UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

PROGETTO DI "BASI DI DATI E SISTEMI INFORMATIVI"

DOCENTI:

Adriano Peron , Alessandro de Luca

STUDENTI:

Cuomo Daniele N86 1346 , Iervolino Riccardo N86 1608

PROGETTO:

Progetto n°2, seconda parte: Archivio Risiko!

Trigger e funzioni

Descrizione dei trigger e delle funzioni

1

```
CREATE OR REPLACE TRIGGER winner_constraint
```

Questo trigger si attiva quando avviene un tentativo di update sulla colonna winner della tabella game. Prima che il record venga modificato, verifica che la plancia di gioco all'ultimo turno giocato rispretti la condizione di vittoria del goal del winner candidato.
Per farlo seleziona nella variabile locale g_val il valore della carta goal del :NEW.winner e in last_turn il massimo valore di turno per la partita con :OLD.codgm.
Subito dopo aver ricavato questi due dati tramite interrogazione del database li utilizza come parametri della funzione Goal_respected(:NEW.winner, g_val, last_turn) al termine della quale effettuerà o meno l'update.

```
create or replace TRIGGER winner_constraint
  BEFORE UPDATE OF winner ON game
  FOR EACH ROW
  BEGIN
  DECLARE
    g_val NUMBER;
    last_turn NUMBER;
  BEGIN
       SELECT R.Goal INTO g_val FROM RECAP R
       WHERE R.game = :OLD.codGm AND R.player = :NEW.winner;

       SELECT MAX(nTurn) INTO last_turn
       FROM TURN
       WHERE OFgame = :OLD.codGm;

       IF NOT goal_respected( :NEW.winner , g_val , last_turn ) THEN
         :NEW.winner := :OLD.winner;
       ELSE
         UPDATE PLAYER SET numwins = numwins + 1
         WHERE username = :NEW.winner;
       END IF;

   END;
 END;
```

1.1

```
CREATE OR REPLACE FUNCTION goal_respected( W GAME.winner%TYPE , G GOALCARD.codgoal%TYPE ,
T TURN.codTr%TYPE  )
```

Questa funzione chiamata dal trigger winner_constraint controlla che l'obiettivo G del giocatore W sia conseguito in base alla condizione della plancia all'ultimo turno T giocato della partita.
Il codice dell'obiettivo servirà a selezionare il controllo da eseguire tra i seguenti:
Per gli obiettivi che richiedono un certo numero di territori si effettua una count dei possedimenti, mentre nei casi in cui è richiesto di possedere interi continenti la funzione chiama n_continents, che controlla proprio questo.
Infine vi sono i controlli per i sei obiettivi che richiedono di eliminare le armate di un particolare colore, per farlo si utilizza la funzione elemination.

```
CREATE OR REPLACE FUNCTION goal_respected( W GAME.winner%TYPE , G GOALCARD.codgoal%TYPE ,
T TURN.codTr%TYPE  )
 RETURN BOOLEAN IS
 B BOOLEAN;
 curr_gm NUMBER;  n_terr NUMBER;
 BEGIN
   IF G = 0 THEN
     SELECT COUNT(O.codOwn) INTO n_terr FROM OWNERSHIP O
     WHERE O.O_turn = T AND O.owner = W AND O.numArmies > 1;
     B := (n_terr > 17);
   ELSIF G = 1 THEN
     SELECT COUNT(O.codOwn) INTO n_terr
     FROM OWNERSHIP O WHERE O.O_turn = T AND O.owner = W;
     B := (n_terr > 23);
   ELSIF G = 2 THEN
     B := n_continents( 'Nord America' , W , T , 9 ) AND n_continents( 'Africa' , W , T , 6 );
   ELSIF G = 3 THEN
     B := n_continents( 'Nord America' , W , T , 9 ) AND n_continents( 'Oceania' , W , T , 4 );
   ELSIF G = 4 THEN
     B := n_continents( 'Sud America' , W , T , 4 ) AND n_continents( 'Asia' , W , T , 12 );
   ELSIF G = 5 THEN
     B := n_continents( 'Asia' , W , T , 12 ) AND n_continents( 'Africa' , W , T , 6 );
   ELSIF G = 6 THEN
     B := n_continents( 'Europa' , W , T , 7 ) AND n_continents( 'Sud America' , W , T , 4 );
     IF B THEN
       B := n_continents( 'Nord America' , W , T , 9 ) OR n_continents( 'Asia' , W , T , 12 )
            OR n_continents( 'Oceania' , W , T , 4 ) OR n_continents( 'Africa' , W , T , 6 );
     END IF;
   ELSIF G = 7 THEN
     B := n_continents( 'Europa' , W , T , 7 ) AND n_continents( 'Oceania' , W , T , 4 );
     IF B THEN
       B := n_continents( 'Nord America' , W , T , 9 ) OR n_continents( 'Asia' , W , T , 12 )
            OR n_continents( 'Sud America' , W , T , 4 ) OR n_continents( 'Africa' , W , T , 6 );
     END IF;
   ELSE
     SELECT TR.Game INTO curr_gm FROM TURN TR WHERE TR.codTr = T;
     FOR I IN 0..5 LOOP
       IF G = I + 8 THEN
         B := elimination ( W , I , T , curr_gm );  END IF;
     END LOOP;
   END IF;
   RETURN B;
 END;
```

