

## INTRODUZIONE AI DATABASE

Un database è un sw che consente di gestire salvataggio, recupero, analisi, aggiornamento dei dati.

Tipi di DBMS (Database ManagementSystem):

- RDBMS: Relational DBMS
- NoSQL database

### RDBMS

I dati sono organizzati in tabelle.

Tipi di RDBMS:

- A pagamento: Oracle, SQLServer
- Free: MySQL, Postgresql

Operazioni **CRUD**: Create, Read, Update, Delete

## MYSQL

Database open-source più diffuso al mondo.

- MySQL CommunityEdition:
  - InnoDB: motore più performante

Dal sito:

The MySQL Community Edition includes:

- **SQL and NoSQL** for developing both relational and NoSQL applications
- **MySQL Document Store** including X Protocol, XDev API and MySQL Shell
- **Transactional Data Dictionary** with Atomic DDL statements for improved reliability
- **Pluggable Storage Engine Architecture** (InnoDB, NDB, MyISAM, etc)
- **MySQL Replication** to improve application performance and scalability **[Utile quando si hanno molti dati e molti accessi al DB]**
- **MySQL Group Replication** for replicating data while providing fault tolerance, automated failover, and elasticity
- **MySQL InnoDB Cluster** to deliver an integrated, native, high availability solution for MySQL
- **MySQL Router** for transparent routing between your application and any backend MySQL Servers
- **MySQL Partitioning** to improve performance and management of large database applications **[Migliora l'accesso ai file del file system]**
- **Stored Procedures** to improve developer productivity
- **Triggers** to enforce complex business rules at the database level
- **Views** to ensure sensitive information is not compromised **[Query che fanno vedere all'utente dei dati, ma che non gli permettono di modificarli]**
- **Performance Schema** for user/application level monitoring of resource consumption
- **Information Schema** to provide easy access to metadata
- **MySQL Connectors** (ODBC, JDBC, .NET, etc) for building applications in multiple languages **[Fa interagire il DB con altri linguaggi di programmazione. In Java JDBC Connector]**
- **MySQL Workbench** for visual modeling, SQL development and administration **[Client che consente di accedere al DB]**

## NOTA:

Quando si lavora con un DB è conveniente non lavorare mai direttamente come utente root, ma creare un utente che abbia gli stessi privilegi di un utente root e fare interfacciare lui con l'applicazione

- Si evita di passare da root, perché se qualcuno buca il DB, ha accesso a tutti i DB presenti su mysql
- Creando un utente apposito, eventuali attacchi saranno limitati soltanto al DB specifico cui quell'utente ha accesso.

## DOWNLOAD MYSQL:

Due possibilità:

1. MySQL: <https://www.mysql.com/it/products/community/>
  - Download MySQLCommunityServer
  - MySQLCommunityServer
2. XAMPP: <https://www.apachefriends.org/it/index.html>
  - Contiene un set di applicazioni, tra cui MySQL


### MYSQL

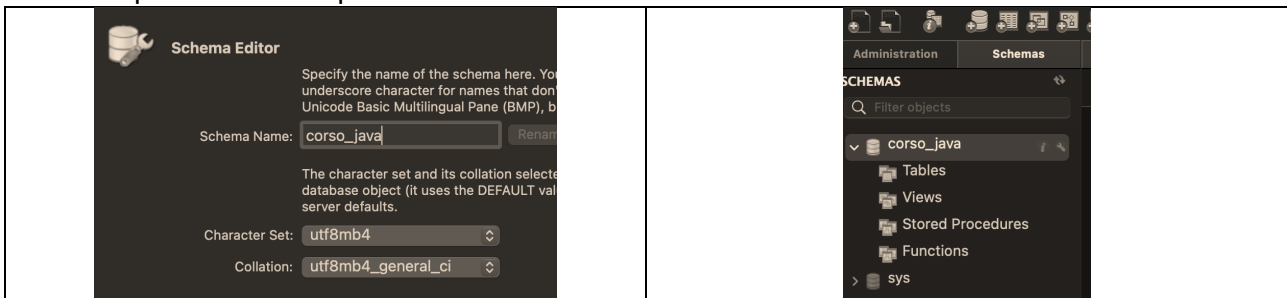
Preferenze di sistema – MYSQL – Start MySQL Server

