1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

Corso di
Basi di Dati

Anno accademico
2020-2021

Progetto realizzato da
Barzotti Cristian

Matricola
290725

Corso tenuto dal professore
Ferretti Stefano

Progettazione e Implementazione di un Database utilizzando MySQL e Python3

Indice

Capitolo 1

Specifiche del problema

Si richiede l'implementazione di un database con interfaccia grafica contenente i seguenti dati.
Si devono catalogare dei campioni, che vengono identificati tramite un nominativo. Il nominativo non è univoco, e campioni provenienti da opere diverse potrebbero avere lo stesso nome. Ogni campione proviene da una singola opera, ma più campioni diversi possono provenire dalla stessa opera.
L'opera ha un autore, un periodo di creazione e una tipologia. Campione ed opera condividono la provenienza.
Del campione si vuole anche immagazzinare la data di prelievo, una descrizione, lo stato.
Si vuole includere in quale laboratorio di analisi è stato inviato, se è stato inviato.
Si vuole registrare data e luogo di catalogazione.
I campioni possono essere citati in articoli o riferimenti; di questi articoli o riferimenti si vogliono ricordare (se presenti) il nome, la data di pubblicazione, il nome del pubblicatore e un collegamento per trovarli online.
Articoli o riferimenti hanno uno o più relatori, dei quali si vuole ricordare il nome e l'afferenza, se presente.
Opera, autore e laboratorio non sono univoci e il loro nome potrebbe essere condiviso tra diversi record[1].
Si richiede solo l'inserimento, la rimozione e la visualizzazione di record.

