Relazione per il progetto di Basi di Dati 2

Lorenzo Dainelli (S4489388)
Mattia Dapino (S4482324)

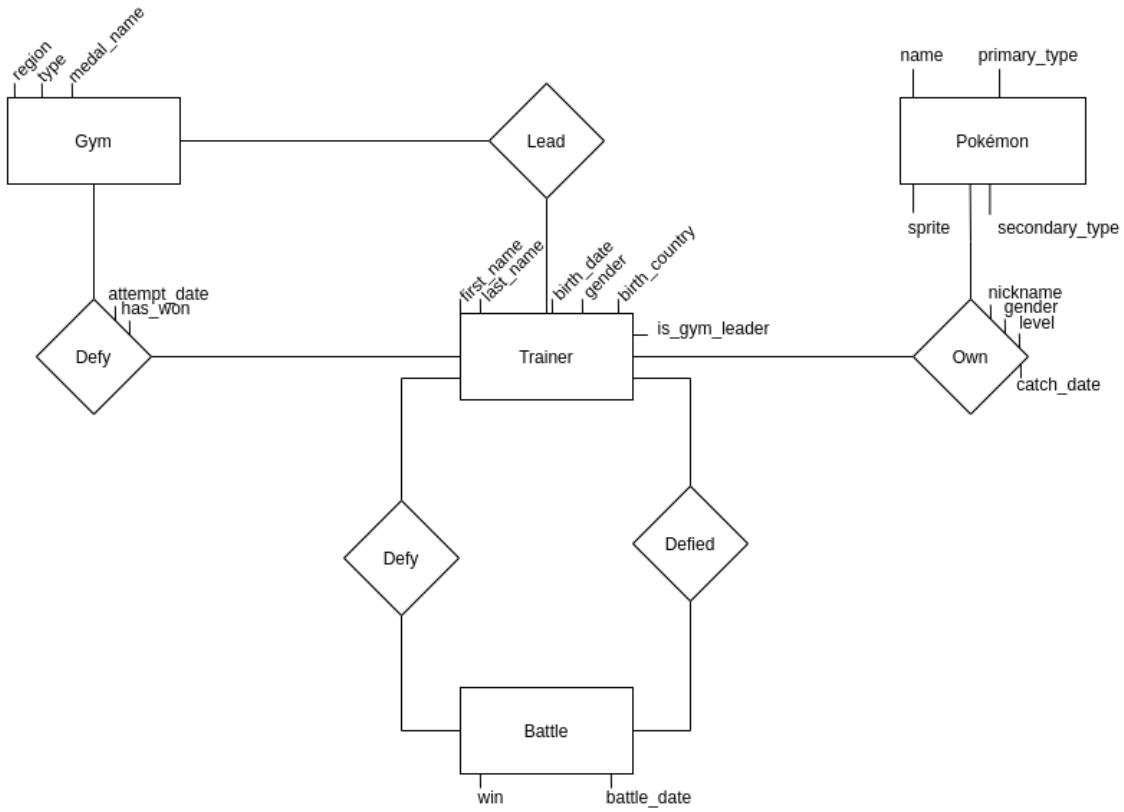
Il sorgente, gli script e la configurazione dell'ambiente sono consultabili sul repository
https://github.com/mitto98/BD2

PARTE A: Scelta dello strumento e creazione della base di dati

A.1 Comprensione dello strumento

A.2 Individuazione della base di dati

Schema relazionale in meta notazione



L'attributo sprite un il dummy field che abbiamo utilizzato per aumentare la dimensione delle tuple della relazione pokemon, siamo consapevoli che non è buona pratica salvare file all'interno di un database.

Codice sql per la creazione dello schema relazionale

```sql
SET search_path TO pokedex;

create table pokemons
(
    id              serial not null,
    name            varchar(255),
```

```
    primary_type   varchar(255),
    secondary_type varchar(255),
    sprite         TEXT
);

create table trainers
(
    id            serial not null,
    first_name    varchar(255),
    last_name     varchar(255),
    birth_date    timestamp,
    gender        char,
    birth_country varchar(255),
    is_gym_leader boolean
);

create table pokemon_trainer
(
    id             serial not null,
    pokemon_id    integer,
    trainer_id    integer,
    nickname      varchar(255),
    gender        char,
    level         integer,
    catch_date    timestamp
);

create table battles
(
    id                serial not null,
    first_trainer_id  integer,
    second_trainer_id integer,
    battle_date       timestamp,
    win               boolean
);

create table gyms
(
    id         serial not null,
    region     varchar(255),
    type       varchar(255),
    medal_name varchar(255),
    gym_leader integer
);

create table trainer_gym
(
    trainer_id   integer,
    gym_id       integer,
    last_attempt timestamp,
    has_won      boolean
);
```

