Département : IFA

Module : BDA

Année Scolaire :2022-2023

PROJET

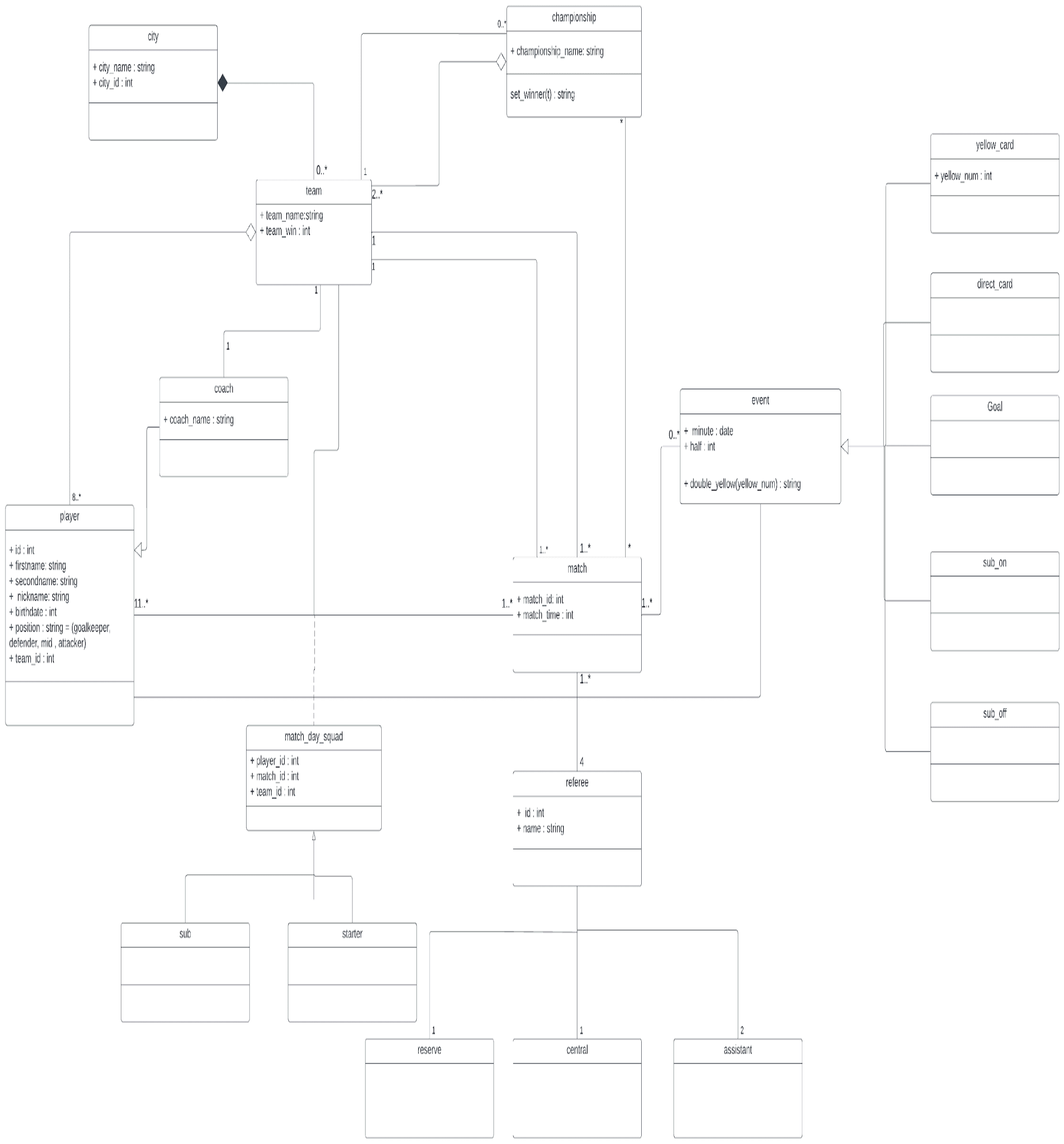
REALISE PAR:

Halladj Hamza.

