A.A. 2021 - 2022

LIBRARY SEAT RESERVATION

HYPERSQL

LIBRARY SEAT RESERVATION

INTRODUZIONE

HSQL Database Engine (**HSQLDB** o **HyperSQL**) è un *database management system* relazionale (**RDBMS**) scritto completamente in Java.

È leggero, veloce, supporta il multithreading ed ha la capacità di lavorare sia in modalità *in-memory* che *in-process*, mettendo a disposizione delle applicazioni diverse modalità di utilizzo (*server*, *web server* e *servlet*). Include anche un programma da riga di comando (**SqlTool**) per interagire con il database in linguaggio SQL.

Può essere installato sia scaricando il **HSQLDB JDBC Driver** (file `.jar` da aggiungere nel *classpath* del progetto), oppure utilizzando **Maven** (specificando l'id `org.hsqldb.hsqldb`).

Si tratta di uno strumento adatto per supportare quelle applicazioni che non richiedono tutte le caratteristiche offerte da altri DBMS (es. MySQL), o per fare pratica con framework come **JPA**.

MODALITÀ DI UTILIZZO

- **Modalità Server:** il database viene eseguito in una *Java virtual machine* (JVM) e ascolta le connessioni da programmi che stanno sullo stesso computer o su altri computer all'interno della rete. Esistono tre modalità server, basate sul protocollo utilizzato per le comunicazioni tra client e server:
  - **Server:** viene utilizzato un protocollo di comunicazione proprietario, generalmente in ascolto sulla porta TCP 9001 — questa è la modalità più veloce e consigliata per avviare il database. Il formato dell'URL di connessione è del tipo:

    ```
    jdbc:hsqldb:hsql://db
    ```

  - **Web Server:** questa modalità viene utilizzata quando l'accesso al computer che ospita il server del database è limitato al protocollo HTTP (ad esempio in caso di restrizioni imposte da un firewall) — in questo caso viene avviato un server Web speciale che consente ai client di connettersi tramite HTTP. Il formato dell'URL di connessione è del tipo:

    ```
    jdbc:hsqldb:http://db
    ```

    - **Servlet:** come Web Server, ma questa modalità viene utilizzata quando è un *servlet engine* separato, come **Tomcat** o **Resin**, a fornire l'accesso al database — questa modalità però può servire solo un singolo database alla volta.

- **Modalità in-process (standalone):** questa modalità utilizza il *filesystem* per eseguire il database come parte dell'applicazione Java, in particolare nella stessa *Java Virtual Machine* (JVM). Questo ha il risultato di favorire accessi più rapidi, poiché i dati non vengono convertiti e inviati in rete, ma lo svantaggio principale è che non è possibile connettersi al database dall'esterno dell'applicazione: non è quindi possibile gestire il contenuto del database mentre l'applicazione è in esecuzione. Il formato dell'URL di connessione è del tipo:

  `jdbc:hsqldb:file:db`

- **Modalità in-memory:** non prevede persistenza, ma il database viene allocato per intero nella RAM. Questa modalità risulta utile nel caso di elaborazione interna dei dati dell'applicazione oppure in fase di testing. In questo caso, il formato dell'URL di connessione è del tipo:

  `jdbc:hsqldb:mem:db`

In tutti i casi, HSQL mette a disposizione un utente con privilegi di amministratore con le seguenti credenziali:

Username: sa

Password: stringa vuota

Ad esempio, una connessione ad un database HSQL in-memory può essere effettuata in Java aggiungendo queste proprietà al file `persistence.xml`:

```
<property name="javax.persistence.jdbc.driver" value="org.hsqldb.jdbcDriver"/>
<property name="javax.persistence.jdbc.url" value="jdbc:hsqldb:mem:standalone"/>
<property name="javax.persistence.jdbc.user" value="sa"/>
<property name="javax.persistence.jdbc.password" value=“"/>
```

VANTAGGI

- Supporto molto esteso per la sintassi standard di SQL (2016), inclusa la maggior parte delle funzionalità opzionali;
- Tabelle in memoria per operazioni più veloci;
- Tabelle basate su disco per set di dati di grandi dimensioni;
- Tabelle con supporto a dati esterni (es. file CSV) possono essere utilizzate come tabelle SQL.

