.Departament_Name, d.City_Name, d.Customer_Name;
```

Analytical Functions and running-aggregations

- Analytical Functions Syntax with **window** clause:
  - FN_ANALITICĂ(expression)
    - OVER(
      - PARTITION BY   partitioning_column
      - ORDER BY   order_by_criteria_column
      - ROWS   <window_clause>
      -)

Analytical Functions Windows - ROWS clause

- Window Clause: **ROWS** BETWEEN <start point> AND <end point>:
  - UNBOUNDED PRECEDING
  - UNBOUNDED FOLLOWING
  - CURRENT ROW
  - value_expr PRECEDING
  - value_expr FOLLOWING

Analytical Functions Windows - ROWS clause

- ROWS UNBOUNDED PRECEDING – aggregation from the first row of the partition to the current row;
- ROWS n PRECEDING;
- ROWS BETWEEN *CURRENT ROW* AND m FOLLOWING;
- ROWS BETWEEN n PRECEDING AND m FOLLOWING;
- ROWS BETWEEN *CURRENT ROW* AND UNBOUNDED FOLLOWING.

ROW Functions

```
SELECT INVOICE_DATE, CUST_ID, SALES_AMOUNT,
SUM(SALES_AMOUNT) OVER(PARTITION BY INVOICE_DATE ORDER BY CUST_ID
    ROWS UNBOUNDED PRECEDING) AS Aggregated_Amount_UP,
SUM(SALES_AMOUNT) OVER(PARTITION BY INVOICE_DATE ORDER BY CUST_ID
    ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS Aggregated_Amount_CRUF,
SUM(SALES_AMOUNT) OVER(PARTITION BY INVOICE_DATE ORDER BY CUST_ID
    ROWS 1 PRECEDING) AS Aggregated_Amount_1PCR,
SUM(SALES_AMOUNT) OVER(PARTITION BY INVOICE_DATE ORDER BY CUST_ID
    ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS Aggregated_Amount_CR1F,
SUM(SALES_AMOUNT) OVER(PARTITION BY INVOICE_DATE ORDER BY CUST_ID
    ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS Aggregated_Amount_1P1F
FROM OLAP_FACTS_SALES_AMOUNT
ORDER BY 1,2;
```

Analytical Functions

- Statistical functions (and common statistical aggregation functions):
  - SUM, MIN, MAX, AVERAGE, MEAN, MODE, STDEV.
- Ranking functions:
  - RANK () OVER()
  - DENSE_RANK () OVER()
  - PERCENT_RANK () OVER()
- Other analytical functions:
  - FIRST_VALUE () OVER(), LAST_VALUE() OVER()
  - RATIO_TO_REPORT () OVER()
  - LAG() OVER(), LEAD() OVER()
  - LISTAGG () WITHIN GROUP(ORDER BY)

Analytical Functions

```
SELECT PRODUCT_CODE, Product_Sales,
  RANK() OVER (ORDER BY Product_Sales DESC) AS Poz1_RANK,
  DENSE_RANK() OVER (ORDER BY Product_Sales DESC) AS Poz2_DENSE_RANK,
  ROW_NUMBER() OVER (ORDER BY Product_Sales DESC) AS Poz3_ROW_NUMBER,
  TRUNC(PERCENT_RANK() OVER (ORDER BY Product_Sales DESC), 2) AS Poz4_PERCENT_RANK2
FROM (SELECT PRODUCT_CODE,SUM(SALES_AMOUNT) AS Product_Sales
      FROM OLAP_FACTS_SALES_AMOUNT
      GROUP BY PRODUCT_CODE ORDER BY 2 DESC) Top_Product_Sales
ORDER BY 3,1;
```

CASE STUDY: ROLAP Multidimensional DB

2. ROLAP Multidimensional Databases

- Multidimensional Database (MDB) Concepts
  - Relational MDB Strategies
  - MDB Structures.
- Multidimensional Analytics.
- ROLAP MDB Design Process.

