* SUPER ASIDE
  + Cascade is a keyword in sl that cause commands to propagate to other tables
  + On update or delete
  + If I delete a row in role, I also delete all users that had that role.
* Hibernate?
* Is a framework, made by Gavin King
  + What is an orm
    - Object Relational Mapper
    - This means we map our objects (models) to tables in our database
    - Movie model and tie it to a movie table in a db
  + What are the benefits of an orm?
    - We don’t have to write sql queries anymore
    - It allows us to use a sql db without having to worry about what dialect of sql we are using
    - Allows to manipulate database in a more OOP way
  + What are the downsides of an orm
    - It’s more complicated
    - It has more upfront configuration
    - It makes errors harder to trace
* How do I configure Hibernate
  + There is a special class hibernate has called Configuration
  + Using an instance of COnfiguration object from hibernate we can build the rest of the objects we need
  + We build that Configuration using an xml file
  + What are some important tags for my xml
    - Hibernate.dialect - what sql dialect
    - Hibernate.connection.username
    - Hibernate.connection.password
    - Hibernate.connection.url
    - Hibernate.connection.pool\_size
      * Just like pg library for node we utilize connection pooling in hibernate
      * (Benefits of connection pooling review)
  + What is the hbm2ddl tag
    - This tag has 4 different values
      * Create - when the program starts up, it completely rebuilds the database schema using the current database model mappings
      * Update - will only rerun ddl statements if they have changed since last time
      * Validate - just make sure the database and the mappings have the same effect or it won’t start
      * None - do nothing with ddl
  + What are mapping tags
    - These tags tell hibernate what classes or files we use to define the database mappings
* Mappings
  + What are the 2 ways to generate object mappings in hibernate?
    - We can create a .hbm.xml file that contains many xml tags that correspond to fields on a class and how to relate those fields to the database
    - We can add annotations on a class and above fields and put the relational information there
  + What annotations are required in a mapping
    - @Entity - tells hibernate this is a model mapping
    - @Id - this one denotes which field is the primary key on the object
  + How does this all work?
    - By default Hibernate assumes the column name in the database is the exact same as the field name on the object
    - Hibernate knows all of the different objects we have mapped and can tell if we a have a field of another mapped object that it will have to resolve a FK relationship ( hibernate will write your joins for you )
  + Annotations for multiplicity?
    - @ManyToOne - with this we put a @JoinColumn ( we put this on the many side )
    - @OneToMany - mappedBy property (we put this on the one side )
    - @ManyToMany - @JoinTable - joinColumn and inverseJoinColumn
      * Actually builds the join table when doing auto ddl
  + Annotations for manipulating the column in the db?
    - @Column - specify column name, column type and such forth
* JDBC?
  + Java database connectivity
  + It’s basically just the pg library but for java
* Interfaces of Hibernate
  + What are the interfaces of Hibernate
    - Class Configuration - this is how we build a hibernate session factory
    - I SessionFactory - connection pool, it has methods to give us sessions
    - I Session - this represents a connection to the database, it has built in crud methods and can give us transaction objects, query objects and criteria objects.
    - I Transaction - interface that abstracts the process of build db transactions
    - I Query - this is for writing HQL or SQL queries to the database
    - I Criteria - this is for building a select statement in a very object oriented way. ( I don’t use very often)
  + Explain the purpose of each and some methods you might use
    - SessionFactory.openSession()
    - Session.close()
    - Transaction.begin() .commit()
    - Query. .list() .setParameter()
    - Criteria .add(Restriction Object) .list()
  + Session Crud Methods
    - Save
      * Take an object that isn’t in the db (has a pk 0) save it into the database and update the PK, then it will return the PK
    - Persist
      * Take an object that isn’t in the db (has a pk 0) save it into the database and update the PK, Returns void
    - Get
      * Get an object from the database by its ID, if that object does not exist, it returns null
    - Load
      * Take an object ID and get a proxy of that object, if it does not exist you get an ObjectNotFoundException
    - Update
      * Take a transient or detached object, search all persistent objects for the same PK as the detached/transient object. If we find one with the same PK, we Throw a NonUniqueObjectException, else we move the object into the persistent state.
    - Merge
      * Take a transient or detached object, search all persistent objects for the same PK as the detached/transient object. If we find one with the same PK, we copy all of the fields of the detached/transient into that persistent object, else we make a new object that is persistent and copy the fields into that.
    - Delete
      * Takes a pk and deletes the object of that pk
* HQL
  + Hibernate query language
  + It looks a lot like sql but it uses our object names and fields instead of db names and fields
  + From Cave where caveId := caveId (
  + From Bear where cave.caveId := caveId (:= for params)
* Object States
  + What are the states of an object in hibernate
    - Transient - Object with a PK of 0 that is not in the database
    - Persistent - Object with a PK matching the DB, in the cache for a current session.
    - Detached - Object with a PK from the database, it is no longer in a cache for a session
  + How can I get an object into each of those states
  + Automatic Dirty Checking
    - Any change I make to a persistent object, Hibernate is aware and will reflect in the database
    - bear.setName(“Something Else”) - hibernate can interpret this and will save it when the session closes
* Caching
  + What is l1 caching?
    - This is Session scoped caching, so every session gets its own cache
    - Cache - a way to store data in a quicker to access format than it normally is. Accessing the database is slow, so we can save data we got from the DB on the server in a cache and accessing the cache on the server is fast.
    - Is built into hibernate, you always get it.
  + How does it relate to object states
    - Any persistent object is in the cache for its session
    - Any data we have retrieved from the DB goes into the cache
    - Any data we have saved goes in the cache
  + What is l2 caching
    - Optional, SessionFactory scoped caching
    - This is one cache for every session made
    - This could potentially give huge speed increases, but becomes very complicated if you ever have more than one server talking to the database
    - Providers, EHCHACHE, JBoss Cache
  + Optimizations
    - We don’t have to save anything until the end of the session
    - Hibernate can use the cache to batch different sql requests together into a single request.
* Proxies
  + What is a proxy?
    - Is a placeholder value that hibernate can resolve in the future if needed
  + When do I get a proxy in Hibernate
    - When you load an object
    - For all Entity fields on an object that don’t have eager initialization ( that means they do have lazy initialization)
  + How do I resolve a proxy
    - Try to use the value
  + Exceptions related?
    - LazyInitializationException
      * If we try to access a proxy, when we are no longer in a session. Because hibernate can’t then resolve the proxy
  + Why are proxies good
    - It allows hibernate to download as little data as possible from the DB, saving us time