U. PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Object Relational Assignment

Group C

Telmo Alexandre Espírito Santo Baptista `up201806554@edu.fe.up.pt`
Telmo Alexandre Espírito Santo Baptista `up201806554@edu.fe.up.pt`

17th May 2022

TBD Project - 2021/22 - MEIC

Teachers

Gabriel de Sousa Torcato David `gtd@fe.up.pt`

Index

1 Introduction

Within the scope of the Databases Technologies course of Masters in Informatics and Computing Engineering, this project was developed with the objective of implementing an object relational model provided by Oracle, from a base relational model, in order to make use of various functionalities provided by the object relational model, such as, collections, inheritance and function methods. Furthermore, a set of queries is to also be implemented for validation of the model.

2 Object Relational Model

2.1 Schema

The original relational schema is composed only of one-to-many relations. As such, the methods used to represent these associations on the object relational model were either of inclusion of the many side on the one side of the relation, if such table didn't deem necessary to exist on its own. This is the case of the table *Leaderships* that will be represented included in the table *Periods*. The other method is to have symmetrical references, by adding a reference to the one side in the many side, and a collection of references in the one side to the many side. This method was used for all the relations where it was useful to have the tables in separate, this includes all the relations other than the one aforementioned.

Additionally, the schema contains two self-referencing tables, *Headings* and *Municipalities*. To convert these two tables, two approaches were followed:

- *Headings* → as there's no explicit hierarchical structure, only implicit, we opted to create symmetrical references in the table itself;
- *Municipalities* → in this case, there's an explicit hierarchical structure (e.g. Municipality → NUTS III → NUTS II → NUTS I → Country), so it was created a table for each of these levels, and used symmetrical references to represent the relations.