[1] Due opere potrebbero avere lo stesso nome ma creatori diversi

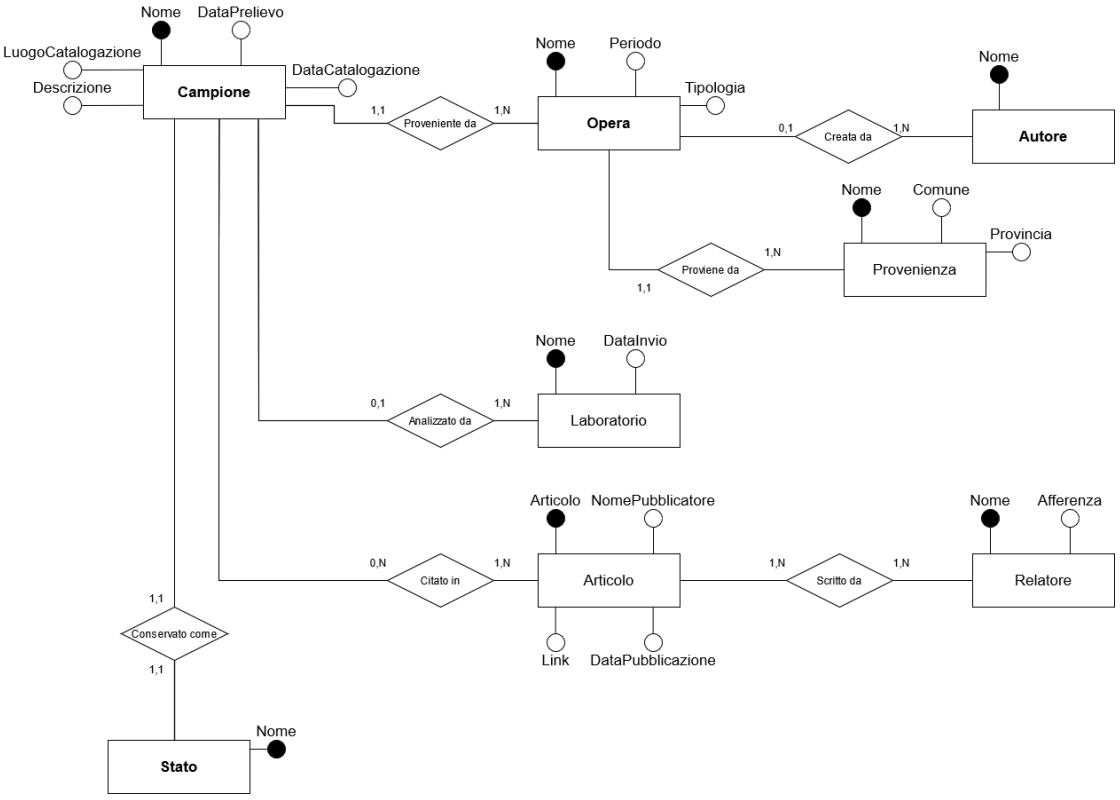
Parte I

Progettazione concettuale

Capitolo 2

Schema E-R

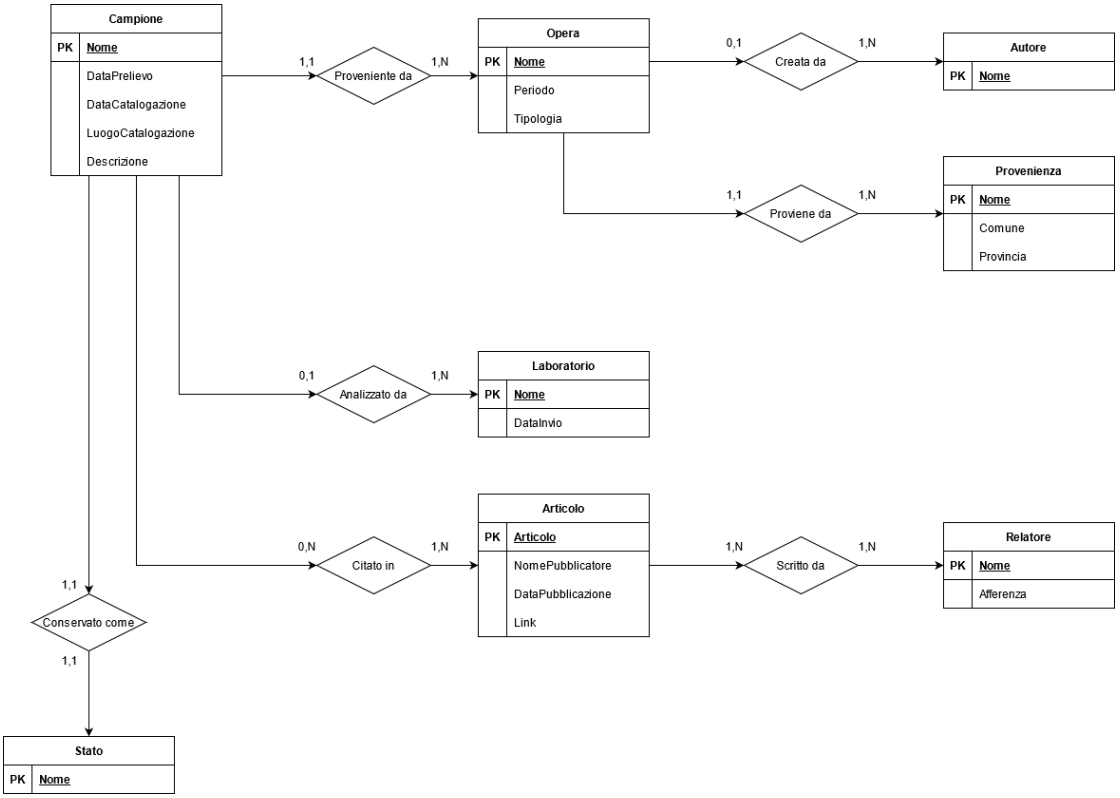
Dopo un iniziale studio del problema, si è passati alla creazione dello schema E-R del database da creare. Di seguito verrà riportato tale schema.



La suddivisione delle tabelle e dei vari attributi è stata pensata per avere maggiore semplicità di future espansioni del database stesso; basti pensare che una tabella come "Autore", contenente solo un campo Nome, potrebbe sembrare superflua e potrebbe essere integrata direttamente nella tabella "Opera".

Si è deciso di evitare questo tipo di approccio, e di strutturare al meglio la divisione delle tabelle. Se, in futuro, si volessero aggiungere ulteriori valori ad "Autore", come ad esempio il cognome, data di nascita e morte o provenienza, avere una tabella apposita già pronta semplificherà enormemente l'operazione.
Questo tipo di approccio è stato implementato, al meglio delle mie capacità, per la suddivisione di tutte le tabelle.

Per poter rendere , a mio avviso, più leggibile lo schema E-R si è deciso di utilizzare la seguente notazione. Ci tengo a precisare che, a livello contenutistico, è assolutamente uguale allo schema precedente.



Capitolo 3

Traduzione dello schema E-R

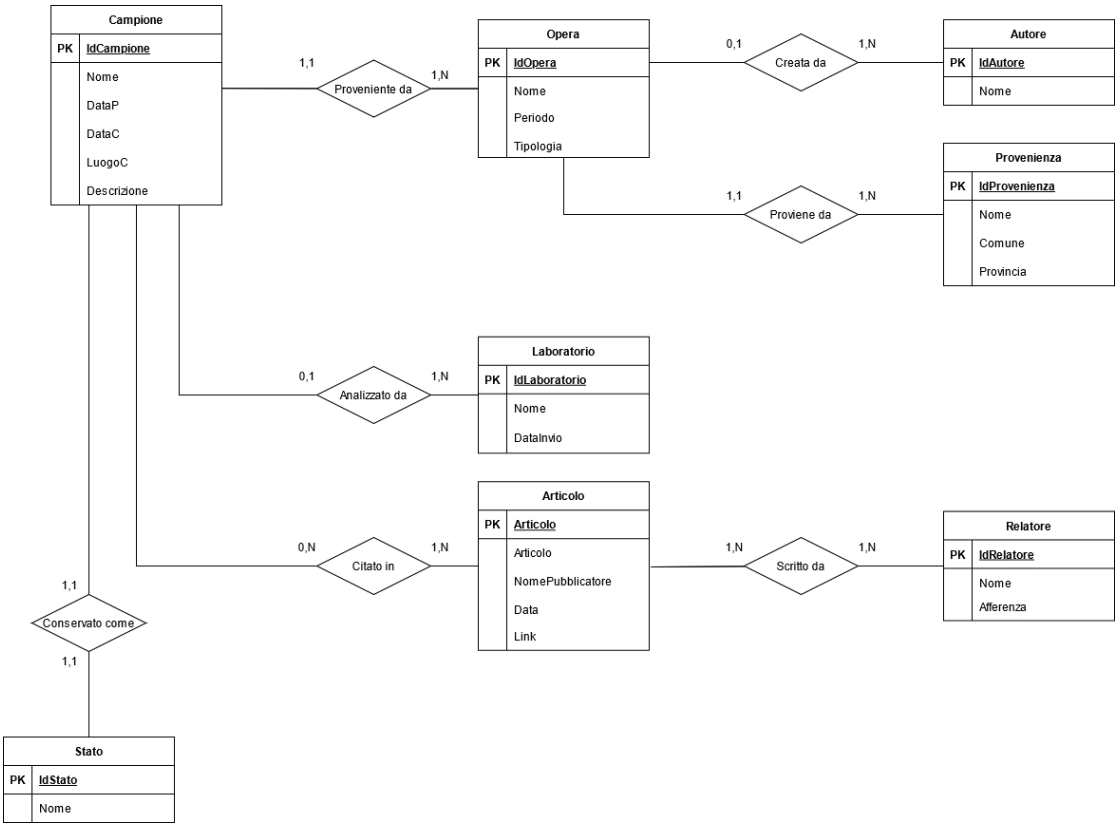
Partendo così dal modello E-R è stato possibile tradurlo nello schema relazionale del database, come mostrato in seguito.

