Documento de Diseño: Gestión de Estudiantes, Cursos y Notas

Objetivo

Desarrollar una aplicación Java utilizando Hibernate para gestionar estudiantes, los cursos en los que están inscritos y las notas obtenidas. Este sistema utiliza relaciones ORM y asegura la integridad referencial de los datos.

Estructura de las Entidades

1. Estudiante

- **Tabla:** estudiantes
- **Atributos:**
  - `id` (Long, Primary Key, autogenerado)
  - `nombre` (String, no nulo)
  - `apellido` (String, no nulo)
  - `email` (String, único)
- **Relaciones:**
  - Many-to-Many con `Curso` (Un estudiante puede inscribirse en varios cursos).

2. Curso

- **Tabla:** cursos
- **Atributos:**
  - `id` (Long, Primary Key, autogenerado)
  - `nombre` (String, no nulo)
  - `descripcion` (String, opcional)
- **Relaciones:**
  - Many-to-Many con `Estudiante`.
  - One-to-Many con `Nota` (Un curso puede tener múltiples notas asociadas).

3. Nota

- **Tabla:** notas
- **Atributos:**
  - `id` (Long, Primary Key, autogenerado)
  - `valor` (Double, no nulo)

    - `fecha` (Date, no nulo)
- **Relaciones:**
    - Many-to-One con `Curso` (Cada nota pertenece a un curso).
    - Many-to-One con `Estudiante` (Cada nota pertenece a un estudiante).

Relaciones ORM

Relación Many-to-Many entre Estudiante y Curso

- **Tabla intermedia:** estudiante_curso
- **Descripción:**
    - Esta tabla tiene dos columnas principales: `estudiante_id` y `curso_id`.
    - Cada fila representa la inscripción de un estudiante en un curso.
- **Anotaciones:**
    - En `Estudiante`:

```
@ManyToMany(mappedBy = "estudiantes")
private Set<Curso> cursos = new HashSet<>();
```

En `Curso`:

```
@ManyToMany(mappedBy = "cursos")
private Set<Estudiante> estudiantes = new HashSet<>();
```

Relación One-to-Many entre Curso y Nota

- **Descripción:**
    - Un curso puede tener muchas notas asociadas.
    - Las notas de un curso dependen de estudiantes específicos inscritos en él.
- **Anotaciones:**
    - En `Curso`:

```
@OneToMany(mappedBy = "curso", cascade = CascadeType.ALL
private List<Nota> notas = new ArrayList<>();
```

En `Nota`:

```
@ManyToOne
@JoinColumn(name = "curso_id", nullable = false)
private Curso curso;
```

Relación Many-to-One entre Nota y Estudiante

- **Descripción:**
  - Una nota pertenece a un estudiante específico.
- **Anotaciones:**
  - En `Nota`:

```
@ManyToOne
@JoinColumn(name = "estudiante_id", nullable = false)
private Estudiante estudiante;
```

Gestión de la Integridad Referencial

Estrategia para las Relaciones

1. **Cascade Operations:**
   - Se configuran cascadas (`CascadeType.PERSIST`, `CascadeType.MERGE`, etc.) para simplificar la propagación de cambios en las entidades relacionadas.
   - Ejemplo: Cuando se guarda un estudiante con cursos asociados, estos también se persisten automáticamente.
2. **Orphan Removal:**
   - Se utiliza `orphanRemoval = true` en relaciones One-to-Many para eliminar correctamente entidades "huérfanas".
3. **Claves Foráneas:**
   - Las claves foráneas (`curso_id`, `estudiante_id`) están configuradas en las anotaciones `@JoinColumn` para garantizar la referencia correcta.

Restricciones en Base de Datos

- Las claves foráneas (`FOREIGN KEY`) aseguran que no se puedan eliminar cursos o estudiantes mientras tengan registros relacionados.
- Configuración de eliminación en cascada en las tablas intermedias para gestionar relaciones Many-to-Many.

Gestión de Excepciones

Errores Comunes Manejados:

1. **`EntityNotFoundException`**:
   - Ocurre cuando se intenta acceder a un registro inexistente.
   - Solución: Se captura esta excepción y se muestra un mensaje descriptivo al usuario.
2. **`ConstraintViolationException`**:
   - Sucede al violar restricciones de la base de datos, como claves únicas o foráneas.
   - Solución: Se captura y se proporciona un mensaje amigable al usuario indicando el problema.
3. **`TransactionException`**:
   - Problemas durante transacciones, como intentos de guardar datos inválidos.
   - Solución: Revertir la transacción y registrar el error para análisis posterior.

Ejemplo de Manejo de Excepciones:

```
try {
    estudianteDAO.save(estudiante);
    System.out.println("Estudiante guardado con éxito.");
} catch (ConstraintViolationException e) {
    System.out.println("⚠ Error: No se pudo guardar el estudia
} catch (Exception e) {
    System.out.println("⚠ Error inesperado: " + e.getMessage()
}
```

Decisiones Importantes

1. **Elección de Hibernate:**
   - Hibernate simplifica la implementación de relaciones complejas entre entidades.
   - Permite abstraer las consultas SQL con HQL (Hibernate Query Language).
2. **Integridad Referencial:**
   - Se priorizó la consistencia de los datos mediante el uso de claves foráneas y restricciones en cascada.
3. **Simplificación del Código:**
   - Uso de DAOs genéricos (`BaseDAO`) para evitar duplicación de lógica CRUD.
4. **Escalabilidad:**
   - La estructura permite agregar nuevas entidades o relaciones con mínimas modificaciones al código existente.