1.2

```
CREATE OR REPLACE FUNCTION n_continents( continent VARCHAR2, winner GAME.winner%TYPE ,
last_turn TURN.codTr%TYPE , tot NUMBER  )
```

La funzione è progettata per avere in input il nome del giocatore candidato a possedere il continente, il nome del continente e il numero di territori (tot) che lo compongono. Con una semplice select ricava il numero di territori posseduti nel continente da winner e restituisce il risultato del confronto tra tot e il valore ricavato dalla select.

```
CREATE OR REPLACE FUNCTION n_continents( cont VARCHAR2, w GAME.winner%TYPE , last_turn
TURN.codTr%TYPE , tot NUMBER  )
 RETURN BOOLEAN IS
 n_terr NUMBER;
 BEGIN
  SELECT COUNT(O.codOwn) INTO n_terr
  FROM OWNERSHIP O JOIN TERRITORY T on O.territory = T.tname
  WHERE (T.Continent = cont) AND O.O_turn = last_turn AND O.owner = w;
  RETURN (n_terr = tot);
 END;
```

1.3

```
CREATE OR REPLACE FUNCTION elimination( winner GAME.winner%TYPE , colour RECAP.colour%TYPE,
last_turn TURN.codTr%TYPE, game RECAP.GAME%TYPE )
```

La prima select ricava il giocatore con il colore da eliminare secondo l'obiettivo in analisi, mentre la seconda somma tutte le armate del suddetto giocatore in relazione a last_turn. La tabella Ownership in last_turn contiene possedimenti di loser anche se questi è stato effettivamente eliminato, solo che il numero delle armate nei territori conquistati sarà uguale a 0, di conseguenza la somma delle armate dei suoi possedimenti darà risultato non nullo e uguale a 0 se è stato effettivamente eliminato.
Siccome il turno in cui loser ha perso è necessariamente il turno "attivo" del giocatore che lo elimina, questi viene considerato vincitore in modo definitivo.

```
CREATE OR REPLACE FUNCTION elimination( winner GAME.winner%TYPE , colour RECAP.colour%TYPE,
last_turn TURN.codTr%TYPE, game RECAP.GAME%TYPE )
 RETURN BOOLEAN IS
 loser RECAP.player%TYPE;  army NUMBER;  n_terr NUMBER;
 BEGIN
  SELECT R.Player INTO loser
  FROM RECAP R JOIN TURN T ON R.game = T.OFGame
  WHERE R.colour = colour AND last_turn = T.codTr;
  SELECT SUM(O.numarmies) INTO army
  FROM OWNERSHIP O
  WHERE O.owner = loser AND O.O_turn = last_turn;
  IF army = 0 THEN RETURN TRUE;ELSE RETURN FALSE; END IF;
  EXCEPTION WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Obiettivo cambiato in -conquista 24 territori-');
    SELECT COUNT(O.codOwn) INTO n_terr
    FROM OWNERSHIP O WHERE O.O_turn = last_turn AND O.owner = winner;
    RETURN (n_terr > 23);
 END;
```

2

```
CREATE OR REPLACE TRIGGER change_goal
```

L'applicazione è programmata in modo tale che ogni partita sia prima composta dai suoi giocatori, e quindi avviata, con la modifica del loro numero (numPL), una volta terminati gli inserimenti su RECAP.
In questa fase di avvio viene eseguito il trigger change_goal il quale controlla, eseguendo le funzioni kill_my_self e kill_nobody (entrambe con ritorno di tipo booleano), se ogni giocatore con l'obiettivo di distruggere le armate di un particolare colore sia nelle condizioni di vincere.
Nel caso in cui le funzioni dovessero restituire valore positivo il trigger imposta in automatico l'obiettivo 1 (conquistare 24 territori) come il gioco stesso prevede.

```
create or replace TRIGGER change_goal
  BEFORE UPDATE OF numPl ON GAME
  FOR EACH ROW
  BEGIN
    DECLARE
      CURSOR curr IS (SELECT * FROM RECAP R
                      WHERE R.game= :OLD.codGm);
    BEGIN
      FOR i IN curr LOOP
        IF kill_my_self( i ) OR kill_nobody( i ) THEN
          UPDATE RECAP SET goal = 1
          WHERE player = i.player AND game = i.game;
        END IF;
      END LOOP;
    END;
  END;
```

