UNIVERSITÀ DI PISA

Relazione progetto di Laboratory of Data Science

Tennis Dataset

Group 06
Biviano Matteo 543933
Graziani Alice 544557

Anno Accademico 2021/2022

Indice

1 Creazione del datawarehouse

In questa sezione verranno descritti i passi effettuati per costruire e popolare il database, partendo da una fase di analisi preliminare dei dati presente nel file **understanding.py** (unico file nel quale è stata utilizzata la libreria *pandas*). In particolare, con la funzione **substitute_nan**, all'interno del file, vengono sostituiti i missing values, per gli attributi winner_rank, loser_rank, winner_rank_points e loser_rank_points (utilizzati in Sezione [3]), utilizzando la media degli altri valori del player per quell'attributo, o la media globale (di winners o di losers) nel caso in cui quel giocatore avesse solo valori *nan*.

1.1 Assignment 0: Definizione dello schema del database

Il primo passo del processo di BI è stato quello di costruire lo schema del database, visibile in Figura [1], utilizzando il software **SQL Server Management Studio**, nominandolo **Group06HWMart**.
Per la definizione dei tipi dei diversi attributi sono state effettuate le seguenti scelte:
- per gli attributi di tipo stringa con un numero variabile di caratteri è stato utilizzato il tipo **nvarchar**, preferito rispetto a **varchar** poichè più generale, in quanto permette di riconoscere tutti i caratteri UNICODE;
- per gli attributi di tipo stringa con un numero fisso di caratteri è stato scelto il tipo **nchar** (ad esempio, nel caso dell'attributo **sex** è stato definito *nchar(1)*);
- l'attributo *date_id* è stato definito come intero, poichè svolge la funzione di chiave primaria per la dimensione **Date**, nonostante sia una data nel formato YYYYMMDD.
Per le dimensioni **Player** e **Tournament** sono stati definiti nuovi numeri di identificazione (in fase di split definita in Sezione [1.2]), rispettivamente **player_id** e **tourney_id**, che fungono da chiavi primarie delle tabelle, mantenendo comunque i vecchi id nell'attributo *old_{table_name}_id*. Questa soluzione è stata preferita, per questioni di praticità, alla definizioni di superchiavi più complesse: nel caso della tabella **Tournament** composta dalla concatenazione di *tourney_id*, *tourney_name*, *tourney_level*; nel caso di **Player** composta dalla concatenazione di *player_id*, *player_name*. Invece, per le altre tabelle presenti nello schema è stato possibile selezionare un attributo già presente come chiave primaria.

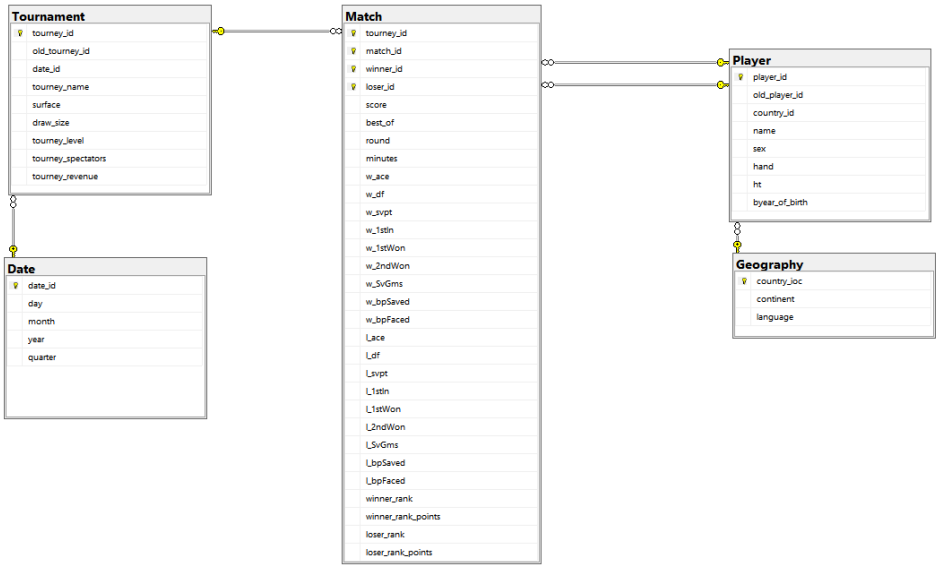

Figura 1: Database schema

1.2 Assignment 1: Generazione tabelle

In questa fase, la tabella **tennis.csv** è stata suddivisa in 4 tabelle distinte (*match*, *tournament*, *player_tmp* e *date*) nel file **split_fact.py**. La tabella *player_tmp* è stata successivamente rielaborata nella tabella definitiva *player* all'interno del file **prepare_player.py** per poter creare l'attributo **byear_of_birth**. Infine, è stata creata la tabella **geography** all'interno del file **create_geography.py**. I dati prodotti sono stati inseriti nella cartella *project_data*.