Schema relazionale	
Campione (Nome, DataPrelievo, DataCatalogazione, LuogoCatalogazione, Descrizione)	
Stato (Nome)	
Opera (Nome, Data, Tipologia)	
Autore (Nome)	
Provenienza (Nome, Comune, Provincia)	
Laboratorio (Nome, DataInvio)	
Articolo (Articolo, NomePubblicatore, DataPubblicazione, Link)	
Relatore (Nome, Afferenza)	
Citato (NomeCampione, Articolo)	NomeCampione REFERENCES Campione, Articolo REFERENCES Articolo
Scritto (Articolo, NomeRelatore)	Articolo REFERENCES Articolo, NomeRelatore REFERENCES Relatore

Capitolo 4

Ottimizzazioni

Per poter ottimizzare inserimento e immagazzinamento delle informazioni, si è deciso di introdurre un ulteriore campo all'interno di tutte le tabelle; questo campo, chiamato "Id*Nometabella*" (i.e. "IdCampione", "IdAutore", etc), fungerà da chiave primaria per la tabella stessa. Di seguito riportiamo lo schema E-R con questo ulteriore campo all'interno delle tabelle.



Di conseguenza, anche lo schema relazionale deve essere aggiornato come segue.

Schema relazionale	
Campione (IdCampione, Nome, DataP, DataC, LuogoC, Descrizione)	
Stato (IdStato, Nome)	
Opera (IdOpera, Nome, Periodo, Tipologia)	
Autore (IdAutore, Nome)	
Provenienza (IdProvenienza, Nome, Comune, Provincia)	
Laboratorio (IdLaboratorio, Nome, DataInvio)	
Articolo (IdArticolo, Articolo, NomePub, Data, Link)	
Relatore (IdRelatore, Nome, Afferenza)	
Citato (IdCampione, IdArticolo)	IdCampione REFERENCES Campione, IdArticolo REFERENCES Articolo
Scritto (IdArticolo, IdRelatore)	IdArticolo REFERENCES Articolo, IdRelatore REFERENCES Relatore

Oltre all'aggiunta di un campo nelle varie tabelle, alcuni dei campi hanno subito un *refactor*, una piccola modifica al nome (i.e. DataPrelievo →DataP, DataCatalogazione →DataC, etc).

Queste ottimizzazioni non vanno in nessun modo ad alterare o perdere informazioni e sono state effettuate solo per avere maggiore pulizia e ordine nel database stesso.

Capitolo 5

Normalizzazione

La normalizzazione è un procedimento volto all'eliminazione della ridondanza delle informazioni e del rischio di incoerenze nel database.
Esistono vari livelli di normalizzazioni, chiamati **Forme Normali**, che certificano la qualità dello schema relazionale del database.
Le forme normali da verificare sono:

- **I Forma Normale**:
  I domini degli attributi devono avere valori atomici;
- **II Forma Normale**:
  Per ogni relazione tutti gli attributi non chiave dipendono funzionalmente dall'intera chiave composta;
- **III Forma Normale**:
  Tutti gli attributi non chiave dipendono solo dalla chiave.
- **Forma Normale di Boyce e Codd**:
  Per ogni dipendenza funzionale non banale X Y(con Y non in X), X è una superchiave della relazione.

Di seguito, andremo ad esaminare le nostre entità.

5.1 Analisi delle entità

Campione

Campione (IdCampione, Nome, DataP, DataC, LuogoC, Descrizione)

- **I FN**: soddisfatta. Gli attributi hanno valori atomici.
- **II FN**: soddisfatta. Vi è un unica chiave.
- **III FN**: soddisfatta. Gli attributi non chiave non dipendono da attributi non chiave.
- **Dipendenze Funzionali**: IdCampione→Nome, DataP, DataC, LuogoC, Descrizione
- **BC FN**: soddisfatta. Il primo termine delle dipendenze funzionali contiene una chiave.

Stato

Stato (IdStato, Nome)

- **I FN**: soddisfatta. Gli attributi hanno valori atomici.
- **II FN**: soddisfatta. Vi è un unica chiave.
- **III FN**: soddisfatta. Gli attributi non chiave non dipendono da attributi non chiave.
- **Dipendenze Funzionali**: IdStato→Nome
- **BC FN**: soddisfatta. Il primo termine delle dipendenze funzionali contiene una chiave.

Opera

Opera (IdOpera, Nome, Periodo, Tipologia)

- **I FN**: soddisfatta. Gli attributi hanno valori atomici.
- **II FN**: soddisfatta. Vi è un unica chiave.
- **III FN**: soddisfatta. Gli attributi non chiave non dipendono da attributi non chiave.
- **Dipendenze Funzionali**: IdOpera→Nome, Periodo, Tipologia
- **BC FN**: soddisfatta. Il primo termine delle dipendenze funzionali contiene una chiave.

Autore

Autore (IdAutore, Nome)

- **I FN**: soddisfatta. Gli attributi hanno valori atomici.
- **II FN**: soddisfatta. Vi è un unica chiave.
- **III FN**: soddisfatta. Gli attributi non chiave non dipendono da attributi non chiave.
- **Dipendenze Funzionali**: IdAutore→Nome
- **BC FN**: soddisfatta. Il primo termine delle dipendenze funzionali contiene una chiave.

Provenienza

Provenienza (IdProvenienza, Nome, Comune, Provincia)

- **I FN**: soddisfatta. Gli attributi hanno valori atomici.
- **II FN**: soddisfatta. Vi è un unica chiave.
- **III FN**: soddisfatta. Gli attributi non chiave non dipendono da attributi non chiave.
- **Dipendenze Funzionali**: IdProvenienza→Nome, Comune, Provincia
- **BC FN**: soddisfatta. Il primo termine delle dipendenze funzionali contiene una chiave.