### MARIADB

mysql -u root --password=pao1oMYSQL1990

## CREARE DB CON MYSQL

- Assicurarsi che MySQL sia connesso
- Tasto “create new schema” 
- Impostare nomi e parametri:



- Cliccando su apply compare la query standard per la creazione dello schema:

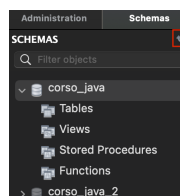
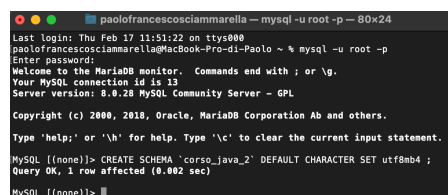
```
CREATE SCHEMA `corso_java` DEFAULT CHARACTER SET utf8mb4 ;
```

- Si può fare la stessa cosa anche da riga di comando [terminale]:

```
mysql -u root -p [oppure: mysql -u root --password=pao1oMYSQL1990]
```

→ posso scrivere le istruzioni mysql

```
CREATE SCHEMA `corso_java_2` DEFAULT CHARACTER SET utf8mb4 ;
```



## CREARE TABELLA CON MYSQL

- Esplodo lo schema in cui inserire la tabella
- Tables – tasto dx – Create table
  - PK = PrimaryKey
  - NN = Non null
  - UQ = Unique (index for the column)
  - B = Binary
  - UN = Unsigned
  - ZF = Zero fill
  - AI = Auto-increment (genera da solo ed incrementa)
  - G = Generated column
- Inserisco le colonne e clicco su apply: verrà generato lo script per la creazione della tabella:

```

1 CREATE TABLE `corso_java`.`clienti` (
2   `idclienti` INT NOT NULL AUTO_INCREMENT,
3   `nome` VARCHAR(255) NOT NULL,
4   `cognome` VARCHAR(255) NOT NULL,
5   `email` VARCHAR(100) NOT NULL,
6   `telefono` VARCHAR(20) NOT NULL,
7   PRIMARY KEY (`idclienti`));
8
  
```

- Creazione chiave esterna (il dominio dei di una colonna è preso da un'altra colonna: posso inserire solo i dati presenti in colonna di altra tabella):

Esempio: creazione tabella ordini

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Defa
idordini	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
id_cliente	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
data_consegna	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
importo	DOUBLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

- Nome della ForeignKey
- Tabella da cui prendere i valori
- Settaggio vincolo:
  - Column: colonna tabella attuale da vincolare
  - Ref column: colonna da cui prendere i dati da usare come vincolo

Foreign Key	Referenced Table	Column	Referenced Column	On Update:	On Delete:	Comment:
id_cliente	corso_java`.`clienti`	id_cliente	idclienti	NO ACTION	NO ACTION	

SQL Generato:

```

CREATE TABLE `corso_java`.`ordini` (
  `idordini` INT NOT NULL AUTO_INCREMENT,
  `id_cliente` INT NULL,
  `data_consegna` DATETIME NULL,
  `importo` DOUBLE NULL,
  PRIMARY KEY (`idordini`),
  INDEX `id_cliente_idx` (`id_cliente` ASC) VISIBLE,
  CONSTRAINT `id_cliente`
    FOREIGN KEY (`id_cliente`)
    REFERENCES `corso_java`.`clienti` (`idclienti`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
  
```

## INSERIRE RIGHE IN TABELLA:

- Tasto dx: insert new row
- Oppure: dal lato cliccando sul bottone +
- Inserendo i dati a mano e cliccando poi su Apply, verranno effettuate le insert

```

INSERT INTO `corso_java`.`clienti` (`nome`, `cognome`, `email`, `telefono`) VALUES ('Avril', 'Lavigne', 'test@test.com', '987');
  
