Test teorico
MySQL – JDBC

20/06/2024

SI – Academy

Andrea Spinelli

1. *Che cos'è un database relazionale e come si differenzia da altri tipi di database?*

   Un **Database Relazione** è un archivio di dati la cui sua peculiarità consiste nell'immagazzinare dati mediante delle tabelle schematizzate interconnesse (o in relazione) tra di loro tramite delle chiavi.
   Le tabelle di un Database Relazione sono composte da record e campi (rispettivamente righe e colonne), dove in ciascun campo viene specificato un attributo dei dati e in ciascun record una medesima serie di attributi con valori differenti. I Database Relazioni inoltre fanno uso di SQL per la gestione e manipolazione dei dati.

   La controparte, **Database NON Relazionale**; invece, non immagazzina i dati fa uso di schemi, bensì li salva in documenti cui non hanno l'obbligo si seguire uno schema preciso.

2. *Quali sono i principali tipi di dati utilizzati in SQL e come vengono definiti?*

   I tipi di dati in SQL si occupano di memorizzare uno specifico tipo di dato che viene passato in tabella; inoltre, si differenziano in varie categorie. Alcuni dei tipi di dati principali sono:

   - INT, memorizza numeri di tipo intero, esistono a sua volta le varianti SMALLINT, BIGINT, …;
   - FLOAT, memorizza numeri reali in virgola mobile a singola precisione;
   - DOUBLE, memorizza numeri reali in virgola mobile a doppia precisione;
   - DECIMAL, memorizza numeri reali con una precisione fissata e scala;
   - BINARY, memorizza dati binari di lunghezza fissa (passato il numero di bit a parametro);
   - CHAR, memorizza stringhe di lunghezza fissa;
   - VARCHAR, memorizza stringhe di lunghezza variabile;
   - TEXT, memorizza stringhe di grandi lunghezze;
   - DATETIME, memorizza sia la data che l'orario, esistono a sua volta le singole controparti (DATE e TIME);

   I tipi di dato vengono definiti al momento della creazione di una tabella, alcuni esempi sono:

```
CREATE TABLE EsempioUno(
     matricola INT
)
```

```
CREATE TABLE EsempioDue(
     Prezzo DECIMAL(100,2)
)
```

```
CREATE TABLE EsempioTre(
     Nome VARCHAR(255)
)
```

3. *Spiegare la differenza tra chiave primaria, chiave esterna.*

   La differenza tra chiave primaria e chiave esterna sta nel fatto che: la prima identifica univocamente ciascun record di una tabella; mentre la seconda è una chiave di riferimento ad una chiave primaria di un'altra tabella, stabilendo una relazione tra i record delle tabelle, assicurando coerenza e integrità referenziale dei dati.
   Per entrambe è possibile specificare una o più campi per l'univocità e l'integrità referenziale.

```
CREATE TABLE EsPrimary(
     prestito_id INT PRIMARY KEY
)

CREATE TABLE EsForeign(
     utente_id INT PRIMARY KEY
     prestito_id INT
     FOREIGN KEY(prestito_id)
REFERENCES EsPrimary(prestito_id)
)
```

4. *Descrivere le diverse clausole utilizzabili in una query SELECT, fornendo esempi concreti.*

   La clausola SELECT permette di effettuare delle query per selezionare dei campi e record di una tabella, anche secondo alcun specifiche condizioni, quest'ultime però sono stabilite da altre clausole che vengono poste dopo la clausola SELECT. Prima di tutto la sintassi base di una SELECT è la seguente

   ```
   SELECT <campo_1>, <campo_2>, …
   FROM <nome_tabella>
   ```

   Le altre clausole utilizzabili, rispettivamente in questo ordine, in una query SELECT sono:

   - WHERE, filtra i record della tabella secondo una o più condizioni specifiche;

     ```
     SELECT <campo_1>, <campo_2>, …
     FROM <nome_tabella>
     WHERE <campo_x> = <valore>
     ```

     WHERE inoltre ha una serie clausole che permettono di stabilire delle condizioni:
     - AND, verificare un ulteriore condizione;
     - OR, verifica se almeno una condizione o entrambe siano vere;
     - IN, verifica se il campo rispetta uno dei valori passati a parametro;
     - LIKE, verifica il campo tramite un patter di caratteri;
     - NOT, verifica la condizione contraria;
     - BETWEEN, verifica se il campo di trova in un range di valori;
     - …

     ```
     SELECT <campo_1>, …
     FROM <nome_tabella>
     WHERE <campo_x> = <val_1>
     AND/OR <campo_y> = < val _2>
     ```

     ```
     SELECT <campo_1>, …
     FROM <nome_tabella>
     WHERE <campo_x> IN (<val _1>, < val _2>, …)
     ```

     ```
     SELECT <campo_1>, …
     FROM <nome_tabella>
     WHERE <campo_x> LIKE "<pattern>"
     ```

     ```
     SELECT <campo_1>, …
     FROM <nome_tabella>
      WHERE <campo_x> BETWEEN <val_1> AND <val_2>
     ```

   - JOIN, unisce la tabella con un'altra, rispetto a un criterio di insiemistica specifica;

     ```
     SELECT <campo_1>, <campo_2>, …
     FROM <tab_1>
     JOIN < tab _2> [AS <alias>] REFERENCES < tab_1.foreignKey> = < tab _2.primaryKey>
     ```

   - GROUP BY, raggruppa i record rispetto a un campo nel caso in cui si hanno dei record duplicati;

     ```
     SELECT <condizione che restituisce duplicati>, ….
     FROM <nome_tabella>
     GROUP BY <campo_x>
     ```

   - HAVING, filtra ulteriormente i record della tabella ma solo per dei gruppi o per funzioni di aggregazione;
   - ORDER BY, ordina i record della tabella secondo uno o più campi;

     ```
     SELECT <campo_1>, <campo_2>, …
     FROM <nome_tabella>
     ORDER BY <campo_x> [ASC/DESC]
     ```

5. *Qual è la differenza tra un'INNER JOIN, una LEFT JOIN e una RIGHT JOIN*?

   La differenza tra i vari JOIN si basa principalmente sull'insiemistica, le differenze sono:

   - INNER JOIN, restituisce tutti record delle tabelle che hanno in comune chiave primaria e chiave esterna, escludendo i record che non hanno alcuna relazione;
   - LEFT JOIN, restituisce tutti record delle tabelle rispetto alla tabella a sinistra, includendo i record che non hanno relazioni lasciando i campi nulli;
   - RIGHT JOIN, restituisce tutti record delle tabelle rispetto alla tabella a destra, includendo i record che non hanno relazioni lasciando i campi nulli;