Case Study: SQL ROLAP MDB

- SQL Script Example:
  - 31_OLAP_Multidimensional_Analytical

ROLAP Multidimensional Databases Concepts

- Relational MDB Strategies:
  - *Star Schemas*: all levels from an analytical dimension will be represented (merged) into a (single) dedicated table/view.
  - *Snowflake Schemas*: each dimensional level will be represented as a distinct table/view.

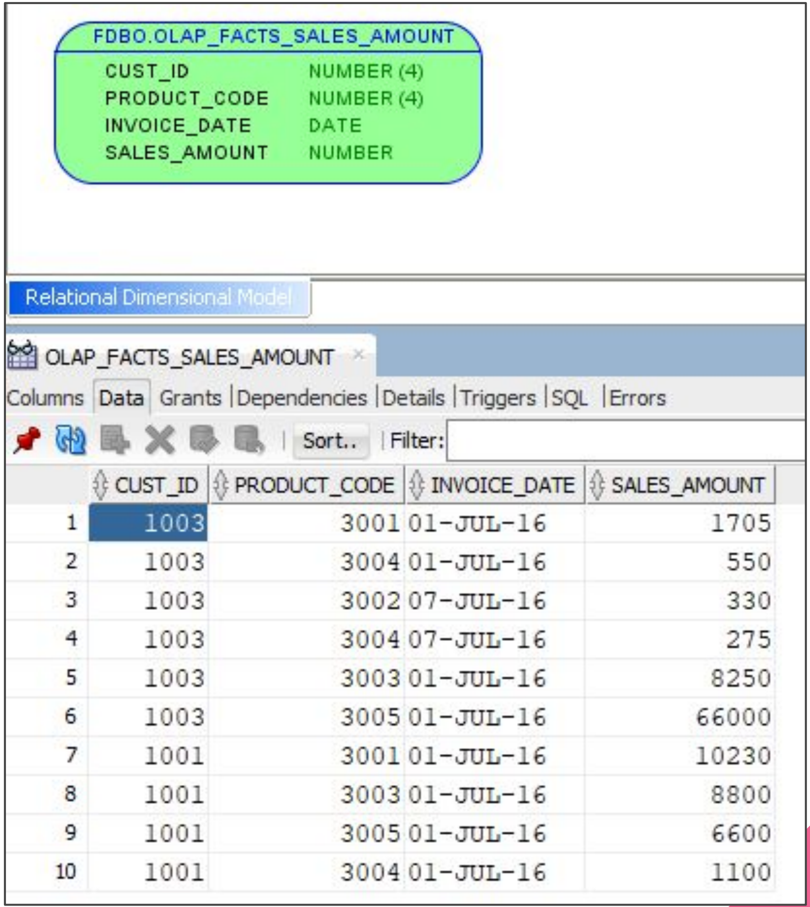
- MDB Specific Structures:
  - Dimension Tables (analytical dimensions)
  - Facts Tables (quantitative data)
  - Analytical Queries (ROLAP queries).

Facts

- "**Facts**" are transactional events coming from transactional or operational (database) systems - OLTP.

- The facts represent numerical values or metrics (quantitative data) that could be hierarchical aggregated (by using analytical functions) along the subsequent levels described by the dimensional structures.