Dimensioni tabelle create

table_name	no_tuple	no_pages	table_size
pokemons	11000	102	816 kB
trainers	150000	1303	10 MB
pokemon_trainer	100000	736	5888 kB
battles	100000	736	5888 kB
gyms	20000	149	1192 kB
trainer_gym	25000	148	1184 kB

(6 rows)

Approccio utilizzato per la generazione dell'istanza
Per la generazione dei dati abbiamo utilizzato un semplece script, scritto da noi, che genera il codice sql delle insert che inseriscono un numero arbitrario di tuple su ogni tabella, in base alla configurazione, è possibile consultare il suddetto script sul repo github indicato.

PARTE B: Elaborazione di interrogazioni

B.1 Definizione del carico di lavoro

- **[I]** Due query utilizzano uno stesso attributo (battles.first_trainer_id e battles.second_trainer_id)
- **[L]** Alemno un attributo per ogni tabella

Query 1

Gli incontri tra allenatori precedenti all'anno 2000

Condizioni soddisfatte:

- **[A]** Condizione di selezione ad alta selettività
- **[D]** Join di tre tabelle

```
SELECT  t1.first_name,
        t1.last_name,
        t2.first_name,
        t2.last_name,
        b.battle_date
FROM pokedex.battles b
         JOIN pokedex.trainers AS t1 ON b.first_trainer_id = t1.id
         JOIN pokedex.trainers AS t2 ON b.second_trainer_id = t2.id
WHERE EXTRACT(YEAR FROM b.battle_date) < 2000;
```

Query 2

I pokemon, esclusi i Missingo, posseduti da piu di 10 allenatori con il relativo conteggio dei proprietari.

Condizioni soddisfatte

- **[B]** Condizione di selezione a bassa selettivita
- **[C]** Join di due tabelle

- **[E]** Raggruppamento

```
SELECT p.id, p.name, count(pt.id)
FROM pokedex.pokemons p
    JOIN pokedex.pokemon_trainer AS pt ON p.id = pt.pokemon_id
WHERE NOT p.name LIKE 'Missingno %'
GROUP BY p.id, p.name
HAVING count(p.id) > 10
ORDER BY count(pt.id) DESC;
```

Query 3

Il numero di incontri di ogni allenatore

Condizioni soddisfatte:

- **[A]** Condizione di selezione ad alta selettività
- **[G]** Sottointerrogazione correlata

```
SELECT t.id,
       t.first_name,
       t.last_name,
       (SELECT count(*)
        FROM pokedex.battles b
        WHERE b.second_trainer_id = t.id OR b.first_trainer_id = t.id) AS battles
FROM pokedex.trainers t;
```

Query 4

Gli allenatori che hanno vinto un incontro in una palestra di tipo roccia

Condizioni soddisfatte:

- **[F]** Sottointerrogazione semplice

```
SELECT id, first_name, last_name
FROM pokedex.trainers
WHERE id IN (SELECT id
               FROM pokedex.gyms
               WHERE type LIKE 'rock');
```

Query 5

Gli allenatori che hanno vinto almeno un incontro in palestra

Condizioni soddisfatte:

- **[M]** Contiene la clausola distinct

```
SELECT DISTINCT trainer_id FROM pokedex.trainer_gym WHERE has_won;
```

Query 6

Le sfide lanciate dagli allenatori di cognome Panini in cui hanno vinto

Condizioni soddisfatte:

- **[C]** Join di due tabelle
- **[F]** Sottointerrogazione semplice

```
SELECT t.id, t.first_name, t.last_name, b.id
FROM (SELECT * FROM pokedex.trainers WHERE last_name LIKE 'Panini') t
  JOIN pokedex.battles AS b ON b.first_trainer_id = t.id
WHERE b.win;
```

Query 7

I pokemon femmina di livello superiore al 50 posseduti da un allenatore di nome 'Goofy'

Condizioni soddisfatte:

- **[M]** Contiene la clausola distinct
- **[H]** Compare un join e almeno due condizioni di selezione

```
SELECT DISTINCT p.name FROM pokedex.pokemons p
JOIN pokedex.pokemon_trainer AS pt ON pt.pokemon_id = p.id
JOIN pokedex.trainers AS t ON t.id = pt.trainer_id
WHERE pt.gender = 'F' AND pt.level > 50 and t.first_name = 'Goofy';
```

Query 8

Gli allenatori che non hanno mai battuto nessuna palestra

Condizioni soddisfatte:

- **[F]** Sottointerrogazione semplice

```
SELECT *
FROM pokedex.trainers
WHERE id NOT IN (SELECT DISTINCT trainer_id
                 FROM pokedex.trainer_gym
                 WHERE NOT has_won IS TRUE);
```

Appendice

FILE: labo.java

```
/* Esempio di programma che si connette a DB attraverso JDBC e esegue una insert di
conto  */


import java.sql.*;
import java.io.*;
import java.util.*;

class labo {


    public static void main(String args[]) {

        // APERTURA CONNESSIONE -- CODICE DA MODIFICARE CON VOSTRI DATI
```

```
        String url = "jdbc:postgresql://130.251.61.30/YOUR_DBNAME";
        //es. db bd2user_n
        String user = "YOUR_USERNAME"; //es. bd2userxx
        String pass = "YOUR_PASSWORD"; //es. BD2userxx


        Connection conn = null;

        // CARICAMENTO DRIVER

        try {
            Class.forName("org.postgresql.Driver");

            // CONNESSIONE

            conn = DriverManager.getConnection(url, user, pass);

            // EVENTUALE VARIAZIONE SCHEMA

            //PreparedStatement st = conn.prepareStatement("set search_path to
account");
            //st.executeUpdate();

            // INIZIALIZZAZIONE AUTOCOMMIT A FALSE PER IMPOSTARE COMPORTAMENTO
TRANSAZIONALE
            //conn.setAutoCommit(false);


            // ESECUZIONE COMANDI
            PreparedStatement st1 = conn.prepareStatement("INSERT INTO Account VALUES
(0,0)");
            st1.executeUpdate();
            //PreparedStatement st2 = conn.prepareStatement("SELECT * FROM Account");
            //st2.executeQuery();

            // chiusura connessione
            //    conn.commit();
            conn.close();

        } catch (java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        } catch (SQLException e1) {
            try {
                if (conn != null) conn.rollback();
            } catch (SQLException e) {
                while (e != null) {
                    System.out.println("SQLState: " + e.getSQLState());
                    System.out.println("    Code: " + e.getErrorCode());
                    System.out.println(" Message: " + e.getMessage());
                    e = e.getNextException();
```

```
                }
            }

        }
    }
}
```

FILE: ConcurrentTransactions.java

```
import org.apache.log4j.BasicConfigurator;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

/**
 * <p>
 * Run numThreads transactions, where at most maxConcurrent transactions
 * can run in parallel.
 *
 * <p>params: numThreads maxConcurrent
 */

public class ConcurrentTransactions {


    static final String url = "jdbc:postgresql://localhost:5432/postgres";
    static final String user = "postgres";
    static final String pass = "secret";

    public static void main(String[] args) {

        BasicConfigurator.configure();

        Logger logger = LoggerFactory.getLogger(ConcurrentTransactions.class);

        Connection conn = null;

        StatementFactory statementFactory = StatementFactory.getInstance();


        try {
            //Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(url, user, pass);
            PreparedStatement st = conn.prepareStatement("set search_path to
```

```
pokedex");
            st.executeUpdate();
        } catch (SQLException se) {
            logger.error("Errore SQL durante la preparazione del DB", se);
            System.exit(-1);
        } catch (Exception e) {
            logger.error("Errore durante la preparazione del DB", e);
            System.exit(-1);
        }

        // read command line parameters
        if (args.length != 2) {
            logger.error("Invalid Params: expected <numThreads> <maxConcurrent>");
            System.exit(-1);
        }
        int numThreads = Integer.parseInt(args[0]);
        int maxConcurrent = Integer.parseInt(args[1]);

        // create numThreads transactions
        Transaction[] trans = new Transaction[numThreads];

        for (int i = 0; i < trans.length; i++) {
            trans[i] = new Transaction(i + 1, conn,
statementFactory.getPreparedStatements(3));
        }


        // start all transactions using a connection pool
        ExecutorService pool = Executors.newFixedThreadPool(maxConcurrent);
        for (Transaction tran : trans) {
            pool.execute(tran);
        }
        pool.shutdown(); // end program after all transactions are done

        //CHIUSURA CONNESSIONE
        try {
            if (!pool.awaitTermination(10, TimeUnit.SECONDS)) {
                pool.shutdownNow();
                try {
                    conn.close();
                } catch (SQLException se) {
                    logger.error("Errore SQL", se);
                } catch (Exception e) {
                    logger.error("Errore generico", e);
                }
            }
        } catch (InterruptedException e) {
            logger.error("Errore di interruzione", e);
        }
    }
}
```