```

idclienti	nome	cognome	email	telefono
1	Avril	Lavigne	test@test.com	987

## NOTA:

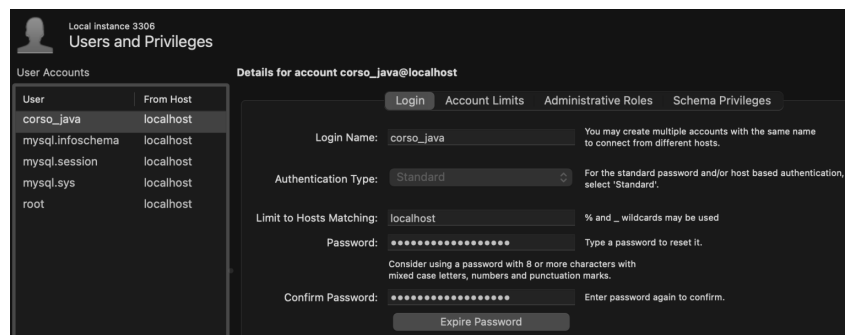
- Se si inserisse una riga che ha come una foreign-key come tra i suoi dati, non si può cancellare la relativa riga dalla tabella d'origine, senza aver prima cancellato quella dalla tabella di destinazione, in cui è usato il riferimento

<p>Es:</p> <ul style="list-style-type: none"><li>Tabella cliente con id</li><li>Tabella ordine che usa id della tabella cliente</li></ul> <p>Non si può eliminare dalla tabella cliente una riga il cui id è usato da ordini:</p> <ul style="list-style-type: none"><li>Prima eliminare da ordini la riga che usa il riferimento</li><li>Poi si elimina da cliente la riga iniziale</li></ul>	<p><b>ERROR 1451: 1451: Cannot delete or update a parent row: a foreign key constraint fails</b></p> <pre>(`corso_java`.`ordini`, CONSTRAINT `id_cliente` FOREIGN KEY (`id_cliente`) REFERENCES `clienti` (`idclienti`)) SQL Statement: DELETE FROM `corso_java`.`clienti` WHERE (`idclienti` = '1')</pre>
---	--

## CREAZIONE UTENTE CON PRIVILEGI PER AGIRE SU DI UN SOLO SCHEMA (SICUREZZA)

Da root:

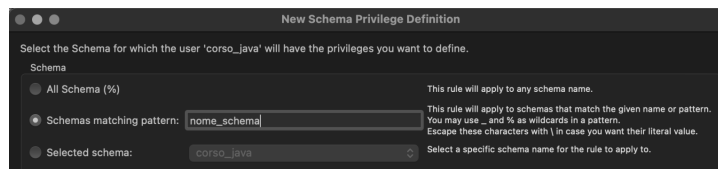
- Management -> Users and Privileges -> Add account



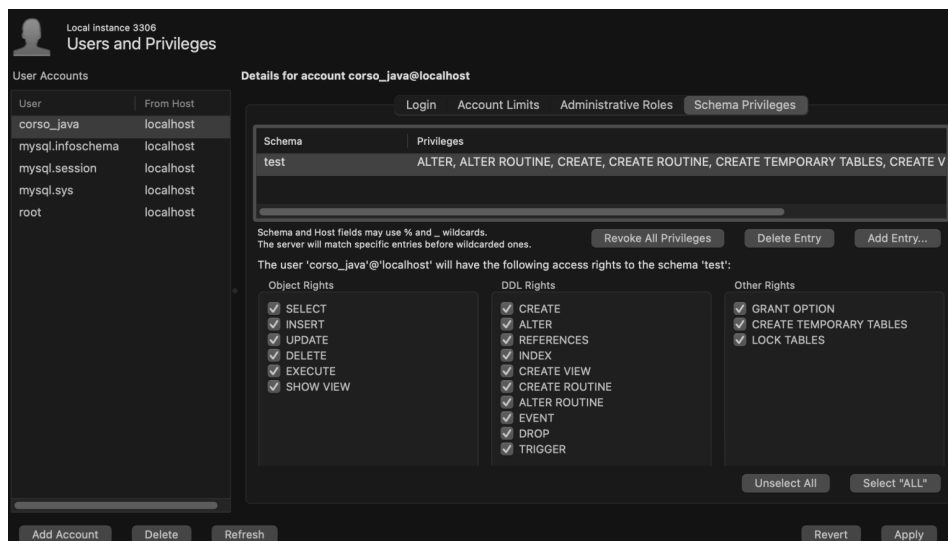
- Si assegnano i permessi:
  - Administrative Roles: se si vogliono assegnare ruoli amministrativi
  - Schema Privileges: se si vuole assegnare la possibilità di gestire/creare schemi

Schema Privileges:

- Add Entry: si aggiunge lo schema per il quale si vogliono assegnare i permessi



- Se si volessero assegnare permessi per la creazione e gestione di uno schema preciso (cosicché se viene bucato il DB, non si ha modo di attaccare gli altri schemi), bisogna specificare il nome dello schema da far gestire: **schemas matching pattern**
  - NOTA:** l'utente potrà solo creare/gestire schema con il nome specificato; non può creare/gestire altri schemi
- Una volta che si è aggiunta la entry, si specificano i permessi che le si assegnano: lettura/scrittura/creazione db / indicizzazione etc
- Si clicca su Apply



### SINTASSI OPERAZIONI CRUD

<b>CREATE: Inserire riga</b>	INSERT INTO nome_tabella(campo1, campo2, ...) VALUES (valore1, valore 2, ...);
<b>READ: Leggere da DB</b>	SELECT campo1, campo2, ... FROM nome_database.nome_tabella; WHERE condizioni_di_ricerca  <ul style="list-style-type: none"> <li>= selezione tutto</li> <li>Nella FROM posso anche specificare direttamente nome_tabella</li> </ul>
Condizione WHERE	<ul style="list-style-type: none"> <li>= : uguaglianza stretta</li> <li>WHERE nome_campo LIKE '%sequenza%' : permette di selezionare le righe che contengono la sequenza di caratteri</li> </ul>
Aliasing AS	SELECT nome_colonna AS 'nuovo_nome_colonna' FROM nome_tabella AS nuovo_nome_tabella
Vincolo SELECT	<ul style="list-style-type: none"> <li>LIMIT int : limita i record restituiti al numero specificato</li> <li>DISTINCT: solo record distinti</li> </ul>
Vincoli dopo WHERE	1. ORDER BY nome_campo ASC/DESC: si scrive dopo WHERE
<b>UPDATE: aggiornamento riga</b>	UPDATE nome_tabella SET campo1 = val1, campo2 = val2, ... WHERE condizione_filtraggio
<b>DELETE: elimino riga</b>	DELETE FROM nome_tabella WHERE condizione_cancellazione

## INTERFACCIARE IL DB CON JAVA

1. Scaricare il **JDBC Connector**: jar che contiene le librerie per la connessione con il db  
+ Oppure se si sta scrivendo un progetto Maven, si aggiunge nel pom la dipendenza

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.28</version>
</dependency>
```

2. Creazione di una variabile di tipo **Connection** che si incarica della creazione della connessione: è un singleton
3. Per creare la connessione:
  - Instancio un data source di tipo **MysqlDataSource**
  - Imposto i parametri di connessione
  - Chiedo la connessione al datasource

```
import java.sql.Connection;
import java.sql.SQLException;
import com.mysql.cj.jdbc.MysqlDataSource;

public class ConnessioneDBMS {
    private Connection con;

    private Connection getConnection() {
        if(con==null) {
            MysqlDataSource dataSource = new MysqlDataSource();
            //SETTAGGIO PARAMETRI: NOME SCHEMA, PORTA, HOST, USER, PASS
            dataSource.setServerName("127.0.0.1");
            dataSource.setPortNumber(3306);
            dataSource.setUser("root");
            dataSource.setPassword("paoloMYSQL1990");
            dataSource.setDatabaseName("corso_java");
            //CREAZIONE CONNESSIONE
            try {
                con=dataSource.getConnection();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return con;
    }

    public static void main(String...args) {
        ConnessioneDBMS c = new ConnessioneDBMS();
        c.getConnection();
    }
}
```

## OPERAZIONI DI CRUD MEDIANE IL JDBC CONNECTOR

### SELECT

Classi necessarie:

- **PreparedStatement**: oggetto che serve a configurare la query da eseguire
- **executeQuery()**: metodo di PS che esegue la query

- **ResultSet**: oggetto che raccoglie i risultati prodotti dalla query
  - **getTipo(int indexColumn)**: metodi per prelevare i dati di uno specifico tipo; si passa l'index della colonna partendo da 1

```
private void esSelect() {
    String query = "SELECT * FROM clienti";
    try {
        //OGGETTO PER PASSARE LA QUERY IN INPUT A MYSQL
        PreparedStatement ps = getConnection().prepareStatement(query);
        ResultSet rs = ps.executeQuery();
        while(rs.next()) {
            //I METODI GET RESTITUISCONO L'OGGETTO DELLA RELATIVA
            //COLONNA ASSOCIATA (GLI INDICI PARTONO DA 1)
            System.out.println(rs.getString(2)+" "+rs.getString(3));
        }
    } catch (SQLException e) { e.printStackTrace(); }
}
```

## INSERT

Classi necessarie:

- **PreparedStatement**: oggetto che serve a configurare la query da eseguire
- **executeUpdate()**: metodo di PS che esegue la query
- **ResultSet**: oggetto che raccoglie i risultati prodotti dalla query
  - **getGeneratedKeys(int indexColumn)**: metodo per mettere in rs la chiave generata automaticamente dalla query