Fact Table



FDBO.OLAP_FACTS_SALES_AMOUNT

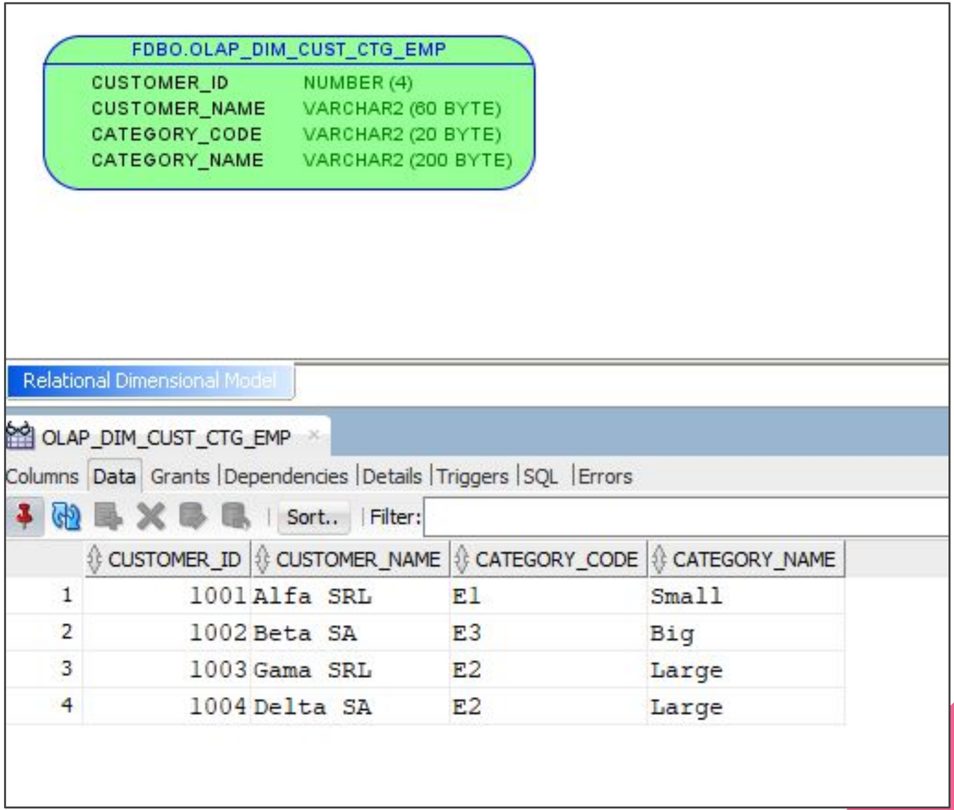
CUST_ID	NUMBER (4)
PRODUCT_CODE	NUMBER (4)
INVOICE_DATE	DATE
SALES_AMOUNT	NUMBER

	CUST_ID	PRODUCT_CODE	INVOICE_DATE	SALES_AMOUNT
1	1003	3001	01-JUL-16	1705
2	1003	3004	01-JUL-16	550
3	1003	3002	07-JUL-16	330
4	1003	3004	07-JUL-16	275
5	1003	3003	01-JUL-16	8250
6	1003	3005	01-JUL-16	66000
7	1001	3001	01-JUL-16	10230
8	1001	3003	01-JUL-16	8800
9	1001	3005	01-JUL-16	6600
10	1001	3004	01-JUL-16	1100

Dimensions

- **The dimensions** represent the analytical axes used for
  - searching and filtering
  - analytical aggregations →
    - of *facts* from *fact table*.
- The Dimensions describe details about the facts that will be engaged within analytical processings.
- The Dimensional hierarchies cover the possible aggregation levels along with the subordinate relationships between these levels.