2.1

```
CREATE OR REPLACE FUNCTION kill_my_self( tuple RECAP%ROWTYPE)
```

Restituisce true quando l'obiettivo del player in RECAP richiede di eliminare il giocatore col suo stesso colore sfruttando la codifica degli obiettivi.

```
create or replace FUNCTION kill_my_self( tuple RECAP%ROWTYPE )
  RETURN BOOLEAN IS
  BEGIN
      RETURN (tuple.goal = (tuple.colour + 8));
  END;
```

2.2

```
CREATE OR REPLACE FUNCTION kill_nobody( tuple RECAP%ROWTYPE )
```

La funzione restituisce true qualora non dovessero esserci per il game in corso tuple di RECAP con il colore descritto nell'obiettivo

```
create or replace FUNCTION kill_nobody( tuple RECAP%ROWTYPE )
  RETURN BOOLEAN IS
  colour RECAP.colour%TYPE;
  ck BOOLEAN;
  CURSOR cur IS (SELECT R.colour FROM RECAP R
                 WHERE tuple.game = R.game);
```

```
  BEGIN
    OPEN cur;
    LOOP
      FETCH cur INTO colour;
      EXIT WHEN cur%NOTFOUND OR ck;
      ck := (tuple.goal - 8 = colour);
    END LOOP;
    RETURN not ck;
  END;
```

3

```
CREATE OR REPLACE TRIGGER battle_manage
```

Battle_manage ha lo scopo di aggiornare la tabella OWNERSHIP quando avviene un'insert nella talbella battle (infatti tale insert andrà necessariamente a modificare i possedimenti del turno, almeno per quanto riguarda il numero di carrarmati sui territori in guerra).
Inizialmente ordina le stringhe nelle quali sono salvati i lanci dei dadi con la funzione dice_sort, subito dopo aggiorna numarmies di OWNERSHIP in base alle perdite militari controllando che non vi siano perdite dovute alla struttura della stringa. Infine controlla che il difensore sia stato sconfitto, in tal caso assegna all'attaccante il territorio conquistato con una sola armata, l'applicazione java servirà poi a decidere quante armate spostare dal territorio di striker al nuovo.

```
create or replace TRIGGER battle_manage
BEFORE INSERT ON BATTLE
FOR EACH ROW
BEGIN
  DECLARE
    blue NUMBER := 0;
    red NUMBER := 0;
    strike OWNERSHIP%ROWTYPE;
    defend OWNERSHIP%ROWTYPE;
  BEGIN
    :NEW.redDice := dice_sort( :NEW.redDice );
    :NEW.blueDice := dice_sort( :NEW.blueDice );
    FOR i IN 1..3 LOOP
      IF substr(:NEW.redDice, (i*2)-1, 1) > substr(:NEW.blueDice ,(i*2)-1, 1) and
      substr(:NEW.redDice, (i*2)-1, 1) > 0 and substr(:NEW.blueDice ,(i*2)-1, 1) > 0 THEN
        blue := blue + 1;
      ELSE
        red := red + 1;
      END IF;
    END LOOP;
    defend := kill_armies( blue , :NEW.defender );
    strike := kill_armies( red , :NEW.striker );
    dbms_output.put_line( defend.numArmies );
    IF defend.numArmies = 0 THEN
      INSERT INTO OWNERSHIP VALUES
        ( count_own.NEXTVAL , defend.Territory , strike.Owner , 1 , defend.O_Turn );
      UPDATE OWNERSHIP SET numArmies = numArmies - 1
            WHERE CODOWN = :NEW.striker;
    END IF;
  END;
END;
```