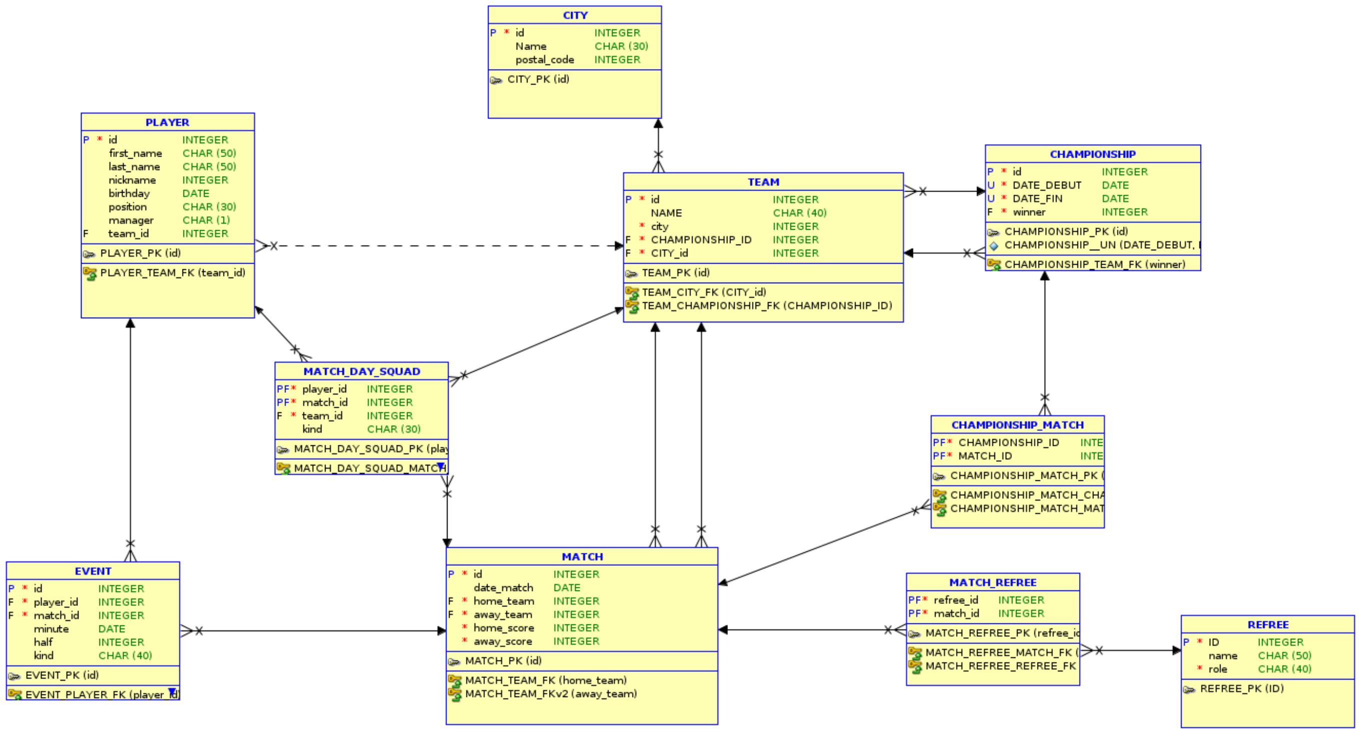
Supervisé par:

Mme.Raida ElMansour

Part One : UML CLASS DIAGRAM



Part Two : THE RELATIONAL MODEL



UML CLASS DIAGRAM TO REALTIONAL MODEL :

Transforming a class diagram into a relational model involves mapping the classes, attributes, associations, and inheritance relationships to tables, columns, and relationships in a relational database. Here is a summary of the transformation process:

1. Classes: Each class in the class diagram becomes a table in the relational model. The class name typically becomes the table name.
2. Attributes: Each attribute in a class becomes a column in the corresponding table. The attribute name usually becomes the column name, and the attribute type determines the column data type.
3. Associations: Associations between classes are represented using foreign keys in the relational model. If a class has a reference to another class, the primary key of the referenced class becomes a foreign key in the referring class's table.
4. Multiplicity and Cardinality: The multiplicity and cardinality of associations determine the type of relationship between tables. For example, a one-to-one association may be

represented by including a foreign key column in one of the tables involved, while a one-to-many association may be represented by having the foreign key in the "many" side table.

5. Inheritance: Inheritance relationships can be represented in different ways. The two common approaches are "class table inheritance" and "single table inheritance." In class table inheritance, each subclass has its own table, including its inherited attributes. In single table inheritance, all attributes from all classes in the inheritance hierarchy are included in a single table.
6. Primary Keys: Each table requires a primary key that uniquely identifies each row. The primary key can be a single column or a combination of columns.
7. Normalization: Apply normalization techniques to ensure data integrity and eliminate redundancy. This involves splitting tables, creating additional tables, and adjusting relationships based on normalization rules.