Dimension Table



FDBO.OLAP_DIM_CUST_CTG_EMP

CUSTOMER_ID	NUMBER (4)
CUSTOMER_NAME	VARCHAR2 (60 BYTE)
CATEGORY_CODE	VARCHAR2 (20 BYTE)
CATEGORY_NAME	VARCHAR2 (200 BYTE)

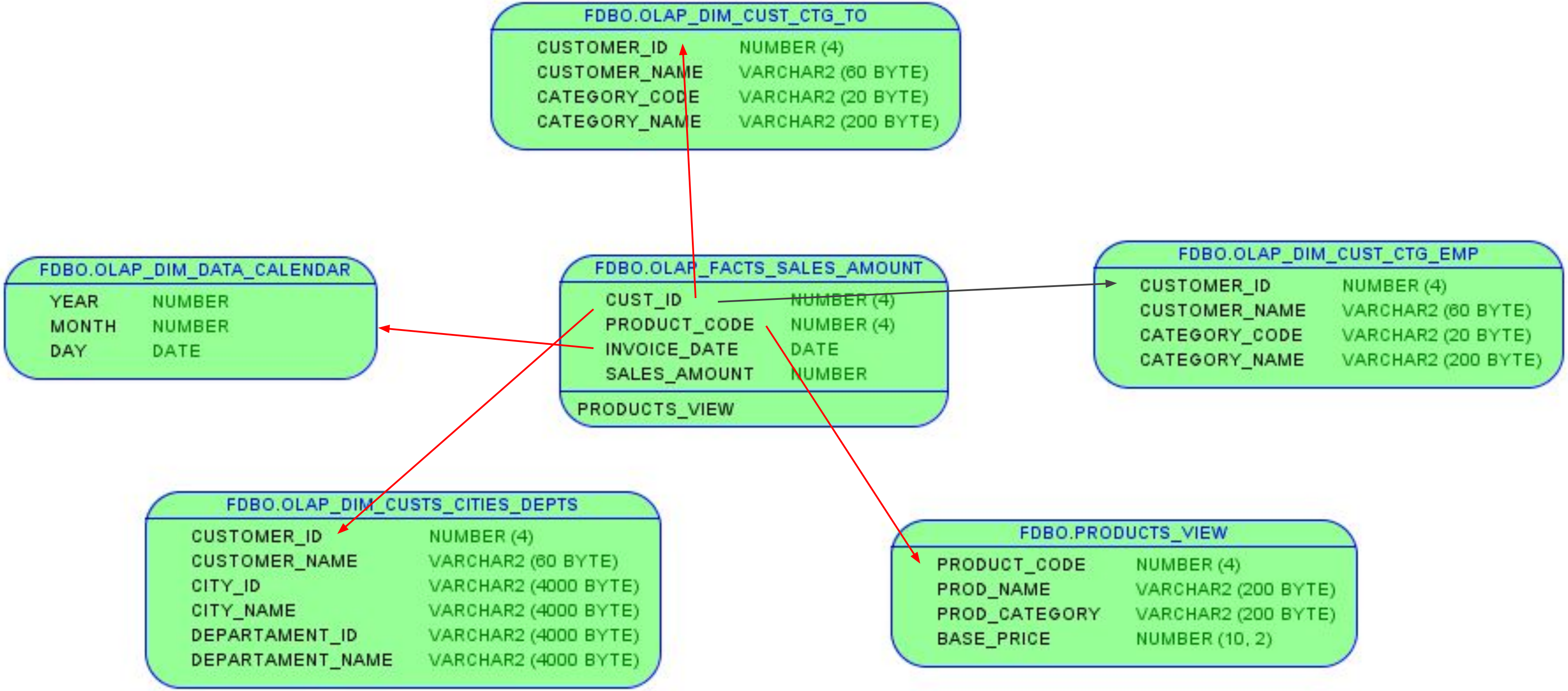
	CUSTOMER_ID	CUSTOMER_NAME	CATEGORY_CODE	CATEGORY_NAME
1	1001	Alfa SRL	E1	Small
2	1002	Beta SA	E3	Big
3	1003	Gama SRL	E2	Large
4	1004	Delta SA	E2	Large

Integration Multidimensional Model
OLAP Business Case

- Fact table
  - OLAP_FACTS_SALES_AMOUNT
- Dimensional tables
  - OLAP_DIM_CUSTS_CITIES_DEPTS
  - OLAP_DIM_DATA_CALENDAR
  - OLAP_DIM_CUST_CTG_TO
  - OLAP_DIM_CUST_CTG_EMP
  - PRODUCTS_VIEW
  - PERIODS_VIEW

- Analytical Views
  - OLAP_VIEW_SALES_DEP_CIT_CUST
  - OLAP_VIEW_SALES_CALENDAR
  - OLAP_VIEW_SALES_CTG_CUST_TO
  - OLAP_VIEW_SALES_CTG_CUST_EMP

Star Dimensional Model



FDBO.OLAP_DIM_CUST_CTG_TO

CUSTOMER_ID	NUMBER (4)
CUSTOMER_NAME	VARCHAR2 (60 BYTE)
CATEGORY_CODE	VARCHAR2 (20 BYTE)
CATEGORY_NAME	VARCHAR2 (200 BYTE)

FDBO.OLAP_DIM_DATA_CALENDAR

YEAR	NUMBER
MONTH	NUMBER
DAY	DATE

FDBO.OLAP_FACTS_SALES_AMOUNT

CUST_ID	NUMBER (4)
PRODUCT_CODE	NUMBER (4)
INVOICE_DATE	DATE
SALES_AMOUNT	NUMBER

PRODUCTS_VIEW

FDBO.OLAP_DIM_CUST_CTG_EMP

CUSTOMER_ID	NUMBER (4)
CUSTOMER_NAME	VARCHAR2 (60 BYTE)
CATEGORY_CODE	VARCHAR2 (20 BYTE)
CATEGORY_NAME	VARCHAR2 (200 BYTE)

FDBO.OLAP_DIM_CUSTS_CITIES_DEPTS

CUSTOMER_ID	NUMBER (4)
CUSTOMER_NAME	VARCHAR2 (60 BYTE)
CITY_ID	VARCHAR2 (4000 BYTE)
CITY_NAME	VARCHAR2 (4000 BYTE)
DEPARTAMENT_ID	VARCHAR2 (4000 BYTE)
DEPARTAMENT_NAME	VARCHAR2 (4000 BYTE)

FDBO.PRODUCTS_VIEW

PRODUCT_CODE	NUMBER (4)
PROD_NAME	VARCHAR2 (200 BYTE)
PROD_CATEGORY	VARCHAR2 (200 BYTE)
BASE_PRICE	NUMBER (10, 2)

Consolidation View: OLAP Facts View

- OLAP_FACTS_SALES_AMOUNT
  - from data source views:
    - INVOICES_VIEW [SQL.ORCL]
    - INVOICE_LINE_ITEMS [SQL.ORCL]
    - PRODUCTS_VIEW [SQL.ORCL]
  - metric column: sales_amount

OLAP Facts View

```
CREATE OR REPLACE VIEW OLAP_FACTS_SALES_AMOUNT AS
SELECT I.Cust_Id, P.Product_Code, I.Invoice_Date