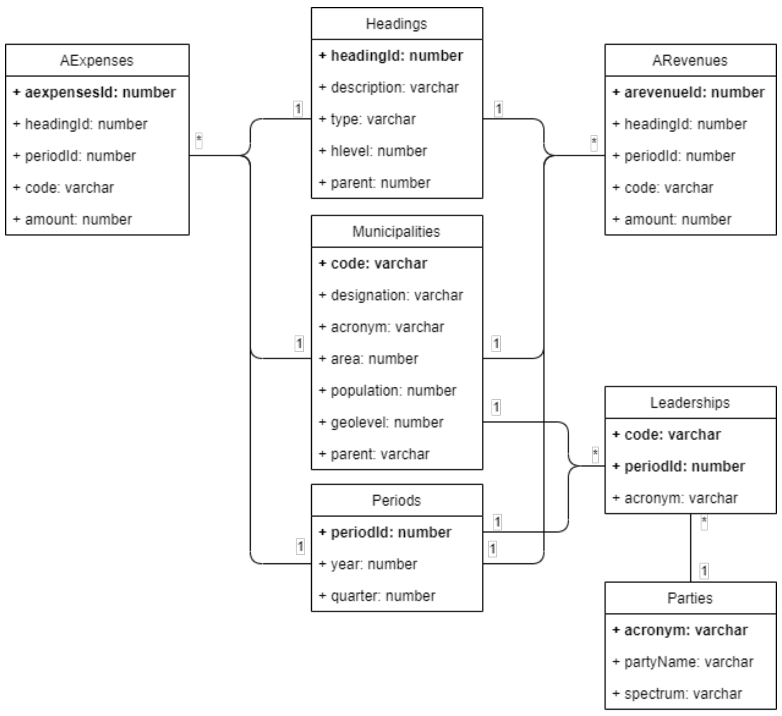


Figure 1: Relational Schema

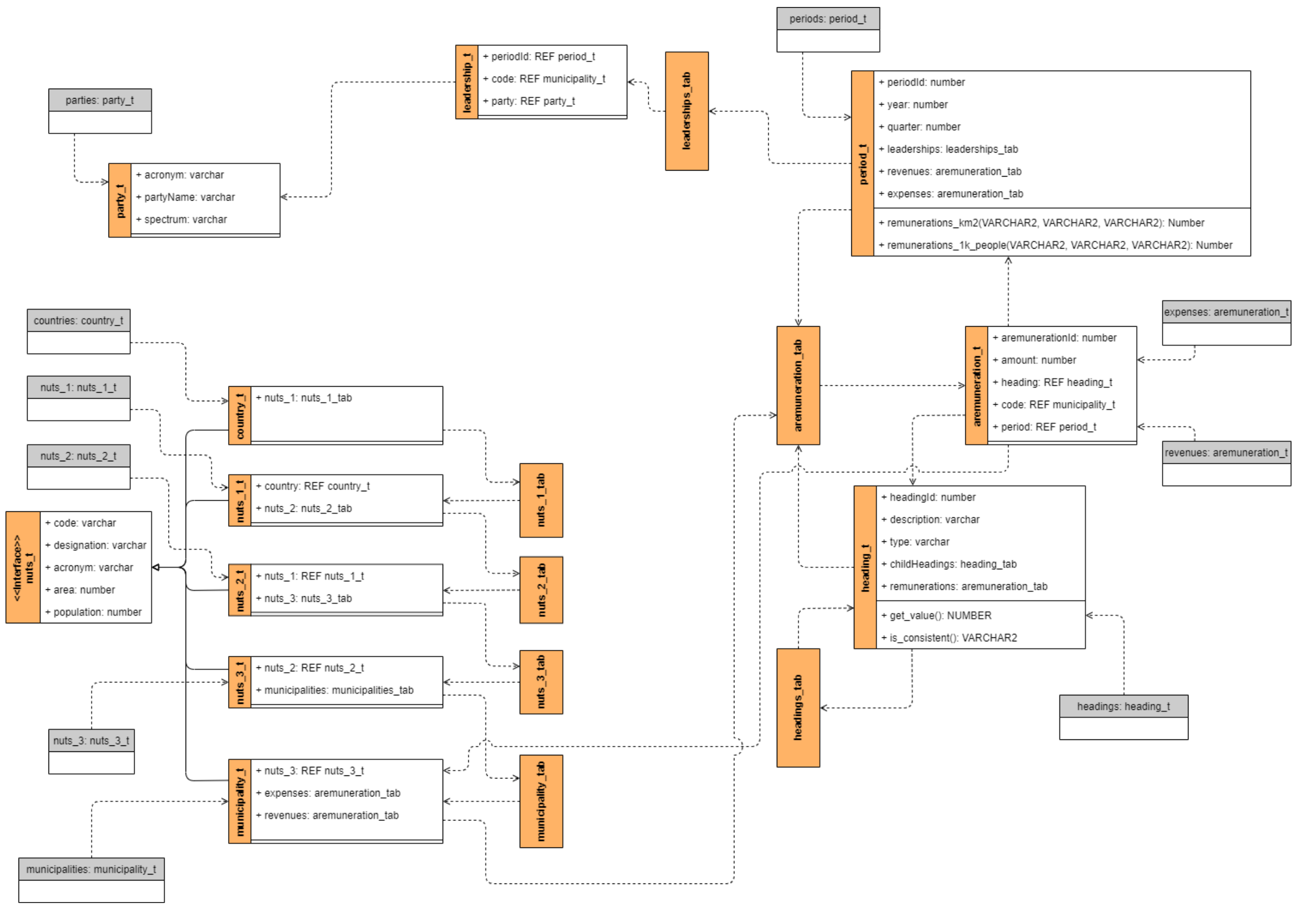


Figure 2: Object-Relational Schema

Script

Only parts of the script are represented here, the full script is available at */scripts/create.sql* in the project folder.

Listing 2.1: Model Creation

```
-- NUTS
---- Base Types
CREATE OR REPLACE TYPE nuts_t AS OBJECT(
    code            VARCHAR2(10),
    designation     VARCHAR2(255),
    acronym         VARCHAR2(10),
    area            NUMBER(38, 0),
    population      NUMBER(38, 0)
) NOT INSTANTIABLE NOT FINAL;
CREATE OR REPLACE TYPE country_t UNDER nuts_t (
);
...
CREATE OR REPLACE TYPE municipality_t UNDER nuts_t (
    nuts_3 REF nuts_3_t
);
---- Table Types
CREATE OR REPLACE TYPE municipality_tab AS TABLE OF REF municipality_t;
...
CREATE OR REPLACE TYPE nuts_1_tab AS TABLE OF REF nuts_1_t;
---- Add nested tables onto base types
ALTER TYPE nuts_3_t ADD ATTRIBUTE (municipalities municipality_tab) CASCADE;
...
ALTER TYPE country_t ADD ATTRIBUTE (nuts_1 nuts_1_tab) CASCADE;
...
-- PERIODS
CREATE OR REPLACE TYPE leadership_t AS OBJECT (
    code REF municipality_t,
    party REF party_t
);
CREATE OR REPlACE TYPE leaderships_tab AS TABLE OF leadership_t;
CREATE OR REPLACE TYPE period_t AS OBJECT (
    periodId        NUMBER(38, 0),
    year            NUMBER(38, 0),
    quarter         NUMBER(38, 0),
    leaderships     leaderships_tab,

    MEMBER FUNCTION remunerations_km2(heading_name VARCHAR2, party_name VARCHAR2,
        municipality_code VARCHAR2) RETURN NUMBER,
    MEMBER FUNCTION remunerations_1k_people(heading_name VARCHAR2, party_name
        VARCHAR2, municipality_code VARCHAR2) RETURN NUMBER
);
-- HEADINGS
CREATE OR REPLACE TYPE heading_t AS OBJECT (
    headingId       NUMBER(38, 0),
    description     VARCHAR2(128),
    remun_type      VARCHAR2(1),
    MEMBER FUNCTION get_value RETURN NUMBER,
    MEMBER FUNCTION is_consistent RETURN VARCHAR2
);
```

```
CREATE OR REPLACE TYPE headings_tab AS TABLE OF REF heading_t;
ALTER TYPE heading_t ADD ATTRIBUTE (parentHeading REF heading_t, childHeadings
    headings_tab) CASCADE;
-- REMUNERATIONS
CREATE OR REPLACE TYPE aremuneration_t AS OBJECT (
    aremunerationId   NUMBER(38, 0),
    amount            NUMBER(38, 2),
    heading           REF heading_t,
    code              REF municipality_t,
    period            REF period_t
);
CREATE OR REPLACE TYPE aremuneration_tab AS TABLE OF REF aremuneration_t;
-- Backwards addition of remunerations tables
ALTER TYPE heading_t ADD ATTRIBUTE (remunerations aremuneration_tab) CASCADE;
...
ALTER TYPE municipality_t ADD ATTRIBUTE (expenses aremuneration_tab, revenues
    aremuneration_tab) CASCADE;
-- TABLE CREATION
CREATE TABLE countries OF country_t (
    code          PRIMARY KEY
)
    NESTED TABLE nuts_1 STORE AS nuts_1_nt;
...
CREATE TABLE periods OF period_t (
    periodId   PRIMARY KEY
)
    NESTED TABLE leaderships STORE AS leadership_nt
    NESTED TABLE expenses STORE AS period_expenses_nt
    NESTED TABLE revenues STORE AS period_revenues_nt;
```