Laboratorio

Laboratorio (IdLaboratorio, Nome, DataInvio)

- **I FN**: soddisfatta. Gli attributi hanno valori atomici.
- **II FN**: soddisfatta. Vi è un unica chiave.
- **III FN**: soddisfatta. Gli attributi non chiave non dipendono da attributi non chiave.
- **Dipendenze Funzionali**: IdLaboratorio→Nome, DataInvio
- **BC FN**: soddisfatta. Il primo termine delle dipendenze funzionali contiene una chiave.

Articolo

Articolo (IdArticolo, Articolo, NomePub, Data, Link)

- **I FN**: soddisfatta. Gli attributi hanno valori atomici.
- **II FN**: soddisfatta. Vi è un unica chiave.
- **III FN**: soddisfatta. Gli attributi non chiave non dipendono da attributi non chiave.
- **Dipendenze Funzionali**: IdArticolo→Articolo, NomePub, Data, Link
- **BC FN**: soddisfatta. Il primo termine delle dipendenze funzionali contiene una chiave.

Relatore

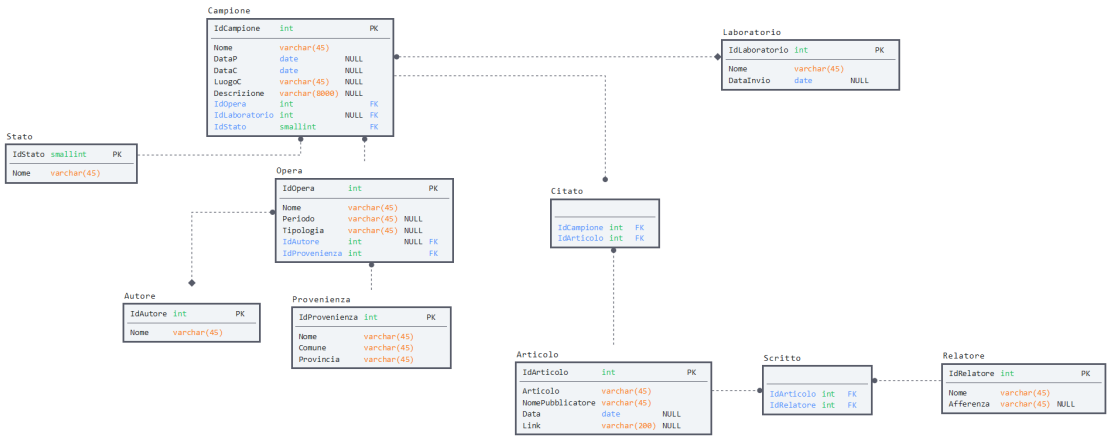
Relatore (IdRelatore, Nome, Afferenza)

- **I FN**: soddisfatta. Gli attributi hanno valori atomici.
- **II FN**: soddisfatta. Vi è un unica chiave.
- **III FN**: soddisfatta. Gli attributi non chiave non dipendono da attributi non chiave.
- **Dipendenze Funzionali**: IdRelatore→Nome, Afferenza
- **BC FN**: soddisfatta. Il primo termine delle dipendenze funzionali contiene una chiave.

Capitolo 6

Schema SQL

Dopo aver studiato, sviluppato ed ottimizzato al meglio lo schema E-R, si è passatti alla progettazione dello schema SQL, utilizzando **SQLdbm**[1], riportato in seguito.



[1] `https://sqldbm.com/Home/`

Capitolo 7

Scelte Architetturali

Come già anticipato dal titolo di questa relazione, per l'implementazione del database si è optato per utilizzare MySQL, lato DB, e Python3 per creare una piccola applicazione che consente la visione, inserimento e rimozione degli elementi.

7.1 MySQL

La prima domanda da porsi durante un progetto del genere riguarda proprio la scelta del gestionale del DB.
Prima di tutto si deve scegliere se adoperare un DBMS relazionale, come MySQL, SQL Server, MariaDB etc oppure optare per altre opzioni. Le due principali opzioni a mia disposizione erano MySQL e MongoDB, entrambi ottimi e validi DBMS per il progetto in questione. Alla fine si è deciso per MySQL, gestionale solido e sul mercato da tantissimi anni[1], open-source e con numerosi OS[2] e linguaggi di programmazione supportati, utilizzando InnoDB come engine.
Questo mi ha consentito di utilizzare le informazioni e nozioni apprese durante il corso ed è stato sicuramente un punto a favore di MySQL rispetto al secondo candidato più papabile, MongoDB.

7.2 Python3

La scelta di utilizzare Python3 è stata arbitraria; dopo averlo visto a lezione, ed avendo sviluppato un progetto per un altro esame proprio con Python3, ero molto propenso ad utilizzarlo nuovamente per cercare di migliorare la mia conoscenza del linguaggio stesso.
In questo progetto Python3 è utilizzato per avere un *backend* e *frontend* con cui interagire con il DB; si è deciso di racchiudere il tutto all'interno di tre file Python con compiti diversi:

- Database.py: il "main" del mio programma, si occupa della creazione e mantenimento della finestra principale.
- connection.py: contiene la classe **MyConnector**, il "backend" del progetto. Qui sono contenute le query che utilizzo per dialogare con il DB.
- utils.py: come suggerisce il nome, contiene funzioni che sono utili ma non indispensabili; si occupa di creare popup per la visualizzazione e l'inserimento dei dati, per poi utilizzare i metodi della classe MyConnector() per le query al DB.

[1] La prima versione di MySQL è uscita nel 1994
[2] Le principali piattaforme di riferimento sono Linux e Solaris, ma è perfettamente integrato anche per Windows

Capitolo 8

Primo Avvio

Assieme a questa relazione verranno inviati i file relativi al database e all'applicazione.