    , SUM(L.Quantity * L.Unit_Price + L.Line_Vat) as SALES_AMOUNT
FROM INVOICES_VIEW i
    INNER JOIN INVOICE_LINE_ITEMS L --INVOICE_LINE_ITEMS_VIEW
        ON i.invoice_id =  l.invoice_id
    INNER JOIN PRODUCTS_VIEW p ON l.PRODUCT_CODE = p.PRODUCT_CODE
GROUP BY I.Cust_Id, P.Product_Code, I.Invoice_Date
```

OLAP Dimensional Views

- OLAP_DIM_CUSTS_CITIES_DEPTS
  - from data source views:
    - Customers_view [SQL.PG]
    - Customers_details_view [SQL.PG]
    - Customers_addresses_view [SQL.PG]
    - Cities_view [XML.File]
    - Departaments_cities_view [XML.File]
    - Departaments_view [XML.File]
  - *Aggregation levels: customer_id/name → city_id/name → departament_id/name*
- OLAP_DIM_DATA_CALENDAR from data source views and local tables:
  - Periods local table from
    - Periods_View [XLSX.File]
  - *Aggregation levels: day → month → year*

OLAP Dimensional Views

- OLAP_DIM_CUST_CTG_TO
  - from data source views:
    - Customers_view [SQL.PG]
    - Customers_details_view [SQL.PG]
    - CTG_CUST_TO table from
      - CTG_CUST_TO_VIEW [XLSX.File]
  - Aggregation levels: customer_id/name → (turnover)category_code/name
- OLAP_DIM_CUST_CTG_EMP
  - from data source views:
    - Customers_view [SQL.PG]
    - Customers_details_view [SQL.PG]
    - CTG_CUST_EMP table from
      - CTG_CUST_EMP_VIEW [XLSX.File]
  - Aggregation levels: customer_id/name → (nr.of employees)category_code/name
- PRODUCTS_VIEW data source views [SQL.ORCL]
  - Aggregation levels: product → prod_category

Dimension View

```
CREATE OR REPLACE VIEW OLAP_DIM_CUST_CTG_TO AS
SELECT
    C.Cust_Id as customer_id, C.Name as customer_name, -- L1
    T.Category_code, T.Category_name -- L2
FROM CUSTOMERS_VIEW C
    INNER JOIN CUSTOMERS_DETAILS_VIEW D ON C.CUST_ID=D.CUST_ID
    INNER JOIN CTG_CUST_TO T ON D.TURNOVER BETWEEN T.Lower_L and T.Upper_L
;
```

OLAP Analytical Views

- OLAP_VIEW_SALES_DEP_CIT_CUST from OLAP views:
  - OLAP_DIM_CUSTS_CITIES_DEPTS dimensional view;
  - OLAP_FACTS_SALES_AMOUNT facts view