2.2 Methods

Our goal was to create methods to aid the most common SQL queries. We used the requested queries to define what was considered common in the context of our problem. As such, we implemented four methods, two for the heading_t type and two others for the period_t type.

Listing 2.2: Heading type methods

```
CREATE OR REPLACE TYPE BODY heading_t AS
    MEMBER FUNCTION get_value RETURN NUMBER IS
        res NUMBER;
    BEGIN
        SELECT SUM(VALUE(r).amount) INTO res FROM TABLE(SELF.remunerations) r;
        IF remun_type = 'D' THEN
            RETURN -res;
        ELSE
            RETURN res;
        END IF;
    END get_value;

    MEMBER FUNCTION is_consistent RETURN VARCHAR2 IS
        child_sum NUMBER;
    BEGIN
```

```
            SELECT SUM(VALUE(h).get_value()) INTO child_sum FROM
                TABLE(SELF.childHeadings) h;
            IF SELF.get_value() = child_sum THEN
                RETURN 'T';
            ELSE
                RETURN 'F';
            END IF;
        END is_consistent;
    END;
```

For the heading, we focused on a get_value() method to calculate the sum of all its expenses or remunerations, taking their type into account to decide when to sum or subtract. The other method was to aid in query B, which asked if a heading was consistent with its children, so we simply compare the heading's amount to the sum of its children.

Listing 2.3: Period type methods

```
CREATE OR REPLACE TYPE BODY period_t AS
    MEMBER FUNCTION remunerations_km2(heading_name VARCHAR2, party_name VARCHAR2,
        municipality_code VARCHAR2) RETURN NUMBER IS
        ret_var NUMBER;
    BEGIN
        SELECT SUM(VALUE(e).amount) / MAX(VALUE(l).code.area) INTO ret_var
            FROM TABLE(SELF.expenses) e, TABLE(SELF.leaderships) l
            WHERE VALUE(e).heading.description = heading_name
                AND VALUE(e).code.code = municipality_code
                AND VALUE(l).party.acronym = party_name
                AND VALUE(l).code.code = municipality_code;
        RETURN ret_var;
    END;

    MEMBER FUNCTION remunerations_1k_people(heading_name VARCHAR2, party_name
        VARCHAR2, municipality_code VARCHAR2) RETURN NUMBER IS
        ret_var NUMBER;
    BEGIN
        SELECT SUM(VALUE(e).amount) / MAX(VALUE(l).code.population) * 1000 INTO
            ret_var
            FROM TABLE(SELF.expenses) e, TABLE(SELF.leaderships) l
            WHERE VALUE(e).heading.description = heading_name
                AND VALUE(e).code.code = municipality_code
                AND VALUE(l).party.acronym = party_name
                AND VALUE(l).code.code = municipality_code;
        RETURN ret_var;
    END;
END;
```

Regarding the period, the methods implemented are mainly helpers to aggregate data, such as calculate the remuneration per kilometre squared for a given party and municipality, or the remuneration per heading type, party and municipality.

3 Data Fetching

Only parts of the script are represented here, the full script is available at */scripts/fetch.sql* in the project folder.

During the fetching period, it is needed to convert columns that represent relations (foreign keys) into references to objects. However, symmetrical references are also explicitly added but in a full system should be covered by triggers that updated those references.

