

Hibernate Tutorial Notes

A framework for persisting / saving java objects in a database

ORM — object-to-relational mapping

- the developer defines mapping between java class and database table

CRUD — Create - Read - Update - Delete

- **Hibernate vs. JDBC ?**

- hibernate uses JDBC for all database communications

First of all create database with:

```
String jdbcUrl = "jdbc:mysql://localhost:3306/hb_student_tracker?
user55L=false";
String user = "hbstudent";
String pass = "hbstudent";
```

need a hibernate config file

—> **java annotations**

Entity Class — Java class that is mapped to a database table

- **Java Annotations**

1. map class to database — @Table — on top of object class
2. map fields to database columns — @Column(name="column_name") — on fields
(note need @Id on id field)
(if column name == field name, then annotation not needed)

SessionFactory

- Reads the hibernate config file
- Create Session objects
- Heavy-weight object, meaning only create once in app

Session

- Wraps a JDBC connection
- Main object used to save/retrieve objects
- Short-lived object
- Retrieved from SessionFactory

**** **Code: hibernate-tutorial/.../CreateStudentDemo** ****

Primary Key (e.g. id)

- Unique identifies each row in a table
- Must be a unique value

- Cannot contain NULL values
- **@GeneratedValue(strategy=GenerationType. ...)**
 - AUTO — pick an appropriate strategy for the particular data
 - IDENTITY — assign primary keys using identity column
 - SEQUENCE — assign primary keys using a database sequence
 - TABLE — assign primary keys using an underlying database table to ensure uniqueness
- can also customize strategy
 - create subclass `org.hibernate.id.DequenceGenerator`
 - override method: `public Serializable generate(...)`
 - much to worry about