So from all the above we concluded this:

CITY(id, Name, postal_code)

PLAYER(id , first_name , last_name , nickname , birthday,position,manager,#team_id)

TEAM(id , NAME ,city, #CHAMPIONSHIP_ID, #CITY_id)

CHAMPIONSHIP(id, DATE_DEBUT,DATE_FIN, #winner)

EVENT(id , #player_id , #match_id,minute,half,kind)

MATCH(id , #home_team, ,#away_team,date_match,home_score,away_score)

MATCH_DAY_SQUAD(#player_id , #match_id , #team_id,kind)

CHOMPIONSHIP_MATCH(#CHAMPIONSHIP_ID , #MATCH_ID)

REFREE(ID, name , date_retour , role)

MATCH_REFREE(#refree_id, #match_id)

IMPLIMENTATION:

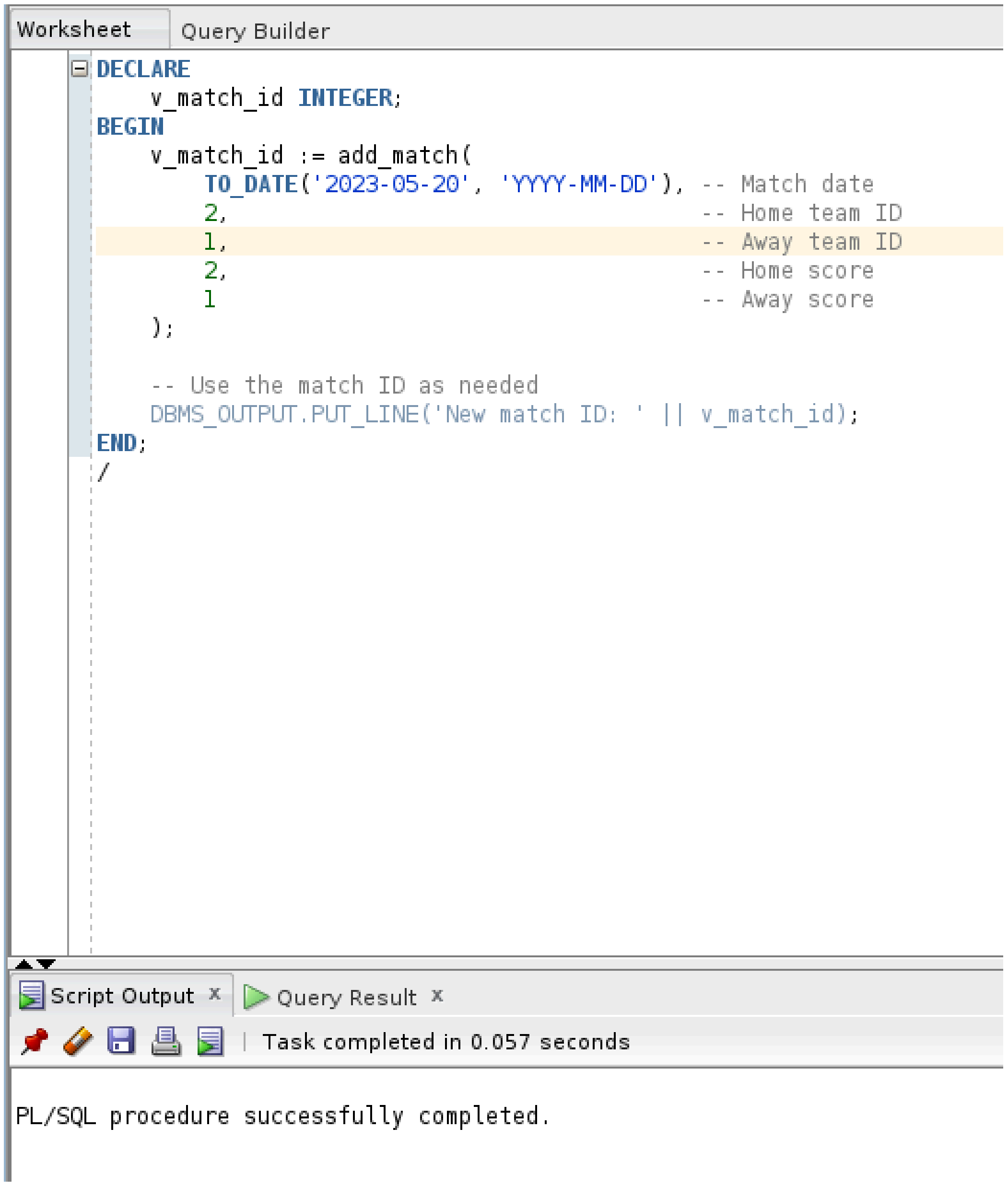
We will translate the model above into the SQL language as follows:

- Classes -> SQL tables
- Attributes -> Columns in each table
- Methods -> Here, we know that methods in a class diagram, when transformed into SQL, become procedures or functions. This will be implemented in question 2-2. And here we are going write just the header of each trigger and procedure and in the next sections we will explain more the implementation.

```
CREATE OR REPLACE TRIGGER championship_id_trg BEFORE
    INSERT ON championship
    FOR EACH ROW
    WHEN ( new.id IS NULL )

CREATE OR REPLACE TRIGGER city_id_trg BEFORE
    INSERT ON city
    FOR EACH ROW
    WHEN ( new.id IS NULL )

CREATE OR REPLACE TRIGGER event_id_trg BEFORE
    INSERT ON event
    FOR EACH ROW
    WHEN ( new.id IS NULL )

CREATE OR REPLACE TRIGGER match_id_trg BEFORE
    INSERT ON match
    FOR EACH ROW
    WHEN ( new.id IS NULL )

CREATE OR REPLACE TRIGGER player_id_trg BEFORE
    INSERT ON player
    FOR EACH ROW
    WHEN ( new.id IS NULL )

CREATE OR REPLACE TRIGGER refree_id_trg BEFORE
    INSERT ON refree
    FOR EACH ROW
    WHEN ( new.id IS NULL )

CREATE OR REPLACE TRIGGER team_id_trg BEFORE
    INSERT ON team
    FOR EACH ROW
    WHEN ( new.id IS NULL )
BEGIN
    :new.id := team_id_seq.nextval;
END;
```

PROCEDURE OF FILLING TABLES:

Here is an example of filling table procedure :



```
DECLARE
    v_match_id INTEGER;
BEGIN
    v_match_id := add_match(
        TO_DATE('2023-05-20', 'YYYY-MM-DD'), -- Match date
        2,                                   -- Home team ID
        1,                                   -- Away team ID
        2,                                   -- Home score
        1                                    -- Away score
    );

    -- Use the match ID as needed
    DBMS_OUTPUT.PUT_LINE('New match ID: ' || v_match_id);
END;
/
```

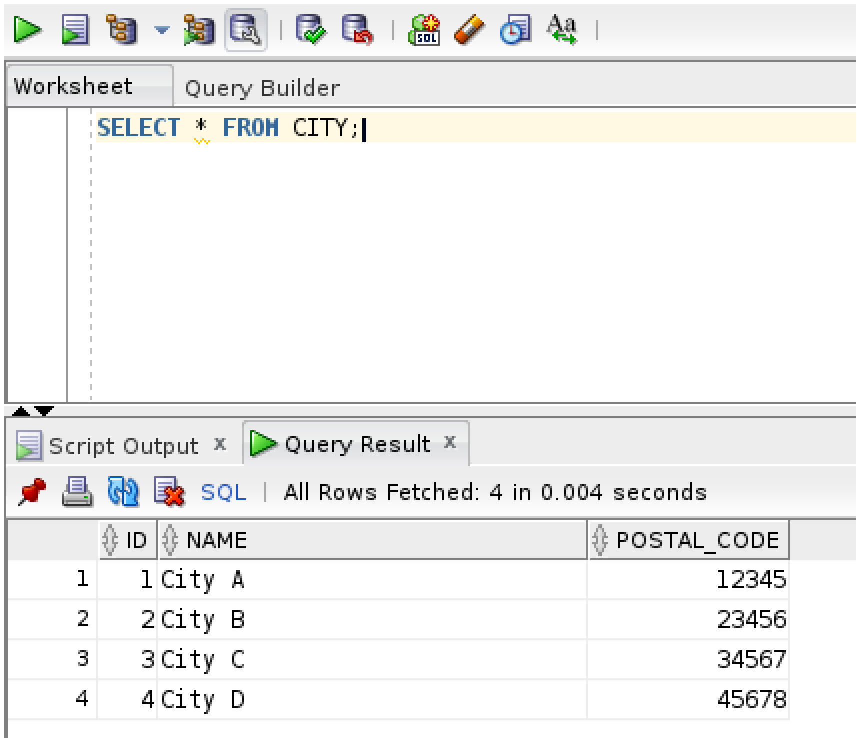
```
Task completed in 0.057 seconds

PL/SQL procedure successfully completed.
```

We have filled the tables in the same manner.

Populate the obtained database:

here we populated our Data base tables, as example “CITY and TEAM” with random values as you can see using the method above:

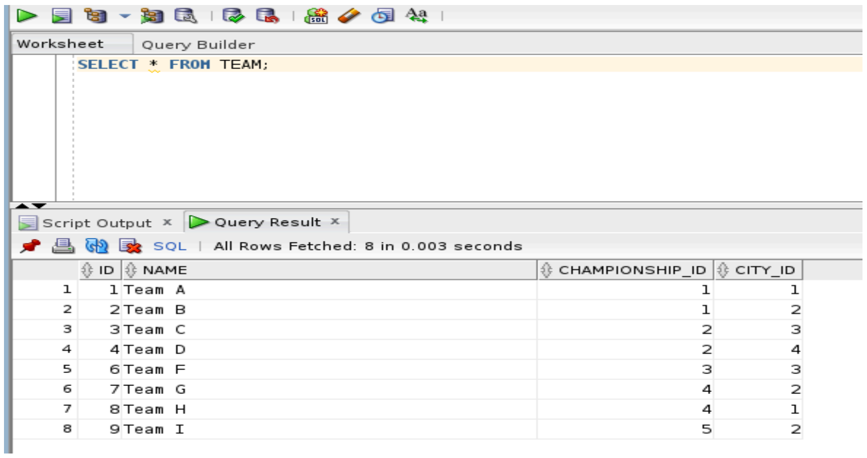
// used the select command to extract data form db



```
SELECT * FROM CITY;
```