PERFORMANCE

HyperSQL ha più opzioni di distribuzione e persistenza che ne influenzano le prestazioni:

- Il **tipo di tabella** (*MEMORY*, *CACHED* o *TEXT*) indica come vengono archiviati i dati di ciascuna riga della tabella e come vi accede il database;
- La modalità **in-process** o **server** indica in che modo l'applicazione accede ai dati del database;
- Il **modello di transazione** indica come e quando diverse sessioni (connessioni) rimangono in attesa l'una con l'altra.

In particolare le prestazioni dei diversi **tipi di tabella** sono:

- Le tabelle **MEMORY** offrono le massime prestazioni. Tutti i dati sono in memoria e ogni campo di ogni riga è un oggetto di memoria che può essere letto dal database senza alcuna conversione. Quando i dati vengono aggiornati, viene scritto su disco solo un record di log, con un *overhead* molto basso;
- Le tabelle **CACHED** hanno prestazioni inferiori rispetto alle tabelle MEMORY. I dati per questo tipo di tabella provengono da una cache che contiene un sottoinsieme di tutte le righe in tutte le tabelle CACHED. La riduzione delle prestazioni è dovuta al fatto che la dimensione della cache delle righe è in genere inferiore al totale delle righe di tutte le tabelle CACHED: per questo le righe vengono spesso eliminate dalla cache e altre righe vengono lette dal disco e convertite in oggetti di memoria.
- Le tabelle **TEXT** hanno somiglianze sia con le tabelle CACHED che con le tabelle MEMORY. Gli indici vengono mantenuti in *memoria*, mentre i dati vengono

mantenuti su *disco* e memorizzati nella cache come le tabelle CACHED. Poiché i dati vengono archiviati come valori separati da virgola (file CSV), la lettura e la scrittura dei dati richiede più tempo rispetto alla stessa operazione in formato binario.

Le prestazioni della modalità **in-process**, rispetto alla modalità **server**, sono:

- L'accesso *in-process* avviene nello stesso spazio di memoria dell'applicazione: non c'è alcuna conversione dei dati o *overhead* dovuto al trasferimento;
- La modalità *server*, al contrario, ha uno spazio di memoria diverso rispetto all'applicazione. I dati vengono convertiti in un flusso di byte, trasferiti sulla rete e quindi riconvertiti in oggetti: questo introduce una latenza, da sommare all'*overhead* dovuto all'elaborazione aggiuntiva necessaria per la conversione;
- L'accesso in modalità *server* può essere velocizzato utilizzando le **stored procedures** SQL, che consentono di incapsulare un'intera transazione in un'unica istruzione SQL.

HyperSQL supporta due **modelli di transazione** per il controllo della concorrenza: **multiversion concurrency control** (**MVCC**) e **two-phase locking** (**2PL**). Le differenze in termini di prestazioni sono:

- HyperSQL ha il supporto completo al *multithreading*, per cui se la maggioranza delle operazioni sono di *lettura*, le prestazioni sono molto elevate in tutti i modelli di transazione;
- Se è presente una quantità significativa di operazioni di aggiornamento, le prestazioni del modello di blocco *2PL* possono essere ricondotte al caso in cui ci sia un singolo thread: ci sono infatti dei *lock* sia in lettura che in scrittura;
- Sotto le stesse condizioni, MVCC offre prestazioni notevolmente maggiori rispetto al 2PL, poiché non vengono utilizzati *lock* di lettura, mentre i *lock* di scrittura vengono mantenuti solo sulle singole righe aggiornate. Più thread possono leggere e aggiornare il database.

In sintesi, le prestazioni più veloci si ottengono in genere con la combinazione di tabelle **MEMORY**, accesso **in-process** e modello di transazione **MVCC**. Se è richiesto un utilizzo ridotto della memoria, alcune tabelle possono essere definite come tabelle **CACHED**, mantenendo invece le tabelle a cui si accede più frequentemente come tabelle **MEMORY**. Se l'accesso è di tipo **server**, è possibile utilizzare le *stored procedures* SQL per ridurre la latenza dovuta ai round trip della rete.