- OLAP_VIEW_SALES_CALENDAR from OLAP views:
  - OLAP_DIM_DATA_CALENDAR dimensional view;
  - OLAP_FACTS_SALES_AMOUNT facts view
- OLAP_VIEW_SALES_CTG_CUST_TO from OLAP views:
  - OLAP_DIM_CUST_CTG_TO dimensional view;
  - OLAP_FACTS_SALES_AMOUNT facts view
- OLAP_VIEW_SALES_CTG_CUST_EMP from OLAP views:
  - OLAP_DIM_CUST_CTG_EMP dimensional view;
  - OLAP_FACTS_SALES_AMOUNT facts view

ROLAP Analytical View

```
CREATE OR REPLACE VIEW OLAP_VIEW_SALES_CTG_CUST_TO AS
SELECT
CASE
    WHEN GROUPING(D.Category_Name) = 1 THEN '{Total General}'
    ELSE D.Category_Name END AS Category_Name,
  CASE
    WHEN GROUPING(D.Category_Name) = 1 THEN ' '
    WHEN GROUPING(D.Customer_Name) = 1 THEN 'subtotal category ' || D.Category_Name
    ELSE D.Customer_Name END AS Customer_Name,
  SUM(NVL(f.SALES_AMOUNT, 0)) as SALES_AMOUNT
FROM OLAP_DIM_CUST_CTG_TO D
    INNER JOIN OLAP_FACTS_SALES_AMOUNT F ON D.customer_id = F.Cust_Id
GROUP BY ROLLUP (d.Category_Name, d.Customer_Name)
ORDER BY d.Category_Name, d.Customer_Name;
```

Refs

- Marin Fotache *SQL, dialecte DB2, PostgreSQL și SQL Server*, Ed. a 2-a, Iași, Polirom 2009
    - Capitolul 11. OLAP în SQL
    - Capitolul 12. SQL, ierarhii și recursivitate
- http://www.oracle-base.com/articles/misc/rollup-cube-grouping-functions-and-grouping-sets.php
- http://www.oracle-base.com/articles/misc/analytic-functions.php
- https://www.oracle.com/database/technologies/bi-datawarehousing.html
- https://www.oracletutorial.com/oracle-aggregate-functions/
- https://www.oracletutorial.com/oracle-aggregate-functions/oracle-listagg/

Refs

- Database Journal - Oracle SQL enhancements
  - http://www.databasejournal.com/features/oracle/article.php/3523296/Strong-SQL-Made-Even-Stronger-Oracle-10g-SQL-Enhancements-Part-1.htm
  - http://www.databasejournal.com/features/oracle/article.php/3530101/Data-Densification-Demystified-Oracle-10g-SQL-Enhancements-Part-2.htm
  - http://www.databasejournal.com/features/oracle/article.php/3550281/Playing-With-MODELs-Oracle-10g-SQL-Enhancements-Part-3.htm