Per il database è previsto un dump contenente struttura e alcuni dati di prova, mentre per l'applicazione è previsto l'invio del codice sorgente[1].

Per poter avviare il database è necessario installare MySQL Community Server 8.0.25[2]. Si consiglia anche di installare MySQL Workbench, per poter importare in maniera semplice il dump.

Per connettersi al DB è necessario inserire i dati di accesso all'interno del programma Python, che non ha ancora un sistema di login. Per farlo, aprire il file "connection.py" e modificare la seguente linea con i dati opportuni.

```
self.connection = mysql.connector.connect(host='localhost', database='progettobdd',
                                          user='root', password='@774c3n2')
```

Per poter avviare il programma Python è necessario avere installati Python3[3] e il modulo Connector/Python 8.0.25[4]. Una volta estratti i contenuti, aprire il terminale nella directory dove sono presenti i contenuti e lanciare Database.py utilizzando Python3.

Ci tengo a precisare che il progetto è stato sviluppato interamente in ambiente Windows e, per mancanza di tempo, non è stato testato in ambiente Linux. Ritengo, però, che sia DB che programma dovrebbero essere funzionanti in entrambi, previa installazione dei giusti componenti nello specifico ambiente.

[1] Creando file eseguibili Python alcuni antivirus lo leggono come trojan o malware e lo bloccano, per cui si è optato per il file sorgente

[2] `https://dev.mysql.com/downloads/mysql/`

[3] `https://www.python.org/downloads/`

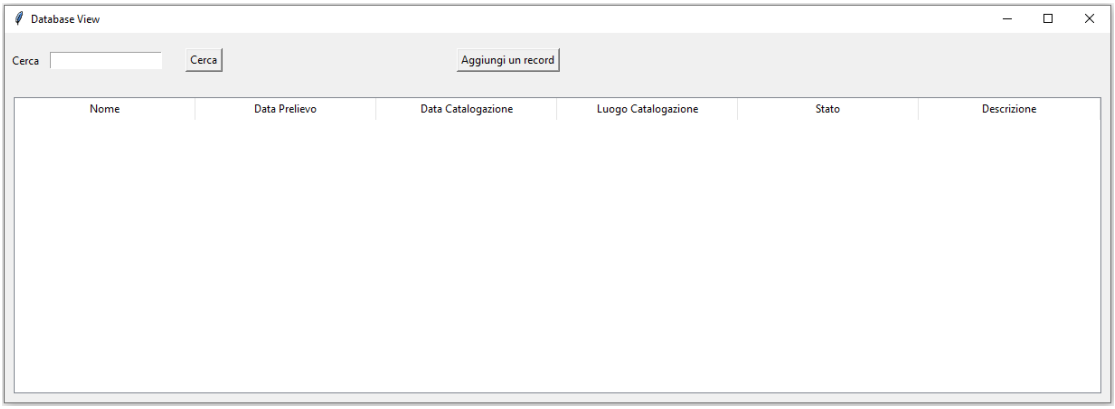
[4] `https://dev.mysql.com/downloads/connector/python/`

Capitolo 9

GUI e walkthrough

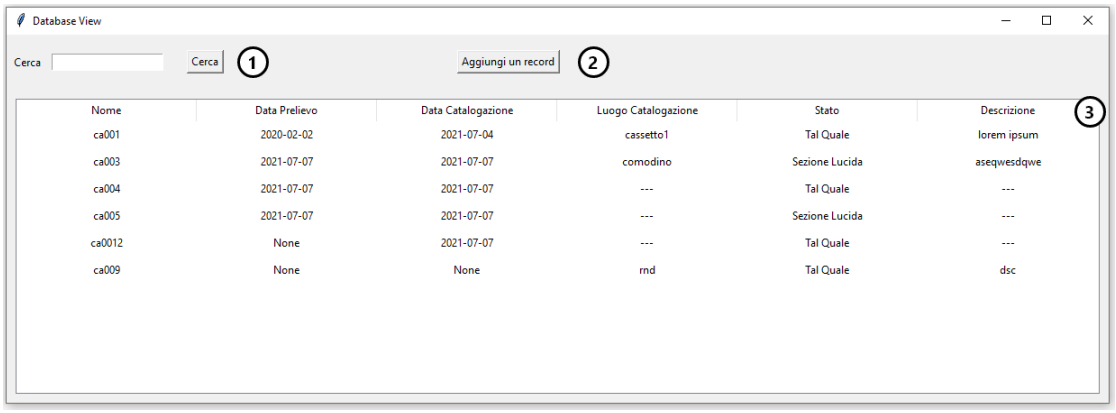
In questa sezione verrà spiegata la GUI[1] e come utilizzarla.

Finestra principale



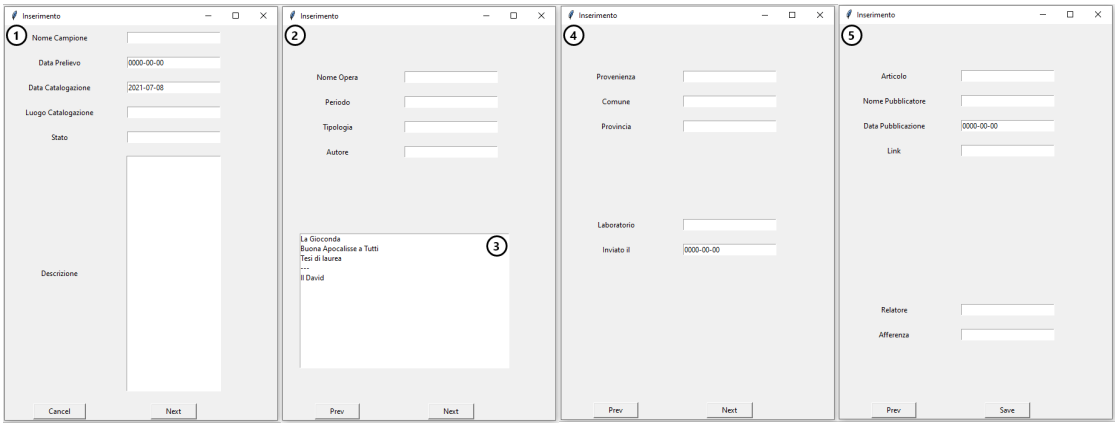
Dopo l'avvio del programma, l'utente si troverà di fronte la finestra principale. Da qui è possibile visualizzare un sommario delle informazioni contenute nel DB[2] effettuanto una ricerca e aggiungere un record al DB.