Listing 3.1: Data Fetching

```
...
-- Headings
DECLARE
    heading_ref    REF heading_t;
BEGIN
    FOR cur_rec IN (
        SELECT *
            FROM GTD12.headings h
            ORDER BY h.hlevel ASC
    )
    LOOP
        BEGIN
            CASE cur_rec.hlevel
            WHEN 1
            THEN
                INSERT INTO headings (headingId, description, remun_type,
                    childHeadings, remunerations)
                    VALUES (cur_rec.headingId, cur_rec.description, cur_rec.type,
                        headings_tab(), aremuneration_tab());
            ELSE
                INSERT INTO headings h (headingId, description, remun_type,
                    parentHeading, childHeadings, remunerations)
                    VALUES (cur_rec.headingId, cur_rec.description, cur_rec.type, (
                        SELECT REF(hp)
                            FROM headings hp
                            WHERE hp.headingId = cur_rec.parent
                    ), headings_tab(), aremuneration_tab()) RETURNING REF(h) INTO
                        heading_ref;

                INSERT INTO TABLE(
                    SELECT hp.childHeadings
                        FROM headings hp
                        WHERE hp.headingId = cur_rec.parent
                ) VALUES (heading_ref);
            END CASE;
        END;
    END LOOP;
END;
...
```

4 Queries

To properly test the queries' results, we implemented them in both the original relational model and our OR representation. Below are included only the SQL queries for the object relational.

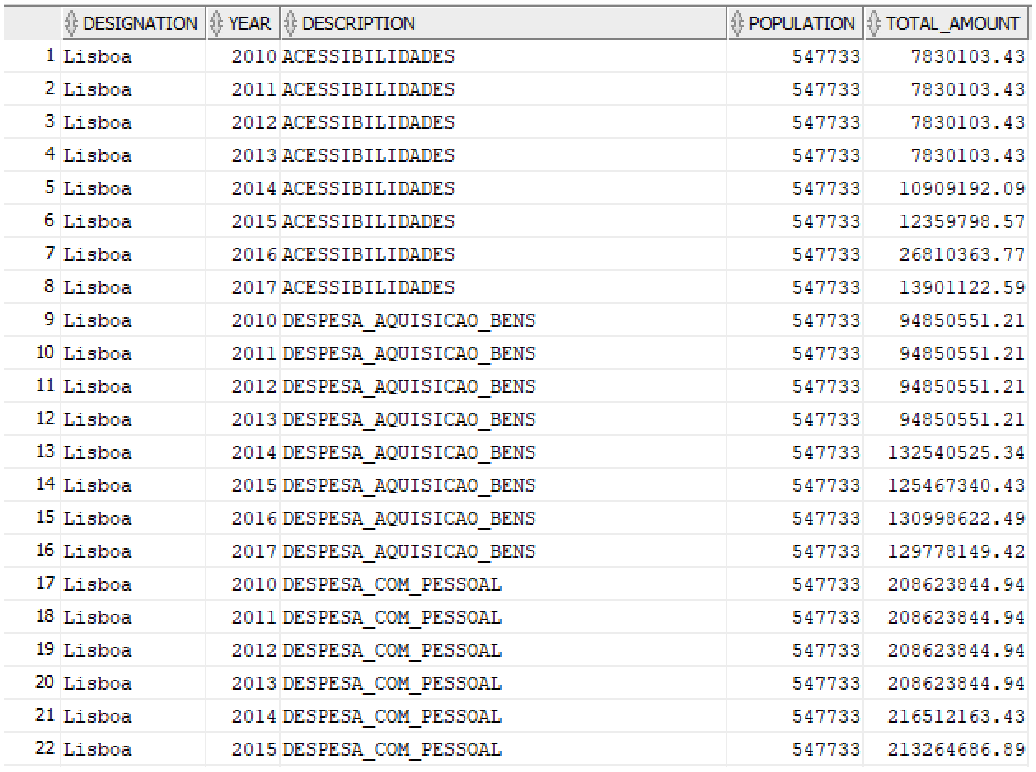
4.1 Query A

This query was left almost intact in the translation from the relational query. Only thing to note is the avoidance of all join operations, as references to expenses are stored in the municipality and the expense has reference to all the other needed tables as well.

'Calculate the expenses by period and by heading of the municipalities of each region. Order municipalities by decreasing population.'

Listing 4.1: OR Query A

```
SELECT m.designation, VALUE(e).period.year year, VALUE(e).heading.description
    description, SUM(VALUE(e).amount) total_amount
FROM municipalities m, TABLE(m.expenses) e
GROUP BY VALUE(e).period.year, VALUE(e).heading.description, m.designation,
    m.population
ORDER BY m.population DESC;
```