1.2.1 Split_fact

La divisione dei dati nelle tabelle è stata effettuata scorrendo con un ciclo for i record del file originale, seguendo il seguente schema di operazioni:
1) Scrittura, tramite la funzione **prepare_write_tournament**, del record relativo al torneo, aggiungendo il nuovo id progressivo per identificare univocamente la tabella finale **Tournament**.
2) Uso della funzione **store_players** per salvare i players (winner e loser) nel dizionario **players_dict** in modo tale da tenere traccia dei valori degli attributi del player (i quali non sono sempre definiti) e salvando un nuovo id progressivo, utile (come menzionato in Sezione [1.1]) per identificare univocamente i giocatori.
3) Scrittura del record relativo al match nella funzione **prepare_write_match**, nella quale vengono gestite le incongruenze nei dati.
4) Uso della funzione **prepare_write_date**, per scrivere i dati relativi alla data, inserendo come valore del trimestre uno tra {Q1, Q2, Q3, Q4} in base al valore del mese, secondo le seguenti associazioni: $[01, 03] \rightarrow Q1$, $[04, 06] \rightarrow Q2$, $[07, 09] \rightarrow Q3$, $[10, 12] \rightarrow Q4$.
Nell'ultima parte del file, i dati dei players raccolti vengono scritti all'interno del file csv *player_tmp.csv*, attraverso la funzione **write_player**.

1.2.2 Prepare_player

Nel file **prepare_player.py** è stata creata la tabella **player.csv**, partendo dai dati ottenuti in Sezione [1.2.1] e creando gli attributi **sex** e **byear_of_birth**. L'anno di nascita del giocatore è stato ottenuto attraverso la funzione **get_byear**, nella quale si effettuano operazioni diverse se il valore di **age** per il giocatore è definito o meno. Nel primo caso, per riuscire a identificare l'anno di nascita del giocatore nel modo più preciso possibile sono stati eseguiti i seguenti step:
1) è stato ottenuto il delta tra i valori di età in due giorni consecutivi (06 Febbraio 2019 e 07 Febbraio 2019), come:

$$delta = age\_07F - age\_06F = 23.1238877481 - 23.1211498973 = 0.0027378507999991086 \quad (1)$$

un valore numerico rappresentativo della distanza tra due giorni;
2) è stata considerata la distanza in termini di giorni con il prossimo compleanno del giocatore, come:

$$days\_dist = round\left(\frac{\lceil age \rceil - age}{delta}\right) \quad (2)$$

3) sono stati sommati i giorni di distanza dal prossimo compleanno **days_dist** alla data del torneo a cui ha partecipato il giocatore, attraverso la libreria **datetime**; ciò è stato pensato in modo da poter scegliere se per ottenere l'anno di nascita si sarebbe dovuto sottrarre all'anno del torneo l'età arrotondata o meno per eccesso. Infatti, nel caso in cui l'anno del prossimo compleanno del giocatore fosse coinciso con l'anno del torneo, sarebbe stato necessario (per maggior precisione) arrotondare per eccesso.

Per poter ottenere i dati mancanti di **byear_of_birth** e degli attributi **ht**, **hand** e **sex**[1], è stato effettuato uno scraping dei dati dal sito `www.tennisexplorer.com` all'interno della funzione **scrapy_player_data**. Tramite questa metodologia è stato creato il file *scraped_player.csv* (file definitivo rappresentante la tabella *Player*) effettuando, in prima istanza, una ricerca del nome del player i-esimo all'interno della lista dei giocatori definita dal sito[2] (tramite libreria **requests**), in modo tale da ottenere l'url corrispondente alla pagina contenente i dettagli dello specifico player[3] (scaricati attraverso l'uso della libreria **BeautifulSoup**).

1.2.3 Create_geography

La tabella **Geography** è stata ottenuta attraverso i procedimenti presenti nel file **create_geography.py** il cui flusso di eventi viene schematizzato in Figura [2]. Poichè non tutti i *country_id* della tabella **Player** sono risultati essere presenti nel file **countries.cvs** (fornito nella consegna), per potere ottenere le informazioni relative ai continenti e alle lingue corrispondenti, sono stati uniti quei dati ad informazioni provenienti da fonti esterne.
Tramite la funzione *get_languages* (4) sono stati creati due dizionari:
- **name_id**, il quale associa ad ogni *country_name* il corrispondente *country_id*;
- **id_language**, il quale associa ad ogni *country_id* una lista contenente la lingua e il continente corrispondenti.

