CS544 EA

Hibernate

Identity Mapping

Mapping Primary Keys

- Object / Relational mismatch
  - JPA requires you to specify the property that will map to the primary key (best non-primitive)
- **Prefer surrogate keys**
  - Natural keys often lead to a brittle schema

```
@Entity
public class Customer {
      @Id
      private String name;

      ...
}
```

Natural key “name” can give problems

```
@Entity
public class Customer {
      @Id
      private Long id;
      private String name;

      ...
}
```

Instead use “id” as surrogate key

Primary Key

- A primary key is
  - Unique
    - No duplicate values
  - Constant
    - Value never changes
  - Required
    - Value can never be null
- Primary key types:
  - Natural key
    - Has a meaning in the business domain
  - Surrogate key
    - Has no meaning in the business domain
    - Best practice

Generating Identity

- The DB can generate surrogate key values
  - Using **@GeneratedValue**
  - Ensuring identity uniqueness
  - No meaning in business anyway

```
@Entity
public class Customer {
      @Id
      @GeneratedValue
      private Long id;
      private String name;

      ...
}
```

Generation Strategies

No longer seems to work well for MySQL

Best to always specify strategy

- On optional strategy argument
  - Hibernate will guess the best strategy based on the database if strategy is not specified
- Strategy options are:

Value	Description
AUTO (or not specified)	Selects the best strategy for your database
IDENTITY	Use an identity column (MS SQL, MySQL, HSQL, ...)
SEQUENCE	Use a sequence (Oracle, PostgreSQL, SAP DB, ...)
TABLE	Uses a table to hold last generated values for PKs
(no annotation)	Specifies that the value is assigned by the application

Identity Column

- Identity columns **automatically generate** the next ID value
  - Popular Databases: MS-SQL server, MySQL

```
@Entity
public class Customer {
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private Long id;
	private String name;

	...
}
```

Sequence

- A sequence is a **separate DB object** that provides ‘next’ values
  - Can be used as identity source by multiple tables
  - Ensuring unique ID column with unique values even when these tables are combined into a single view (or resultset)
- Popular databases that use sequences:
  - Oracle, PostgreSQL

```
@Entity
public class Customer {
	@Id
	@GeneratedValue(strategy=GenerationType.SEQUENCE)
	private Long id;
	private String name;

	...
}
```

Sequences Names

- Each sequence has its own name
  - Hibernate defaults to “[entity name]_sequence”

```
@Entity
public class Customer {
      @Id
      @GeneratedValue(strategy=GenerationType.SEQUENCE)
      private Long id;
      private String name;

      ...
}
```

Hibernate creates:
customer_sequence

Specifying a Sequence

- Manually specify a sequence
  - Then tell JPA to **use that** one for generation

Specifies that the CUSTOMER_SEQUENCE Sequence exists in the DB

```
@Entity
@SequenceGenerator(name="customer", sequenceName="CUSTOMER_SEQUENCE")
public class Customer {
      @Id
      @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="customer")
      private Long id;
      private String name;

      ...
}
```

Indicates that we should use the customer generator

Table

- JPA can use a Table to emulate a Sequence
  - **Slow** because it requires an additional transaction
  - Sometimes useful on Databases that don't have sequences

```
@Entity
public class Customer {
	@Id
	@GeneratedValue(strategy=GenerationType.TABLE)
	private Long id;
	private String name;

	...
}
```