	DESIGNATION	YEAR	DESCRIPTION	POPULATION	TOTAL_AMOUNT
1	Lisboa	2010	ACESSIBILIDADES	547733	7830103.43
2	Lisboa	2011	ACESSIBILIDADES	547733	7830103.43
3	Lisboa	2012	ACESSIBILIDADES	547733	7830103.43
4	Lisboa	2013	ACESSIBILIDADES	547733	7830103.43
5	Lisboa	2014	ACESSIBILIDADES	547733	10909192.09
6	Lisboa	2015	ACESSIBILIDADES	547733	12359798.57
7	Lisboa	2016	ACESSIBILIDADES	547733	26810363.77
8	Lisboa	2017	ACESSIBILIDADES	547733	13901122.59
9	Lisboa	2010	DESPESA_AQUISICAO_BENS	547733	94850551.21
10	Lisboa	2011	DESPESA_AQUISICAO_BENS	547733	94850551.21
11	Lisboa	2012	DESPESA_AQUISICAO_BENS	547733	94850551.21
12	Lisboa	2013	DESPESA_AQUISICAO_BENS	547733	94850551.21
13	Lisboa	2014	DESPESA_AQUISICAO_BENS	547733	132540525.34
14	Lisboa	2015	DESPESA_AQUISICAO_BENS	547733	125467340.43
15	Lisboa	2016	DESPESA_AQUISICAO_BENS	547733	130998622.49
16	Lisboa	2017	DESPESA_AQUISICAO_BENS	547733	129778149.42
17	Lisboa	2010	DESPESA_COM_PESSOAL	547733	208623844.94
18	Lisboa	2011	DESPESA_COM_PESSOAL	547733	208623844.94
19	Lisboa	2012	DESPESA_COM_PESSOAL	547733	208623844.94
20	Lisboa	2013	DESPESA_COM_PESSOAL	547733	208623844.94
21	Lisboa	2014	DESPESA_COM_PESSOAL	547733	216512163.43
22	Lisboa	2015	DESPESA_COM_PESSOAL	547733	213264686.89

Figure 3: Results of Query A (200 total rows)

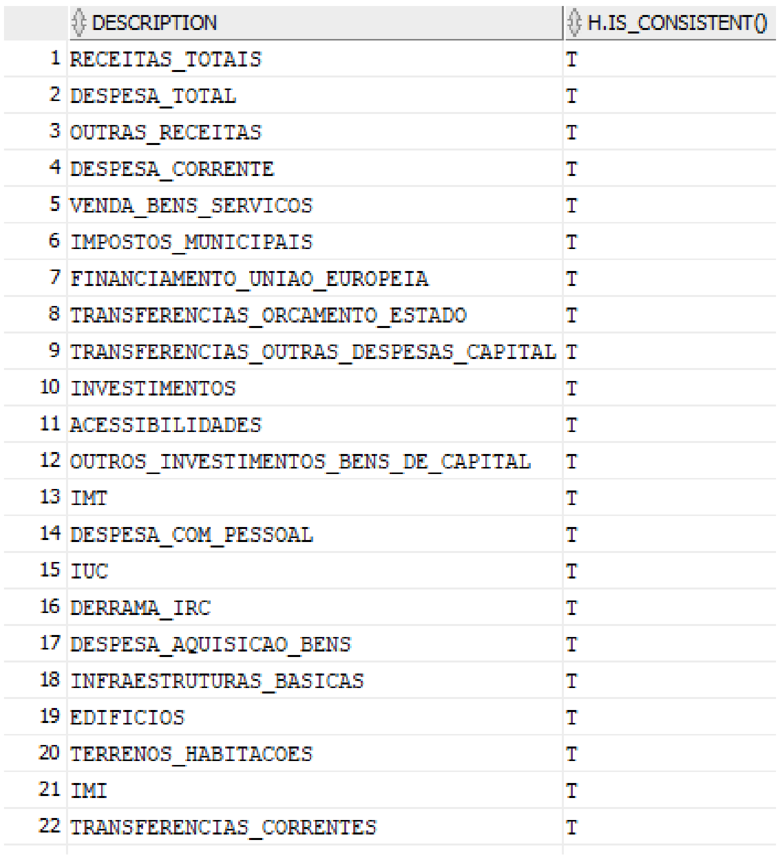
4.2 Query B

By making use of the method created *is_consistent* the query can be simplified to single line, as all the information needed to perform the check is stored within the object.

'Check whether the higher level headings values are consistent with the corresponding lower values.'

Listing 4.2: OR Query B

```
SELECT h.description, h.is_consistent() FROM headings h;
```



	DESCRIPTION	H.IS_CONSISTENT()
1	RECEITAS_TOTAIS	T
2	DESPESA_TOTAL	T
3	OUTRAS_RECEITAS	T
4	DESPESA_CORRENTE	T
5	VENDA_BENS_SERVICOS	T
6	IMPOSTOS_MUNICIPAIS	T
7	FINANCIAMENTO_UNIAO_EUROPEIA	T
8	TRANSFERENCIAS_ORCAMENTO_ESTADO	T
9	TRANSFERENCIAS_OUTRAS_DESPESAS_CAPITAL	T
10	INVESTIMENTOS	T
11	ACESSIBILIDADES	T
12	OUTROS_INVESTIMENTOS_BENS_DE_CAPITAL	T
13	IMT	T
14	DESPESA_COM_PESSOAL	T
15	IUC	T
16	DERRAMA_IRC	T
17	DESPESA_AQUISICAO_BENS	T
18	INFRAESTRUTURAS_BASICAS	T
19	EDIFICIOS	T
20	TERRENOS_HABITACOES	T
21	IMI	T
22	TRANSFERENCIAS_CORRENTES	T