3.1

```
CREATE OR REPLACE FUNCTION dice_sort( dice IN OUT VARCHAR2 )
```

Funzione per l'ordinamento decrescente dei valori dei dadi, che l'utente può inserire intervallati da virgole in una stringa.

```
create or replace FUNCTION dice_sort( dice IN OUT VARCHAR2 )
  RETURN VARCHAR2 IS
  curr_i CHAR(1); curr_c CHAR(1); curr_j CHAR(1); curr_max CHAR(1); curr_min CHAR(1);
  BEGIN
      /* I valori sono separati da virgole */
      curr_i := substr( dice , 1 , 1 );
      curr_max := curr_i; curr_min := curr_i;
      curr_c:=curr_i;
      FOR j IN 2..3 LOOP
        curr_j := substr( dice , (j*2)-1 , 1 );
        IF curr_j > curr_max THEN
          IF curr_max != curr_c THEN curr_c :=curr_max; END IF;
          curr_max := curr_j;

        ELSE if curr_j < curr_min  THEN
          IF curr_min != curr_c THEN curr_c :=curr_min; END IF;
          curr_min := curr_j;
        ELSE
          curr_c := curr_j;
        END IF;
        END IF;
      END LOOP;
      /*---swap---*/
      dice := (curr_max || ','||curr_c || ','||curr_min);
    RETURN dice;
  END;
```

3.2

```
CREATE OR REPLACE FUNCTION kill_armies( n NUMBER , fighter OWNERSHIP.codOwn%TYPE )
```

Funzione per l'update del numero di unità sul territorio dopo una battaglia, prende in input il numero di carrarmatini distrutti e codice del possedimento del giocatore in battaglia nel turno in corso.

```
create or replace FUNCTION kill_armies( n NUMBER , fighter OWNERSHIP.codOwn%TYPE )
  RETURN OWNERSHIP%ROWTYPE IS
  owner OWNERSHIP%ROWTYPE;
  armies NUMBER;
  BEGIN
    SELECT O.numArmies INTO armies
    FROM OWNERSHIP O
    WHERE O.codOwn = fighter;
    armies := armies - n;
    IF armies < 0 THEN armies := 0;  END IF;
    UPDATE OWNERSHIP SET numarmies = armies
    WHERE codown = fighter;
    SELECT * INTO owner
    FROM OWNERSHIP O
    WHERE O.codOwn = fighter;
    RETURN owner;
  END;
```

4 FUNZIONI USATE DALL'APPLICAZIONE JAVA

CREATE OR REPLACE FUNCTION not_duplicate(C TERRITORYCARD.codCard%TYPE , T TURN.codTr%TYPE)

Funzione utilizzata dall'applicazione java per verificare che una carta in input C, non sia in mano ad alcun giocatore durante il turno T.

```
CREATE OR REPLACE FUNCTION not_duplicate( C TERRITORYCARD.codCard%TYPE , T
TURN.codTr%TYPE )
 RETURN BOOLEAN IS
 card TERRITORYCARD.codCard%TYPE;
 BEGIN
  SELECT H.Card INTO card
  FROM HAND H
  WHERE H.H_Turn = T AND H.Card = C;
  RETURN FALSE;
  EXCEPTION WHEN NO_DATA_FOUND THEN
   RETURN TRUE;
 END;
```

CREATE OR REPLACE FUNCTION next_turn(T TURN%ROWTYPE)

Funzione invocata all'atto di creare un nuovo turno per la partita in corso da un utente dell'applicazione, popolo i possedimenti e le mani del nuovo turno, viene fatta eccezione per i territori con numero di armate uguale a zero, questi infatti sono territori conquistati nel turno precedente e non vanno inseriti tra i possedimenti del nuovo turno.

```
create or replace FUNCTION next_turn( oldTurn TURN.CODTR%TYPE )
    RETURN TURN.CODTR%TYPE AS
    CURSOR crs_h IS
      SELECT H.Card, H.Holder FROM HAND H
      WHERE H.H_Turn = oldTurn;
    CURSOR crs_o IS
      SELECT  O.numArmies , O.Owner , O.Territory FROM OWNERSHIP O
      WHERE O.O_Turn = oldTurn;
      m NUMBER; n NUMBER; l NUMBER;
      p RECAP.Player%TYPE;
      g RECAP.Game%TYPE;
      T TURN.codTr%TYPE;
    BEGIN
      SELECT T2.nTurn + 1 into m
      FROM TURN T2 JOIN RECAP R2 ON T2.OfGame = R2.Game AND T2.CurrPl = R2.Player
      WHERE T2.codTr = oldTurn;

      SELECT MAX(myTurn) into n
      FROM RECAP
      WHERE Game = (SELECT ofGame FROM TURN WHERE codTr = oldTurn);

      l := MOD( m , n ) + 1;

      SELECT Player into p
      FROM RECAP
      WHERE Game = (SELECT ofGame FROM TURN WHERE codTr = oldTurn) AND myTurn = l;
      SELECT Game into g
      FROM RECAP
      WHERE Game = (SELECT ofGame FROM TURN WHERE codTr = oldTurn) AND myTurn = l;

      T := count_turn.NEXTVAL;
```

```
    INSERT INTO TURN VALUES( T , m , p , g );

    FOR I IN crs_h LOOP
      INSERT INTO HAND VALUES
        ( T , I.card , I.Holder );
    END LOOP;
    FOR J IN crs_o LOOP
      IF J.numArmies > 0 THEN
        INSERT INTO OWNERSHIP VALUES
          ( count_own.NEXTVAL , J.Territory , J.Owner , J.numArmies , T );
      END IF;
    END LOOP;
    RETURN T;
  END;
```