SQL | All Rows Fetched: 4 in 0.004 seconds

	ID	NAME	POSTAL_CODE
1	1	City A	12345
2	2	City B	23456
3	3	City C	34567
4	4	City D	45678



```
SELECT * FROM TEAM;
```

SQL | All Rows Fetched: 8 in 0.003 seconds

	ID	NAME	CHAMPIONSHIP_ID	CITY_ID
1	1	Team A	1	1
2	2	Team B	1	2
3	3	Team C	2	3
4	4	Team D	2	4
5	6	Team F	3	3
6	7	Team G	4	2
7	8	Team H	4	1
8	9	Team I	5	2

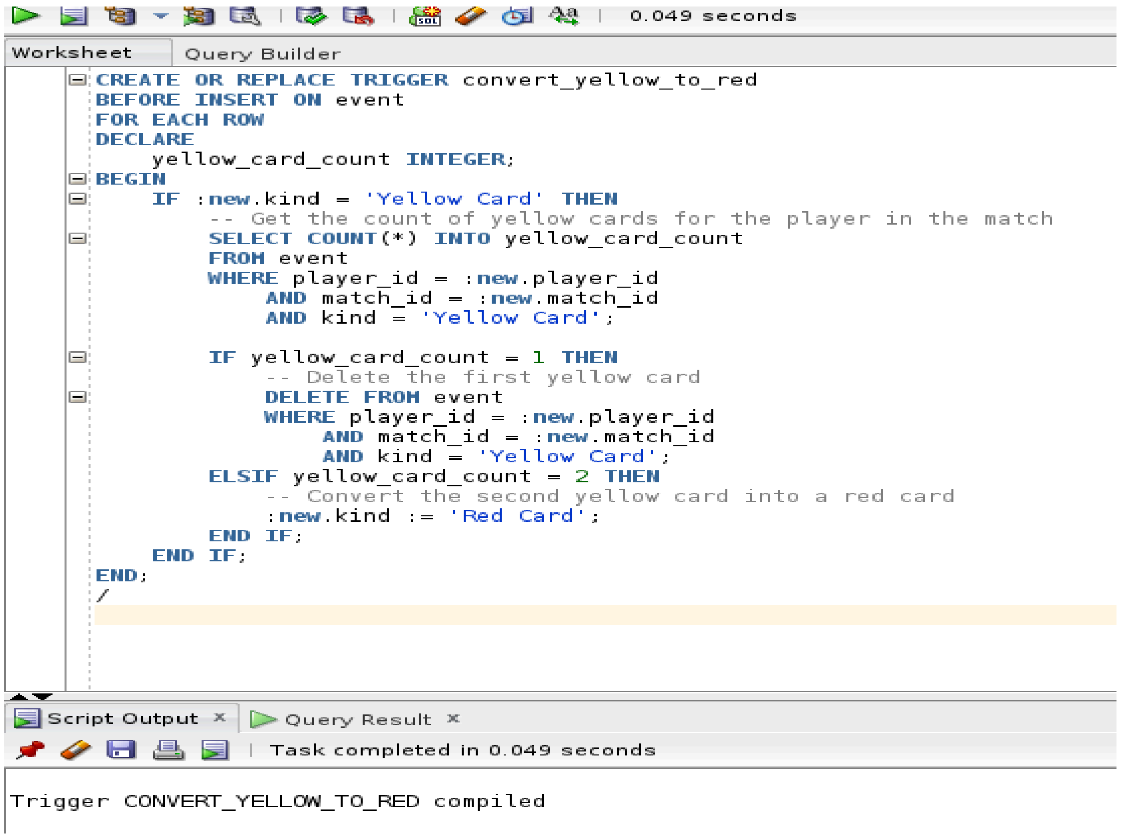
And so on for the other tables..