Figure 4: Results of Query B (24 total rows)

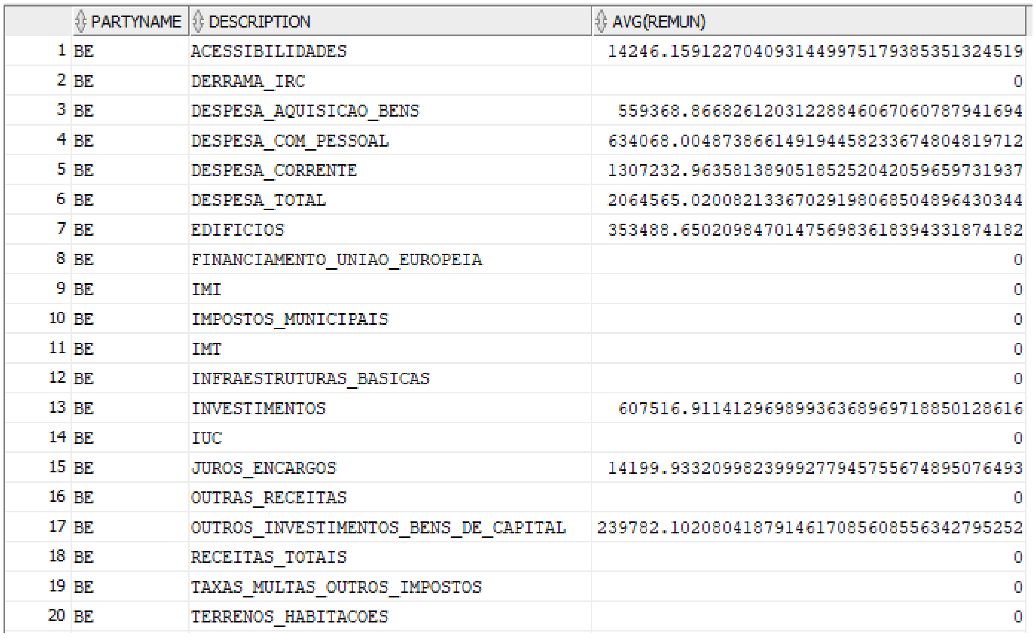
4.3 Query C

This query and the two following ones (D and E) all went through the same translation process. The methods created in *period_t* allows us to extract the metrics needed for each heading, each municipality and each year. From there it was only needed to aggregate data appropriately, similarly to the second step of the relational query.

Which the average expense by thousand inhabitants on each heading for each party?

Listing 4.3: OR Query C

```
WITH aux_view (partyName, description, remun) AS (
    SELECT VALUE(l).party.partyName partyName, h.description,
        SUM(p.remunerations_1k_people(h.description, VALUE(l).party.acronym,
        VALUE(l).code.code))
    FROM periods p, TABLE(p.leaderships) l, headings h
    GROUP BY VALUE(l).party.partyName, h.description
)
SELECT partyName, description, AVG(remun)
FROM aux_view v
GROUP BY partyName, description
ORDER BY partyName, description;
```



	PARTYNAME	DESCRIPTION	AVG(REMUN)
1	BE	ACESSIBILIDADES	14246.1591227040931449975179385351324519
2	BE	DERRAMA_IRC	0
3	BE	DESPESA_AQUISICAO_BENS	559368.86682612031228846067060787941694
4	BE	DESPESA_COM_PESSOAL	634068.004873866149194458233674804819712
5	BE	DESPESA_CORRENTE	1307232.96358138905185252042059659731937
6	BE	DESPESA_TOTAL	2064565.02008213367029198068504896430344
7	BE	EDIFICIOS	353488.650209847014756983618394331874182
8	BE	FINANCIAMENTO_UNIAO_EUROPEIA	0
9	BE	IMI	0
10	BE	IMPOSTOS_MUNICIPAIS	0
11	BE	IMT	0
12	BE	INFRAESTRUTURAS_BASICAS	0
13	BE	INVESTIMENTOS	607516.91141296989936368969718850128616
14	BE	IUC	0
15	BE	JUROS_ENCARGOS	14199.9332099823999277945755674895076493
16	BE	OUTRAS_RECEITAS	0
17	BE	OUTROS_INVESTIMENTOS_BENS_DE_CAPITAL	239782.1020804187914617085608556342795252
18	BE	RECEITAS_TOTAIS	0
19	BE	TAXAS_MULTAS_OUTROS_IMPOSTOS	0
20	BE	TERRENOS_HABITACOES	0