```
private void esInsert(String nome, String cognome, String email, String tel) {
    //I VARCHAR DEVONO ESSERE PASSATI TRA ' '
    String query = "INSERT INTO clienti(nome, cognome, email, telefono)"+
        "VALUES ('"+nome+"', '"+cognome+"', '"+email+"', '"+telefono+"')";
    //OGGETTO PER PASSARE IN INPUT A MYSQL LA QUERY
    try {
        //SPECIFICANDO IL PARAMETRO PER L'AUTO-INCREMENT, SI OTTIENE
        //LA RESTITUZIONE DEL PARAMETRO AUTOMATICO GENERATO COME CHIAVE
        PreparedStatement ps =
            getConnection().prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
        ps.executeUpdate();
        ResultSet rs = ps.getGeneratedKeys();
        while(rs.next()) {
            System.out.println("ID CHIAVE:"+rs.getInt(1));
        }
    } catch (SQLException e) { e.printStackTrace(); }
}
```

## UPDATE

Classi necessarie:

- **PreparedStatement**: oggetto che serve a configurare la query da eseguire
- **executeUpdate()**: metodo di PS che esegue la query

```
private void esUpdate(String telefono) {
    String query = "UPDATE clienti SET telefono = '"+telefono+"' WHERE idclienti=1" ;

    //OGGETTO PER PASSARE IN INPUT A MYSQL LA QUERY
    try {
        PreparedStatement ps =
            getConnection().prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
        ps.executeUpdate();
    } catch (SQLException e) {e.printStackTrace();}
}
```

## DELETE

Classi necessarie:

- **PreparedStatement**: oggetto che serve a configurare la query da eseguire

- **executeUpdate():** metodo di PS che esegue la query

```
private void esDelete() {
    String query = "DELETE FROM clienti WHERE idclienti = 2" ;
    try {
        PreparedStatement ps =
            getConnection().prepareStatement(query,Statement.RETURN_GENERATED_KEYS);
        ps.executeUpdate();
    } catch (SQLException e) { e.printStackTrace(); }
}
```

## PREVENIRE LA SQL INJECTION

Tipo di attacco: Quando si esegue la query, all'interno è presente un'istruzione malevola, che può compromettere il funzionamento del db (es. DROP table)

- In Java per prevenire attacchi di questo tipo, si usa il **PreparedStatement**

**Es#1:** query fragile – un hacker può eseguire la query passando un nome

```
private void esSelectAttaccabile(String nome) {
    String query = "SELECT * FROM clienti WHERE nome='"+nome+"'";
    try {
        PreparedStatement ps = getConnection().prepareStatement(query);
        ResultSet rs = ps.executeQuery();
    } catch (SQLException e) { e.printStackTrace(); }
}
```

**Es#2:** query più sicura – controlli fatti dal metodo

```
private void esSelectPiuSicura(String nome) {
    String query = "SELECT * FROM clienti WHERE nome=?";
    try {
        PreparedStatement ps = getConnection().prepareStatement(query);
        ps.setString(1, nome);
        ResultSet rs = ps.executeQuery();
    } catch (SQLException e) { e.printStackTrace(); }
}
```

Nella query si pone un ? come segnaposto per il parametro da inserire/cercare

- **setTipoParam(int indexDel?, valore):** l'indice del punto in cui si trova il ? parte da 1

In questo corso abbiamo visto un modo per interagire con il DB:

- JDBC Connector

Qualora ci si dovesse trovare a lavorare con altre applicazioni più complesse, si possono usare altri strumenti più complessi:

- Hibernate
- JPA

Sono degli ORM, ovvero, uno strato applicativo in più, che si occupa di:

- Stabilire connessioni al DB
- Creare pool di connessioni
- Datasource
- Consentono attraverso delle classi particolari, di prevenire degli attacchi
- Ottimizzare inserimento e ricerca dei dati