I dizionari sono stati costruiti in due fasi:
- Una prima fase considerando le informazioni contenute nel file *geonames.txt* ottenuto in (1)[4].
- Una seconda fase nella quale è stata corretta una parte dei dati, corrispondente ad IDs in un formato diverso da quello presente in **countries.csv**.

La tabella finale è stata ottenuta dalla funzione **write_geography** in (5), nella quale per ogni **country_id** presente nella tabella **Player** (ottenuto in (3) dalla lista *players_c_codes*), viene scritta una riga in *geography.csv* cercando i dati nei dizionari di input, gestendo esplicitamente eventuali errori nei dati. In particolare, il dizionario **name_id** è stato utilizzato come ponte per poter associare le chiavi del dizionario **countries** (ottenuto in (2)[5]) ai valori di lingua e continente presenti nel dizionario **id_language**, in modo tale da completare i dati mancanti. Sono stati utilizzati 3 dizionari poichè è risultato necessario tenere traccia di *id* sia in formato **ISO3166-1 alpha3** che in formato **IOC** presenti nei dati.

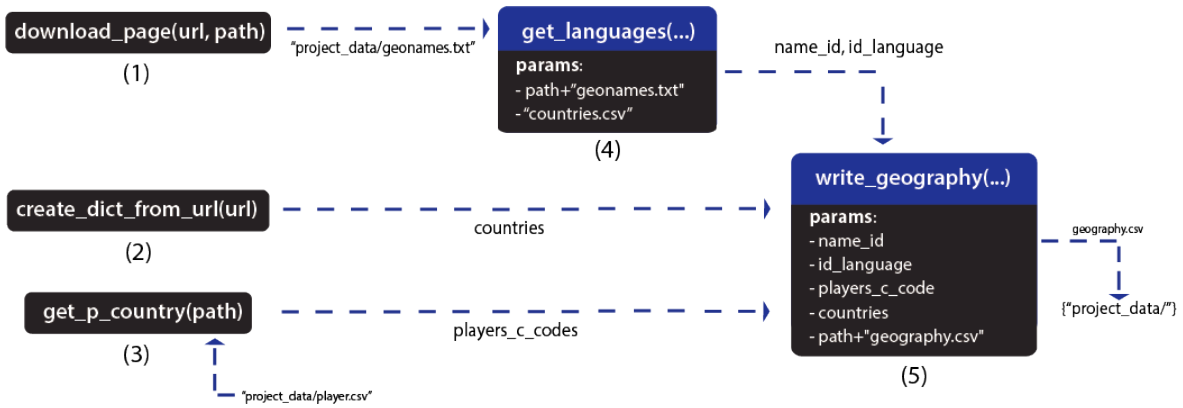

Figura 2: Creazione geography.csv

[1] All'interno del codice sono state mantenute, per completezza, le funzioni *get_gender_set* e *get_gender_dict* risultate superflue, grazie allo scraping, per l'ottenimento dell'attributo *sex*
[2] `https://www.tennisexplorer.com/list-players/?search-text-pl={p_name2}&country="`
[3] `https://www.tennisexplorer.com/player/{to_search}/`
[4] I dati sono stati scaricati dall'url `http://download.geonames.org/export/dump/countryInfo.txt`
[5] I dati sono stati ottenuti dall'url `https://www.worlddata.info/countrycodes.php`

1.3 Assignment 2: Inserimento dei dati nello schema

Per popolare il database è stato scritto il programma **upload_data.py**, con la seguente struttura:
1) Apertura della connessione al database *Group_6_DB* tramite la libreria **pyodbc** e creazione del cursore.
2) Lettura dei file *.csv*, utilizzando il metodo **csv.reader**.
3) Creazione di un dizionario contenente le associazioni <nome_tabella>→<reader_tabella>.
4) Per ogni elemento del dizionario è stata generata una query **INSERT INTO**, dinamicamente in base al numero di elementi presenti nell'header del file corrispondente, eseguita tramite **cursor.execute()**.
5) Chiusura dei file, cursore e connessione.

2 Analisi dati tramite SSIS

Le analisi SSIS delle business questions assegnate sono state inserite nel progetto *Part2_SSIS*: **Assignment_0.dtsx**, **Assignment_1.dtsx** e **Assignment_2.dtsx**. In ogni file sono state inserite delle annotazioni contenenti una descrizione completa della soluzione implementata.

