Database initialization

Spring Boot and Database Initialization

- **Spring Boot** is hands down a great framework, saving the developer a lot of time and energy when developing a Spring application. One of its great features is database initialization.
- **Spring Boot** can automatically create the schema (DDL scripts) of your DataSource and initialize it (DML scripts). It loads SQL from the standard root classpath locations: schema.sql and data.sql, respectively. In addition, Spring Boot processes the schema-${platform}.sql and data-${platform}.sql files (if present), where a platform is the value of spring.datasource.platform. This allows you to switch to database-specific scripts if necessary.

Sample JPA class

```java
@Entity
@Table(name = "user")
public class User {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;
    private String name;

    public User() { }

    public User(Integer id, String name)
    {
        this.id = id;
        this.name = name;
    }

    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

data.sql

When we run our application, **Spring Boot** will create an empty table, but won't populate it with anything.
An easy way to populate some data in database tables while application startup is to create a file named data.sql.

```
INSERT INTO `users_database`.`user` (`id`, `name`) VALUES ('1', 'Salman');
INSERT INTO `users_database`.`user` (`id`, `name`) VALUES ('2', 'SRK');
INSERT INTO `users_database`.`user` (`id`, `name`) VALUES ('3', 'AMIR');
INSERT INTO `users_database`.`user` (`id`, `name`) VALUES ('4', 'Tiger');
INSERT INTO `users_database`.`user` (`id`, `name`) VALUES ('5', 'Prabhas');
```

When the project run with this file on the classpath, **Spring Boot** will pick it up and use it for populating the database.