IMPLIMENTATION OF TRIGGERS USED:

```
CREATE OR REPLACE TRIGGER championship_id_trg BEFORE
    INSERT ON championship
    FOR EACH ROW
    WHEN ( new.id IS NULL )
BEGIN
    :new.id := championship_id_seq.nextval;
END;
/

CREATE OR REPLACE TRIGGER city_id_trg BEFORE
    INSERT ON city
    FOR EACH ROW
    WHEN ( new.id IS NULL )
BEGIN
    :new.id := city_id_seq.nextval;
END;
/

CREATE OR REPLACE TRIGGER event_id_trg BEFORE
    INSERT ON event
    FOR EACH ROW
    WHEN ( new.id IS NULL )
BEGIN
    :new.id := event_id_seq.nextval;
END;
/

CREATE OR REPLACE TRIGGER match_id_trg BEFORE
    INSERT ON match
    FOR EACH ROW
    WHEN ( new.id IS NULL )
BEGIN
    :new.id := match_id_seq.nextval;
END;
/

CREATE OR REPLACE TRIGGER player_id_trg BEFORE
    INSERT ON player
    FOR EACH ROW
    WHEN ( new.id IS NULL )
BEGIN
    :new.id := player_id_seq.nextval;
END;
/

CREATE OR REPLACE TRIGGER refree_id_trg BEFORE
    INSERT ON refree
    FOR EACH ROW
    WHEN ( new.id IS NULL )
BEGIN
    :new.id := refree_id_seq.nextval;
```

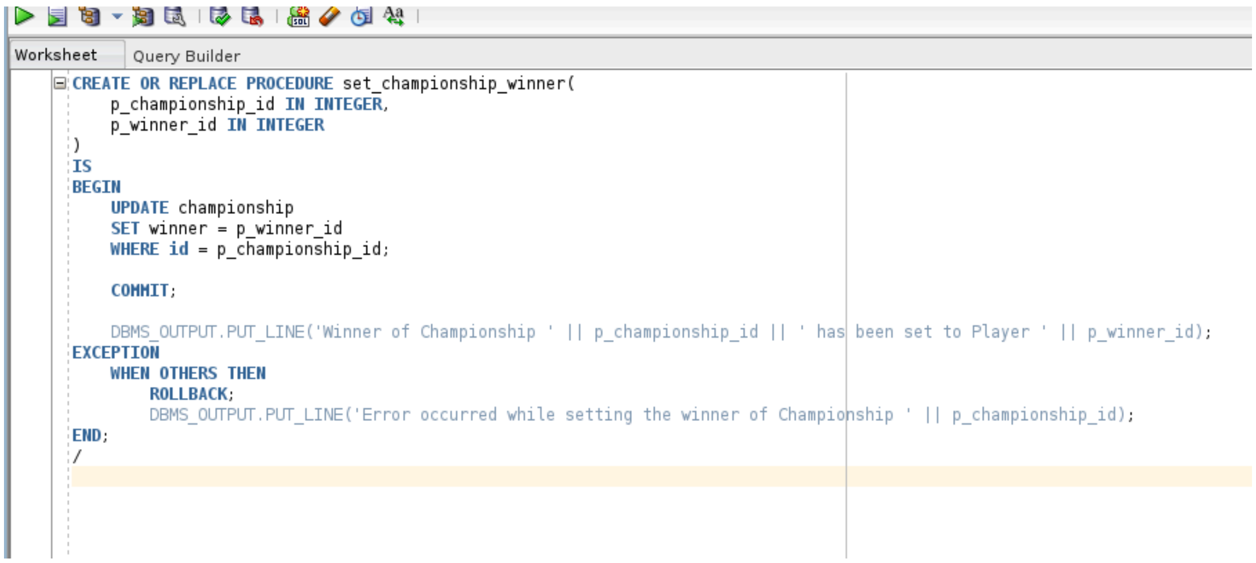
```
END;
/

CREATE OR REPLACE TRIGGER team_id_trg BEFORE
    INSERT ON team
    FOR EACH ROW
    WHEN ( new.id IS NULL )
BEGIN
    :new.id := team_id_seq.nextval;
END;
```

Examples of procedure and functions to meet the requirements requested:



```
CREATE OR REPLACE TRIGGER convert_yellow_to_red
BEFORE INSERT ON event
FOR EACH ROW
DECLARE
    yellow_card_count INTEGER;
BEGIN
    IF :new.kind = 'Yellow Card' THEN
        -- Get the count of yellow cards for the player in the match
        SELECT COUNT(*) INTO yellow_card_count
        FROM event
        WHERE player_id = :new.player_id
            AND match_id = :new.match_id
            AND kind = 'Yellow Card';

        IF yellow_card_count = 1 THEN
            -- Delete the first yellow card
            DELETE FROM event
            WHERE player_id = :new.player_id
                AND match_id = :new.match_id
                AND kind = 'Yellow Card';
        ELSIF yellow_card_count = 2 THEN
            -- Convert the second yellow card into a red card
            :new.kind := 'Red Card';
        END IF;
    END IF;
END;
/
```