[1] Graphical User Interface
[2] Queste sono le informazioni più importanti alle quali è necessario accedere "velocemente"



La finestra contiene 3 aree principali:

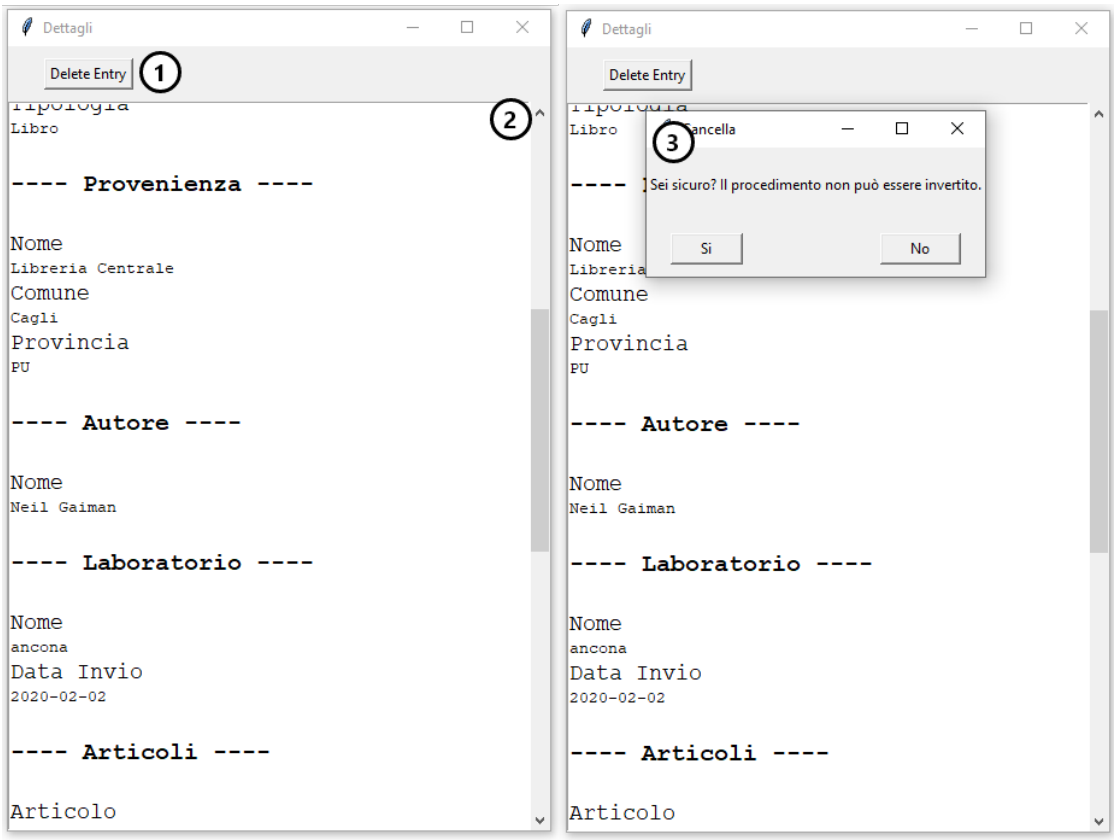
1. Casella di ricerca con relativo pulsante[3];
2. Pulsante di aggiunta di nuovi record;
3. Tabella di visualizzazione dei record.[4][5]



Popup di inserimento per nuovi record. È composto da:

1. Finestra di inserimento dei dati relativi al campione;
2. Finestra di inserimento dei dati relativi all'opera;
3. Menù a scelta contenente le opere già presenti nel DB;[6]
4. Finestra di inserimento dei dati relativi a provenienza e laboratorio;
5. Finestra di inserimento dei dati relativi ad articoli scritti e relatori.

[3]Lasciando il campo vuoto verranno riportati tutti i campioni
[4]Nell'esempio, dei record utilizzati durante lo sviluppo per il debug. Potrebbero variare
[5]Facendo "Doppio Click" su una entry della tabella si aprirà la finestra di visualizzazione dei dati completi
[6]Scegliendo una entry, verranno popolati i campi relativi a opera, autore e provenienza



Popup di visualizzazione dei dettagli di un campione.

1. Pulsante di eliminazione del record;
2. Tabella di visualizzazione di tutti i dati relativi al campione;
3. Popup di conferma per l'eliminazione di un record.

Capitolo 10

Bug conosciuti

Il programma è ancora in fase di sviluppo e non ancora perfetto; sono infatti presenti dei bug e delle features mancanti, che verranno elencate in seguito.

Bug

Il bug sicuramente più fastidioso e subito notabile è un errore durante l'inserimento dovuto all'inesperienza nell'interfacciare Python con MySQL. Durante l'inserimento dei dati, tutti i campi sono necessari per evitare errori con le query di inserimento, pena l'impossibilità di inserimento del record. Per ovviare a questo problema, almeno in parte, sono stati utilizzati i seguenti metodi:

1. Alcuni dei campi più suscettibili sono stati precompilati, sia per evitare che vengano lasciati vuoti sia per aiutare l'utente a capire come vanno inseriti determinati dati[1];
2. Il popup di inserimento, durante la sua compilazione, crea una lista di elementi che poi vengono selezionati singolarmente durante lésecuzione delle query. In caso in cui ci siano dei campi vuoti, questi verranno riempiti automaticamente con dei simboli[2] che rappresentano un "record standard" all'interno del mio DB. Questo evita errori la maggior parte delle volte.