3 Analisi dati tramite SSAS

3.1 Costruzione Datacube

In questa fase sono stati costruiti due cubi OLAP (i cui progetti SSAS sono stati chiamati **group6_cube** e **Group06_LDS_Part3**), equivalenti nei risultati delle query svolte in Sezione [3.2] e nelle dashboards sviluppate in Sezione [3.3], ma differenti in termini di numero di dimensioni:

1. Nel cubo del progetto **group6_cube** le dimensioni *Time* e *Geography* sono state inserite esplicitamente, definendo quindi i riferimenti con le chiavi esterne delle dimensioni Tournament e Winner/Loser (avendo quindi GeographyWinner e GeographyLoser).

2. Nel cubo del progetto **Group06_LDS_Part3** gli attributi delle dimensioni Time e Geography sono stati inseriti direttamente nelle dimensioni Tournament e Winner/Loser.

Entrambe le soluzioni presentano due gerarchie *non flat* costruite al fine di gestire sia le dimensioni temporali che spaziali dei dati, ovvero **DayMonthQuarterYear** e **Gerarchia**, come visibile in Figura [3].

(a) DayMonthQuarterYear (b) Gerarchia

Figura 3: Gerarchie

Nel costruire la gerarchia **DayMonthQuarterYear** è stato tenuto conto del fatto che la dipendenza funzionale nei dati non è presente tra tutti i livelli: un trimestre o un mese possono appartenere a più anni. Per motivi di chiarezza, è stato creato tramite *calcolo denominato* la colonna **Month Str** (inserita all'interno della gerarchia) contenente il nome dei mesi invece del singolo numero. Per **Gerarchia**, invece, ad ogni valore dell'attributo del livello inferiore (*Country*) corrisponde un solo valore del livello superiore (*Continent*). Sono state definite, come richiesto, due misure per ogni tipologia di player (winner

e loser) basate sui campi **rank** e **rank_points**. In aggiunta a quelle richieste, sono state definite altre tre misure, la prima rappresentante il numero di winner distinti (utilizzata per poter risolvere le queries mdx), mentre le altre ottenute da **tourney_revenue** e **tourney_spectators** (utilizzate nella dashboard in Sezione [3.3.2]).

3.2 Query MDX

Le soluzioni alle query mdx richieste sono state implementate nei files **Assignment1-3_cube1.mdx** e **Assignment1-3_cube2.mdx**, il primo basato sul cubo creato in **group6_cube** (di cui al punto (1)), il secondo basato sul cubo creato in **Group06_LDS_Part3** (di cui al punto (2)). In ogni file è stata inserita una descrizione della soluzione implementata, proponendo per l'assegnamento 2 due differenti soluzioni per rendere la query più generica.

3.3 Dashboards

Le dashboards, realizzate in **PowerBI**, sono presenti all'interno dei files **Assignment4&5_cube1.pbix** (basate sul cubo (1)) e **Assignment4&5_cube2.pbix** (basate sul cubo (2)). Le soluzioni risultano, come visto per le query mdx, equivalenti.

3.3.1 Assignment 4 - Geographical distribution

Per mostrare la distribuzione geografica dei **rank points** sono state usate due mappe, una per Winner e una per Loser, ognuna delle quali legata ad uno specifico grafico ad anello, che mostra la distribuzione delle misure a diversa granularità geografica (tramite *drilldown*). La variazione di dimensione delle bolle sulle mappe dipendono dal valore totale dell'attributo esplorato (**rank points**) per quel paese, partendo da una dimensione minima di *-10*. Attraverso la dashboard è possibile vedere come, considerando come granularità il continente, si ha lo stesso quantitativo di *rank points* per winner e losers. Tuttavia, esplorando attraverso drill down, emergono alcune differenze per paese: considerando il continente africano, per loser rank points sono presenti (a differenza di winner) i paesi: Benin (BEN), Ghana (GHA) ed Eritrea (ERI).

3.3.2 Assignment 5 - Tournament Analysis

L'obiettivo di questa implementazione è quello di fornire un'analisi dei tornei, non considerati nelle analisi precedenti. La dashboard mostra, attraverso un grafico ad area, il tourney revenue totale per dimensione del tabellone, selezionabile attraverso il filtro **DrawSize** posto nella parte superiore della dashboard. Nella parte inferiore, invece, viene riportato un istogramma attraverso il quale è possibile osservare il Tourney Revenue per i valori di superficie a diversa granularità temporale (esplorabile tramite *drilldown*). L'istogramma è collegato alla matrice, la quale mostra esplicitamente i valori di *Tourney Revenue* e *Tourney Spectators*. Infine, il filtro per **Tourney Level** permette all'utente di affinare la dashboard per i livelli di interesse. La dashboard ha permesso di osservare come i valori maggiori di revenue si hanno per tornei con un draw size pari a 32 per qualsiasi anno, con una riduzione consistente nel 2020 dovuta al blocco dei tornei nel periodo Aprile-Luglio, a causa della situazione pandemica.