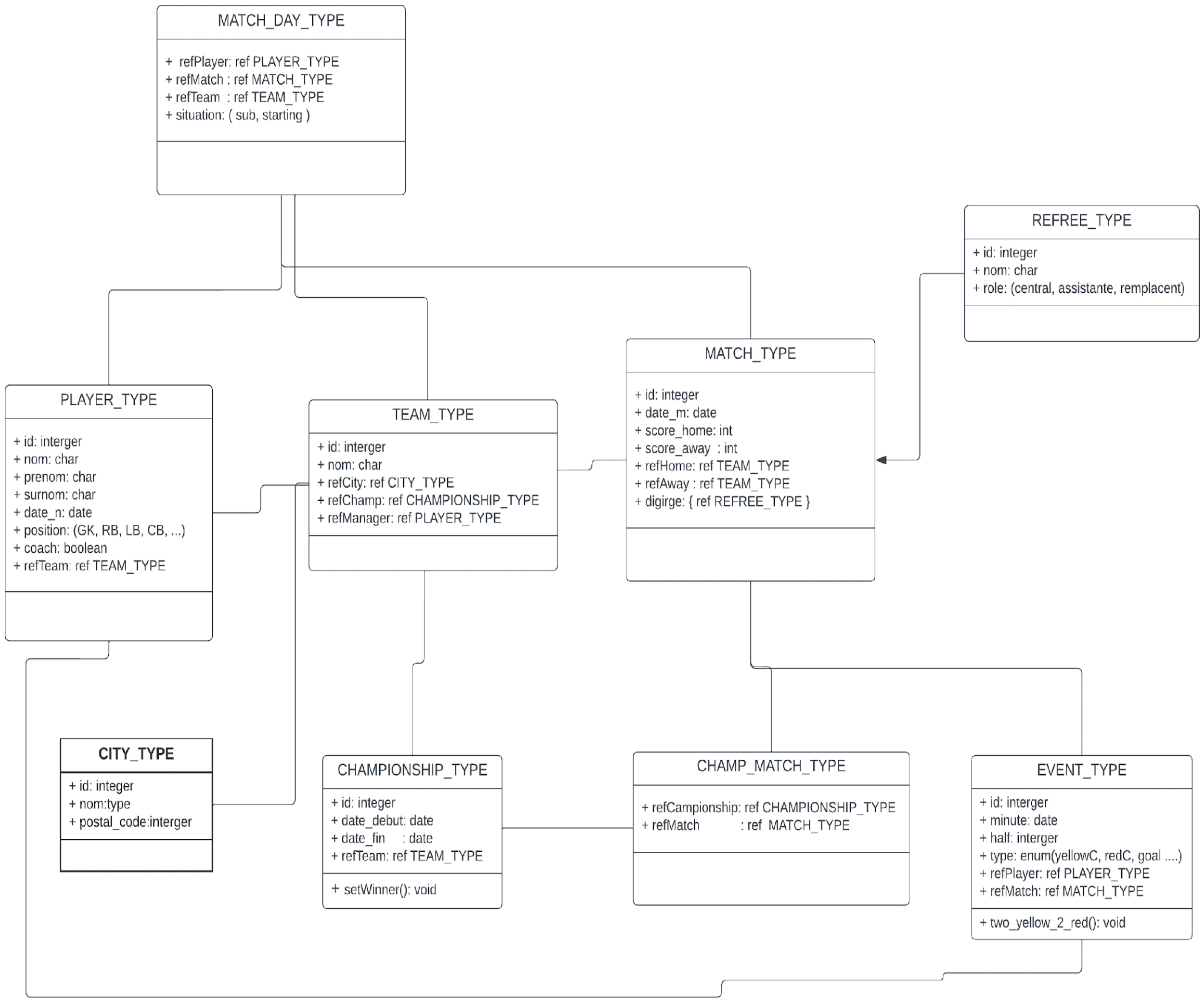
```
Trigger CONVERT_YELLOW_TO_RED compiled
```



```
CREATE OR REPLACE PROCEDURE set_championship_winner(
    p_championship_id IN INTEGER,
    p_winner_id IN INTEGER
)
IS
BEGIN
    UPDATE championship
    SET winner = p_winner_id
    WHERE id = p_championship_id;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Winner of Championship ' || p_championship_id || ' has been set to Player ' || p_winner_id);
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error occurred while setting the winner of Championship ' || p_championship_id);
END;
/
```

...