Features Mancanti

Alcune features sono ancora mancanti o incomplete all'interno del programma; in particolare, al momento è molto complicato associare diversi campioni ad un unico articolo (e, al contrario, diversi articoli ad un unico campione).[3][4]

Inoltre, al momento non è ancora implementato un metodo per modificare i dati di un campione dopo averlo inserito all'interno del DB[5].

[1] Riguarda principalmente le date
[2] In particolare, —
[3] Tabella di riferimento "Citato"
[4] Si applica anche ad Articoli e Relatori (tab. rif. "Scritto")
[5] Non era stato richiesto dal committente del lavoro

Capitolo 11

Query utilizzate

In questa sezione andremo a parlare delle query utilizzate, andando a spiegare perchè sono state utilizzate e come.

11.1 Tabelle

Campione

```
CREATE TABLE campione‘ (
  ‘idCampione‘ int NOT NULL AUTO_INCREMENT,
  ‘Nome‘ varchar(45) NOT NULL,
  ‘DataP‘ date DEFAULT NULL,
  ‘DataC‘ date DEFAULT NULL,
  ‘LuogoC‘ varchar(45) DEFAULT NULL,
  ‘Descrizione‘ varchar(8000) DEFAULT NULL,
  ‘IdStato‘ smallint NOT NULL,
  ‘IdOpera‘ int NOT NULL,
  ‘IdLaboratorio‘ int DEFAULT NULL,
  PRIMARY KEY (‘idCampione‘),
  KEY ‘IdStato_idx‘ (‘IdStato‘),
  KEY ‘IdOpera_idx‘ (‘IdOpera‘),
  KEY ‘IdLaboratorio_idx‘ (‘IdLaboratorio‘),
  CONSTRAINT ‘IdLaboratorio‘ FOREIGN KEY (‘IdLaboratorio‘) REFERENCES ‘laboratorio‘ (‘IdLaboratorio‘),
  CONSTRAINT ‘IdOpera‘ FOREIGN KEY (‘IdOpera‘) REFERENCES ‘opera‘ (‘IdOpera‘),
  CONSTRAINT ‘IdStato‘ FOREIGN KEY (‘IdStato‘) REFERENCES ‘stato‘ (‘idStato‘)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Stato

```
CREATE TABLE ‘stato‘ (
  ‘idStato‘ smallint NOT NULL,
  ‘Nome‘ varchar(45) NOT NULL,
  PRIMARY KEY (‘idStato‘)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Opera

```
CREATE TABLE ‘opera‘ (
  ‘IdOpera‘ int NOT NULL AUTO_INCREMENT,
  ‘Nome‘ varchar(45) NOT NULL,
  ‘Periodo‘ varchar(45) DEFAULT NULL,
  ‘Tipologia‘ varchar(45) DEFAULT NULL,
  ‘IdAutore‘ int DEFAULT NULL,
  ‘IdProvenienza‘ int NOT NULL,
  PRIMARY KEY (‘IdOpera‘),
  KEY ‘IdAutore_idx‘ (‘IdAutore‘),
  KEY ‘IdProvenienza_idx‘ (‘IdProvenienza‘),
  CONSTRAINT ‘IdAutore‘ FOREIGN KEY (‘IdAutore‘) REFERENCES ‘autore‘ (‘IdAutore‘),
  CONSTRAINT ‘IdProvenienza‘ FOREIGN KEY (‘IdProvenienza‘) REFERENCES ‘provenienza‘ (‘IdProvenienza‘)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Autore

```
CREATE TABLE ‘autore‘ (
  ‘IdAutore‘ int NOT NULL AUTO_INCREMENT,
  ‘Nome‘ varchar(45) NOT NULL,
  PRIMARY KEY (‘IdAutore‘)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Provenienza

```
CREATE TABLE ‘provenienza‘ (
  ‘IdProvenienza‘ int NOT NULL AUTO_INCREMENT,
  ‘Nome‘ varchar(45) NOT NULL,
  ‘Comune‘ varchar(45) NOT NULL,
  ‘Provincia‘ varchar(45) NOT NULL,
  PRIMARY KEY (‘IdProvenienza‘)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Laboratorio

```
CREATE TABLE ‘laboratorio‘ (
  ‘IdLaboratorio‘ int NOT NULL AUTO_INCREMENT,
  ‘Nome‘ varchar(45) NOT NULL,
  ‘DataInvio‘ date DEFAULT NULL,
  PRIMARY KEY (‘IdLaboratorio‘)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Articolo

```
CREATE TABLE ‘articolo‘ (
  ‘IdArticolo‘ int NOT NULL AUTO_INCREMENT,
  ‘Articolo‘ varchar(45) NOT NULL,
  ‘NomePubblicatore‘ varchar(45) NOT NULL,
  ‘Data‘ date DEFAULT NULL,
  ‘Link‘ varchar(200) DEFAULT NULL,
  PRIMARY KEY (‘IdArticolo‘)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Relatore