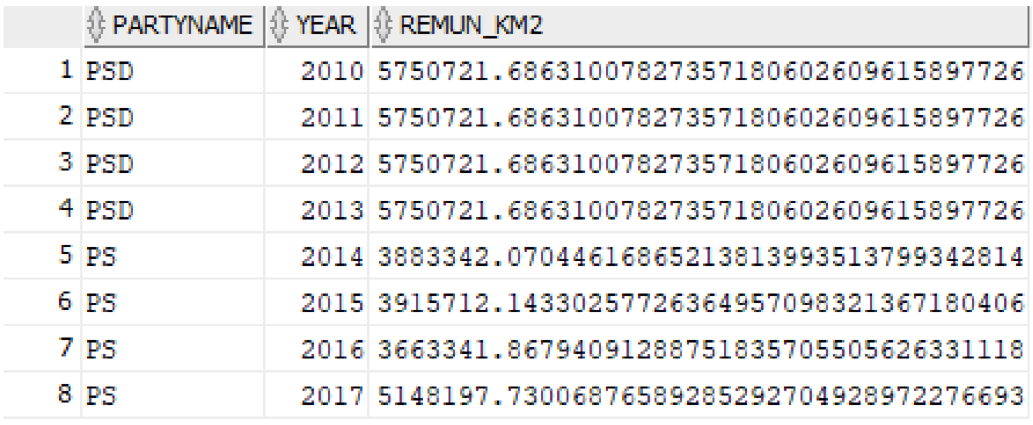
Figure 5: Results of Query C (144 total rows)

4.4 Query D

'Which is the party with more investment per square km on each year?Order municipalities by decreasing population.'

Listing 4.4: OR Query D

```
WITH aux_view (partyName, year, remun_km2) AS (
    SELECT VALUE(l).party.partyName partyName, p.year,
        SUM(p.remunerations_km2('INVESTIMENTOS', VALUE(l).party.acronym,
        VALUE(l).code.code))
    FROM periods p, TABLE(p.leaderships) l
    GROUP BY VALUE(l).party.partyName, p.year
)
SELECT partyName, year, remun_km2
FROM aux_view v
WHERE (v.year, v.remun_km2) IN (
    SELECT vmax.year, MAX(vmax.remun_km2)
           FROM aux_view vmax
           GROUP BY vmax.year
)
ORDER BY year;
```



	PARTYNAME	YEAR	REMUN_KM2
1	PSD	2010	5750721.68631007827357180602609615897726
2	PSD	2011	5750721.68631007827357180602609615897726
3	PSD	2012	5750721.68631007827357180602609615897726
4	PSD	2013	5750721.68631007827357180602609615897726
5	PS	2014	3883342.07044616865213813993513799342814
6	PS	2015	3915712.14330257726364957098321367180406
7	PS	2016	3663341.86794091288751835705505626331118
8	PS	2017	5148197.73006876589285292704928972276693

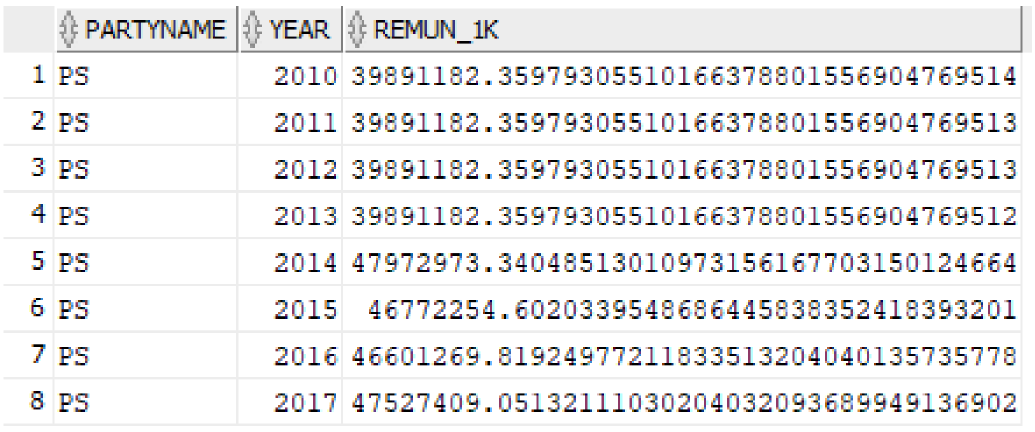
Figure 6: Results of Query D (8 total rows)

4.5 Query E

'Which is the party with more salaries per thousand inhabitants on each year?'

Listing 4.5: OR Query E

```
WITH aux_view (partyName, year, remun_1k) AS (
    SELECT VALUE(l).party.partyName partyName, p.year,
        SUM(p.remunerations_1k_people('DESPESA_COM_PESSOAL', VALUE(l).party.acronym,
        VALUE(l).code.code))
    FROM periods p, TABLE(p.leaderships) l
    GROUP BY VALUE(l).party.partyName, p.year
)
SELECT partyName, year, remun_1k
FROM aux_view v
WHERE (v.year, v.remun_1k) IN (
    SELECT vmax.year, MAX(vmax.remun_1k)
           FROM aux_view vmax
           GROUP BY vmax.year
)
ORDER BY year;
```



	PARTYNAME	YEAR	REMUN_1K
1	PS	2010	39891182.35979305510166378801556904769514
2	PS	2011	39891182.35979305510166378801556904769513
3	PS	2012	39891182.35979305510166378801556904769513
4	PS	2013	39891182.35979305510166378801556904769512
5	PS	2014	47972973.34048513010973156167703150124664
6	PS	2015	46772254.6020339548686445838352418393201
7	PS	2016	46601269.81924977211833513204040135735778
8	PS	2017	47527409.05132111030204032093689949136902