PART THREE :RELATIONAL OBJECT MODEL



2-Implimentation:

Creating table city:

```
CREATE TABLE city OF city_type(
    CONSTRAINT pk_city PRIMARY KEY(id)
);
```

Creating table team:

```
CREATE TABLE team OF team_type(
    CONSTRAINT pk_team PRIMARY KEY(id)
);
```

Creating table event:

```
create table event of event_type(
    constraint pk_event primary key(id)
);
```

Creating table championship:

```
create table championship of championship_type(
    constraint pk_champ primary key(id)
);
```

Creating table player:

```

create table player of player_type(
    constraint pk_player primary key(id)
);
```

Creating table match:

```
create table match of match_type(
    constraint pk_match primary key(id)
);
```

Creating table match_day:

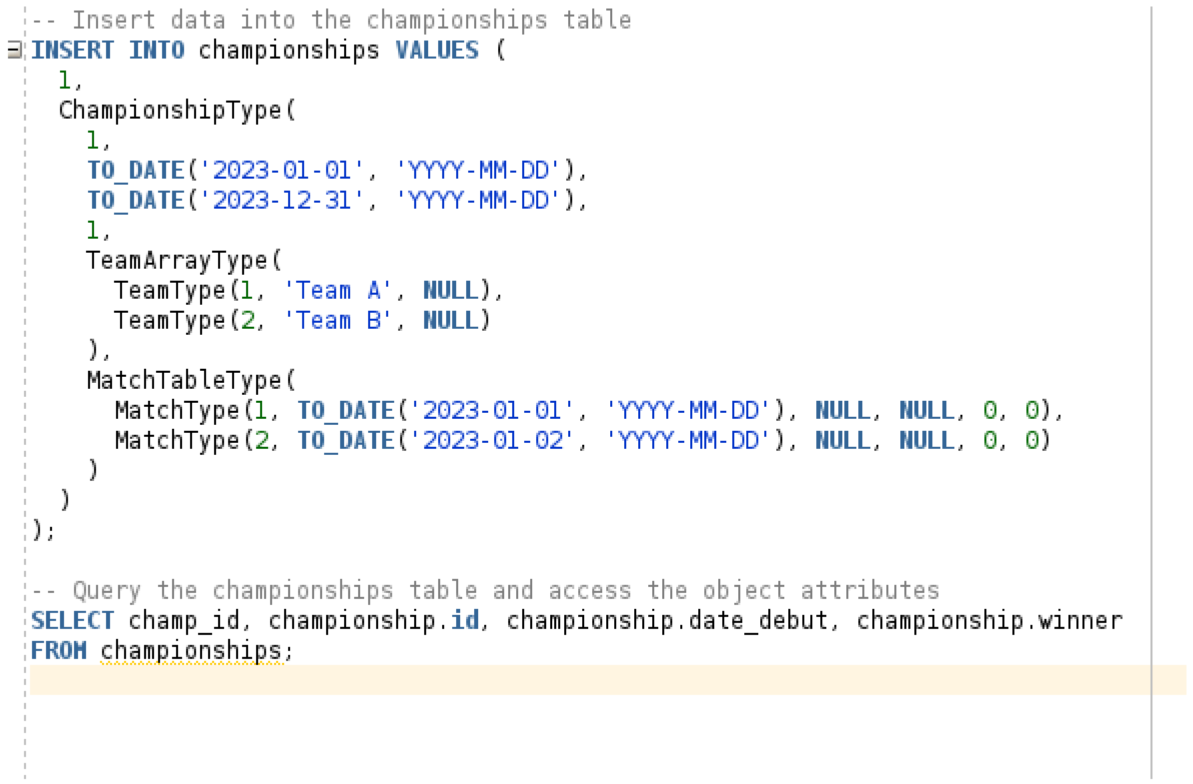
```
create table match_day of match_day_type(
    constraint pk_match_day primary key(refPlayer, refMatch)
);
```

Creating table match_championship:

```
create table match_championship of match_championship_type(
    constraint pk_match_champ primary key(refMatch, refChamp)
);
```

3-POPULATE ROM :

Here is an example of populating an object championships:



```sql
-- Insert data into the championships table
INSERT INTO championships VALUES (
  1,
  ChampionshipType(
    1,
    TO_DATE('2023-01-01', 'YYYY-MM-DD'),
    TO_DATE('2023-12-31', 'YYYY-MM-DD'),
    1,
    TeamArrayType(
      TeamType(1, 'Team A', NULL),
      TeamType(2, 'Team B', NULL)
    ),
    MatchTableType(
      MatchType(1, TO_DATE('2023-01-01', 'YYYY-MM-DD'), NULL, NULL, 0, 0),
      MatchType(2, TO_DATE('2023-01-02', 'YYYY-MM-DD'), NULL, NULL, 0, 0)
    )
  )
);

-- Query the championships table and access the object attributes
SELECT champ_id, championship.id, championship.date_debut, championship.winner
FROM championships;
```