```
CREATE TABLE ‘relatore‘ (
  ‘IdRelatore‘ int NOT NULL,
  ‘Nome‘ varchar(45) NOT NULL,
  ‘Afferenza‘ varchar(45) DEFAULT NULL,
  PRIMARY KEY (‘IdRelatore‘)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Citato

```
CREATE TABLE ‘citato‘ (
  ‘IdCampione‘ int NOT NULL,
  ‘IdArticolo‘ int NOT NULL,
  KEY ‘IdArticolo_idx‘ (‘IdArticolo‘),
  KEY ‘IdCampione_idx‘ (‘IdCampione‘),
  CONSTRAINT ‘IdArticolo‘ FOREIGN KEY (‘IdArticolo‘) REFERENCES ‘articolo‘ (‘IdArticolo‘)
   ON DELETE CASCADE,
  CONSTRAINT ‘IdCampione‘ FOREIGN KEY (‘IdCampione‘) REFERENCES ‘campione‘ (‘idCampione‘)
   ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Scritto

```
CREATE TABLE ‘scritto‘ (
  ‘IdArticolo‘ int NOT NULL,
  ‘IdRelatore‘ int NOT NULL,
  KEY ‘IdArticolo_idx‘ (‘IdArticolo‘),
  KEY ‘IdRelatore_idx‘ (‘IdRelatore‘),
  CONSTRAINT ‘IdArticolo1‘ FOREIGN KEY (‘IdArticolo‘) REFERENCES ‘articolo‘ (‘IdArticolo‘)
   ON DELETE CASCADE,
  CONSTRAINT ‘IdRelatore‘ FOREIGN KEY (‘IdRelatore‘) REFERENCES ‘relatore‘ (‘IdRelatore‘)
   ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Le query delle tabelle sono state prese direttamente da MySQL Workshop.

11.2 Backend

Date le specifiche del problema, una volta implementato il DB non ci sono operazioni particolarmente complicate da effettuare su esso; la parte più complicata è sicuramente l'inserimento dei dati.

Per poter effettuare l'inserimento vengono utilizzate diverse query con la seguente struttura:

```
INSERT INTO tabella (colonna1, colonna2)
    SELECT %s, %s
    WHERE NOT EXISTS
      (SELECT * FROM tabella WHERE colonna1 = (%s) AND colonna2 = (%s)), (var1, var2, var1, var2))

SELECT IdTabella FROM tabella WHERE colonna1 = (%s) AND colonna2 = (%s), (var1, var2, var1, var2))
```

La prima query viene utilizzata per l'inserimento, utilizzando delle variabili; prima di tutto la query controlla se esiste già un record all'interno del DB controllando due valori scelti arbitrariamente in maniera sensata.[1]
Nel caso in cui non esista un record con i valori che stiamo cercando di inserire, un nuovo record verrà creato con quei valori.
Nel caso in cui esista già un record con quei valori, non viene inserito nulla.

La seconda query ci serve, successivamente, per andare a prelevare l'id del record che abbiamo appena inserito; questo id verrà utilizzato in query successive per poter inserire correttamente le informazioni relative alle chiavi esterne della tabella.

Andando ad intersecare diverse di queste query all'interno del backend si può così popolare una entry del DB. Ci tengo a precisare che queste query, per quanto eseguite tutte all'interno dello stesso metodo, sono indipendenti l'una dall'altra. L'indipendenza mi è fondamentale per poter capire eventuali errori; il commit del DB[2] viene effettuata solo alla fine di tutte queste query e se una qualsiasi dovesse restituire un errore, questo viene gestito e trasmesso a schermo, bloccando l'inserimento del record e "annullando" le query già eseguite.

Altre query utilizzate nel backend sono quella per eliminare un campione

```
DELETE FROM campione WHERE IdCampione = (%s), (id,)
```

e query per ricercare informazioni specifiche tramite id.

```
SELECT * FROM tabella WHERE tabella.IdTabella = (%s), (id,)
```

[1] Per esempio, per evitare duplicati in "Provenienza", viene controllato se ci sono già record con "Nome" e "Comune" uguale a quello che stiamo inserendo. In generale, a meno di casi molto specifici, controllare anche "Provincia" non è necessario

[2] La funzione che applica effettivamente tutti i cambiamenti al DB

L'ultima query utilizzata viene usata per visualizzare un sommario dei dati più significativi ed è la seguente

```
SELECT campione.Nome, campione.DataP, campione.DataC, campione.LuogoC, stato.Nome as Stato,
       campione.Descrizione, campione.IdOpera, campione.IdLaboratorio, campione.IdCampione
   FROM campione JOIN stato ON campione.IdStato = stato.IdStato
   WHERE campione.Nome = (%s), (search_value,))
```

In effetti vengono utilizzate due versioni di questa query; nella versione sopra riportata viene ricercato un elemento per nome, mentre nell'altra vengono semplicemente restituiti tutti gli elementi[3]. Mi sembra quindi abbastanza superfluo inserirle entrambe.

[3]Stessa query senza l'opzione "WHERE"

Capitolo 12

Conclusioni

Come mio primo approccio al mondo "vero" dei database, spero e credo di essere riuscito ad implementare qualcosa di funzionante e funzionale, ed è stata sicuramente una sfida apprezzata.

Devo ammettere che, dallo studio del problema fino all'implementazione in MySQL e Python3, il progetto è stato più grande del previsto e mi è sfuggito un po' di mano per le tempistiche.

Nonostante tutto, è stato un bel progetto da sviluppare che continuerò ad ampliare e migliorare durante l'estate. La specifica del problema è, infatti, parte di un vero lavoro che dovrò svolgere e ho pensato che portarlo durante un esame sia il modo migliore per imparare e correggere gli errori presenti, così da avere una implementazione ottimale quando verrà effettivamente fatto il deploy.

A tal proposito è molto gradito un feedback sugli errori e su possibili migliorie/future implementazioni, se possibile.

Capitolo 13

Bibliografia

`https://docs.python.org/3/`

Documentazione Python

`https://dev.mysql.com/`

Documentazione e Installazione di MySQL

`https://www.w3schools.com/sql/`

Documentazione e Tutorial per SQL

`https://stackoverflow.com/`

Debug del codice