Figure 7: Results of Query E (8 total rows)

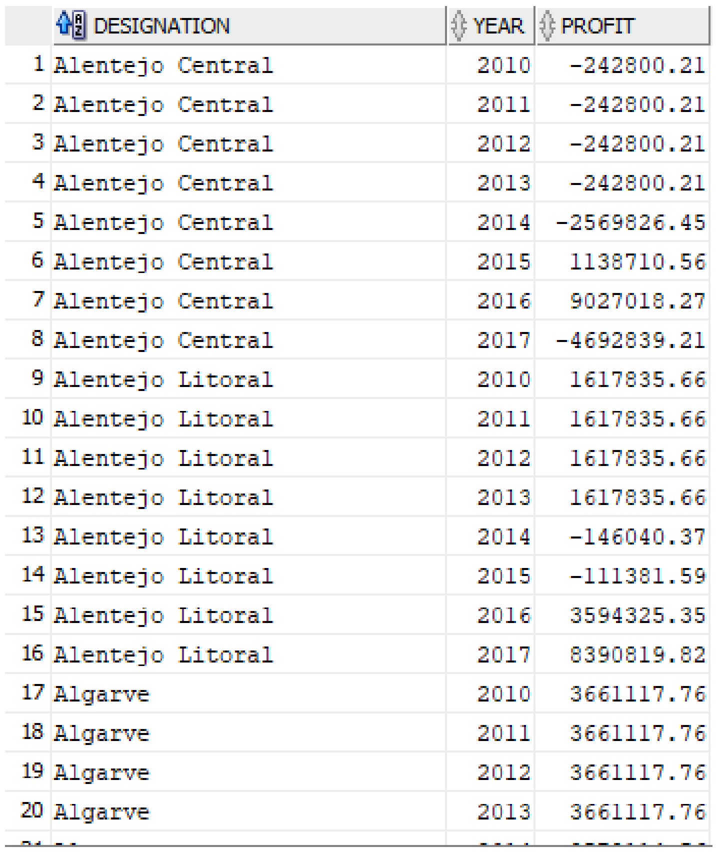
4.6 Query F

The object relational model allows us to avoid leading with joins on tables that are exclusive with each other (*AREVENUES* and *AEXPENSES*), and neatly treat them a similar way as they are stored in a table of references with their context already defined. Furthermore, the hierarchicy defined for the NUTS regions allows us to save a subquery executed to fetch those.

‘Add a query that illustrates the use of OR extensions.’ ‘Profit of each NUTS III for each year’

Listing 4.6: OR Query F

```
SELECT n3.designation, VALUE(e).period.year AS year, SUM(VALUE(r).amount) -
    SUM(VALUE(e).amount) AS profit
    FROM nuts_3 n3, TABLE(n3.municipalities) m, TABLE(VALUE(m).expenses) e,
        TABLE(VALUE(m).revenues) r
    WHERE VALUE(e).heading.description = 'DESPESA_TOTAL' AND
        VALUE(r).heading.description = 'RECEITAS_TOTAIS'
        AND VALUE(e).period.year = VALUE(r).period.year
    GROUP BY n3.designation, VALUE(e).period.year;
```



	DESIGNATION	YEAR	PROFIT
1	Alentejo Central	2010	-242800.21
2	Alentejo Central	2011	-242800.21
3	Alentejo Central	2012	-242800.21
4	Alentejo Central	2013	-242800.21
5	Alentejo Central	2014	-2569826.45
6	Alentejo Central	2015	1138710.56
7	Alentejo Central	2016	9027018.27
8	Alentejo Central	2017	-4692839.21
9	Alentejo Litoral	2010	1617835.66
10	Alentejo Litoral	2011	1617835.66
11	Alentejo Litoral	2012	1617835.66
12	Alentejo Litoral	2013	1617835.66
13	Alentejo Litoral	2014	-146040.37
14	Alentejo Litoral	2015	-111381.59
15	Alentejo Litoral	2016	3594325.35
16	Alentejo Litoral	2017	8390819.82
17	Algarve	2010	3661117.76
18	Algarve	2011	3661117.76
19	Algarve	2012	3661117.76
20	Algarve	2013	3661117.76

Figure 8: Results of Query F (200 total rows)

5 Conclusion

Although the Object Relational extensions have some interesting properties, the Relational model already can represent most problems. While the methods offered by OR can be helpful to uniformize and simplify some queries, the uncommon syntax and lacking documentation presented many challenges. The OR queries also performed significantly slower than the original relational queries, mainly on the queries that use function methods.

The OR also requires extra care when maintaining references, demanding the use of triggers to ensure the consistency of the database. Coupled with the syntax problems mentioned above, we struggled to obtain the reference of new objects inserted into the table, preventing the implementation of the triggers we desired.