5 VINCOLI SULLE TABELLE

```
ALTER TABLE GAME
  ADD CONSTRAINT winner_fk FOREIGN KEY ( Winner ) REFERENCES Player( Username );
```

```
ALTER TABLE TURN ADD CONSTRAINT
game_fk1 FOREIGN KEY ( OFGame ) REFERENCES Game( codGm ) ON DELETE CASCADE;
ALTER TABLE TURN
  ADD CONSTRAINT player_fk1 FOREIGN KEY ( currPl ) REFERENCES Player( Username );
```

```
ALTER TABLE RECAP ADD CONSTRAINT
game_fk2 FOREIGN KEY ( Game ) REFERENCES GAME( codGm ) ON DELETE CASCADE;
```

```
ALTER TABLE RECAP
  ADD CONSTRAINT player_fk2 FOREIGN KEY ( Player ) REFERENCES PLAYER( Username )
  ADD CONSTRAINT goal_fk FOREIGN KEY ( Goal ) REFERENCES GOALCARD( codGoal )
  ADD CONSTRAINT colour_fk FOREIGN KEY ( Colour ) REFERENCES COLOUR( codCl );
```

```
ALTER TABLE TERRITORYCARD
  ADD CONSTRAINT tname_fk FOREIGN KEY ( TC_Name ) REFERENCES TERRITORY( TName );
```

```
ALTER TABLE BORDER
  ADD CONSTRAINT territory_fk1 FOREIGN KEY ( TName1 ) REFERENCES TERRITORY( TName )
  ADD CONSTRAINT territory_fk2 FOREIGN KEY ( TName2 ) REFERENCES TERRITORY( TName );
```

```
ALTER TABLE OWNERSHIP
  ADD CONSTRAINT territory_fk3 FOREIGN KEY ( Territory ) REFERENCES TERRITORY( TName )
  ADD CONSTRAINT owner_fk FOREIGN KEY ( Owner ) REFERENCES PLAYER( Username );
```

```
ALTER TABLE OWNERSHIP ADD CONSTRAINT
turn_fk FOREIGN KEY ( O_Turn ) REFERENCES TURN( codTr ) ON DELETE CASCADE;
```

```
ALTER TABLE BATTLE ADD CONSTRAINT
striker_fk FOREIGN KEY ( Striker ) REFERENCES OWNERSHIP( codOwn ) ON DELETE CASCADE;
```

```
ALTER TABLE BATTLE ADD CONSTRAINT
defender_fk FOREIGN KEY ( Defender ) REFERENCES OWNERSHIP( codOwn ) ON DELETE CASCADE;
```

```
ALTER TABLE BATTLE
```

```
  ADD CONSTRAINT different_ck CHECK ( Striker != Defender );
```

```
ALTER TABLE HAND
  ADD CONSTRAINT card_fk FOREIGN KEY ( Card ) REFERENCES TERRITORYCARD( codCard )
  ADD CONSTRAINT player_fk FOREIGN KEY ( Holder ) REFERENCES PLAYER( Username );
```

```
ALTER TABLE HAND ADD CONSTRAINT
turn_fk2 FOREIGN KEY ( H_Turn ) REFERENCES TURN( codTr ) ON DELETE CASCADE;
```

```
ALTER TABLE RECAP
ADD CONSTRAINT RECAP_UK1 UNIQUE ( GAME, PLAYER )
ADD CONSTRAINT RECAP_UK2 UNIQUE ( GAME, COLOUR )
ADD CONSTRAINT RECAP_UK3 UNIQUE (GAME, MYTURN )
```

```
ALTER TABLE TERRITORYCARD
            ADD CONSTRAINT bonus_ck CHECK Bonus_t IN ('Fante', 'Cannone', 'Cavaliere', 'Jolly'))
```