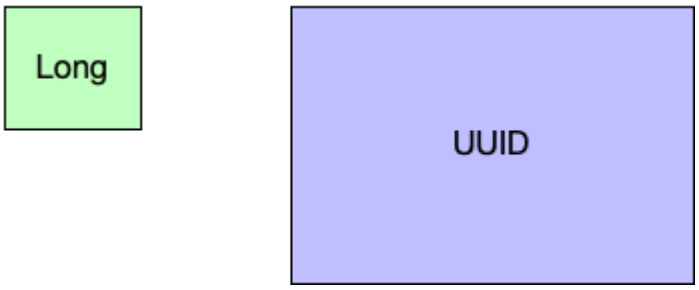
Tabla de Contenidos

Si estás trabajando con JPA y empiezas a diseñar entidades, tarde o temprano aparece la pregunta de siempre: ¿uso un `Long` autoincremental o me paso a `UUID`? En este artículo vamos a verlo desde lo básico, sin meternos en complicaciones innecesarias, con una explicación cercana y un par de ejemplos prácticos para que puedas empezar a usar UUID en tus entidades JPA con confianza.



Qué es un UUID y por qué usarlo en JPA hoy

Un `UUID`, o *Universally Unique Identifier*, es un identificador único universal. En Java lo solemos manejar con la clase `java.util.UUID`, y tiene una pinta como esta: `550e8400-e29b-41d4-a716-446655440000`. La idea principal es que ese valor sea prácticamente único, incluso aunque se genere en distintas máquinas, servicios o bases de datos.

En JPA, usar UUID como identificador puede venir muy bien cuando tu aplicación crece, cuando tienes varios servicios creando datos o cuando quieres evitar depender tanto de un id numérico autoincremental generado por la base de datos. Por ejemplo, en una arquitectura con microservicios, generar el id desde la aplicación puede simplificar bastante las cosas porque no necesitas esperar a que la base de datos te devuelva el identificador.

Eso sí, no significa que UUID sea siempre mejor que `Long`. Un `Long` es más pequeño, más fácil de leer y suele ser muy eficiente para índices. Pero un UUID te da ventajas interesantes: es difícil de adivinar, se puede generar antes de guardar la entidad y funciona muy bien en sistemas distribuidos. Como casi siempre en desarrollo, la respuesta correcta depende del contexto.

Configurar un UUID como id en tu entidad JPA

La forma más básica de usar un UUID en una entidad JPA es declarar el campo `id` como `UUID` y anotarlo con `@Id`. Si estás usando una versión moderna de Jakarta Persistence, puedes usar `GenerationType.UUID`, que permite generar el UUID automáticamente. Es una opción bastante limpia y directa para proyectos nuevos.

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

import java.util.UUID;

@Entity
public class Producto {

    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    private String nombre;

    private Double precio;
```

```
    public UUID getId() {
        return id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public Double getPrecio() {
        return precio;
    }

    public void setPrecio(Double precio) {
        this.precio = precio;
    }
}
```

Si usas Hibernate, también es bastante habitual encontrarte con `@UuidGenerator`, sobre todo en proyectos donde se quiere ser un poco más explícito con el proveedor JPA. Por ejemplo, con Hibernate 6 puedes usarlo así:

```
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import org.hibernate.annotations.UuidGenerator;

import java.util.UUID;

@Entity
```

```
public class Cliente {

    @Id
    @UuidGenerator
    private UUID id;

    private String email;

    private String nombre;

    public UUID getId() {
        return id;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

A nivel de base de datos, el tipo exacto puede variar. En PostgreSQL, por ejemplo, existe el

tipo `UUID`, lo cual es bastante cómodo. En otras bases de datos puede guardarse como `CHAR(36)`, `VARCHAR(36)` o incluso como binario. Para empezar, no hace falta obsesionarse demasiado, pero sí conviene revisar cómo lo está mapeando tu proveedor JPA para evitar sorpresas en producción.

Dos ejemplos simples para guardar UUID con JPA

El primer ejemplo sería el caso más común: dejamos que JPA genere el UUID automáticamente cuando guardamos la entidad. Imagina que tenemos la entidad `Producto` que vimos antes y queremos persistirla desde un servicio. El código podría ser algo así:

```
import jakarta.persistence.EntityManager;
import jakarta.transaction.Transactional;

public class ProductoService {

    private final EntityManager entityManager;

    public ProductoService(EntityManager entityManager) {
        this.entityManager = entityManager;
    }

    @Transactional
    public Producto crearProducto() {
        Producto producto = new Producto();
        producto.setNombre("Teclado mecánico");
        producto.setPrecio(79.99);

        entityManager.persist(producto);

        return producto;
```

```
    }
}
```

En este caso, tú no asignas el `id` manualmente. Creas el objeto, rellenas sus campos y llamas a `persist`. JPA se encarga de generar el UUID y asociarlo a la entidad. Una ventaja práctica de este enfoque es que tu código queda limpio y no tienes que preocuparte por el identificador en la mayoría de casos.

El segundo ejemplo es útil cuando quieres generar tú mismo el UUID antes de guardar la entidad. Esto puede servir si recibes el id desde otro sistema, si quieres crear el identificador antes de llamar a la base de datos o si simplemente prefieres tener control explícito. Para eso, no usamos `@GeneratedValue`, sino que asignamos el UUID manualmente.

```
import jakarta.persistence.Entity;
import jakarta.persistence.Id;

import java.util.UUID;

@Entity
public class Pedido {

    @Id
    private UUID id;

    private String referencia;

    private String estado;

    public Pedido() {
        this.id = UUID.randomUUID();
    }
```

```
    public UUID getId() {
        return id;
    }

    public String getReferencia() {
        return referencia;
    }

    public void setReferencia(String referencia) {
        this.referencia = referencia;
    }

    public String getEstado() {
        return estado;
    }

    public void setEstado(String estado) {
        this.estado = estado;
    }
}
```

Y luego lo podríamos guardar de una forma muy parecida:

```
import jakarta.persistence.EntityManager;
import jakarta.transaction.Transactional;

public class PedidoService {

    private final EntityManager entityManager;

    public PedidoService(EntityManager entityManager) {
        this.entityManager = entityManager;
```

```
    }

    @Transactional
    public Pedido crearPedido() {
        Pedido pedido = new Pedido();
        pedido.setReferencia("PED-2025-001");
        pedido.setEstado("CREADO");

        entityManager.persist(pedido);

        return pedido;
    }
}
```

Usar UUID en JPA no es complicado, pero sí conviene entender qué estás ganando y qué estás sacrificando. Para proyectos modernos, APIs públicas, sistemas distribuidos o entidades que necesitas identificar antes de llegar a la base de datos, puede ser una opción muy cómoda. Si estás empezando, mi recomendación es probar primero con un ejemplo simple, revisar cómo se guarda en tu base de datos y, a partir de ahí, decidir si encaja bien con tu aplicación.

- ¿Que es un UUID?
- JPA Merge y el manejo del EntityManager
- JPA Remove y Objetos gestionados
- Curso Experto Arquitectura y Productividad
- El modelo vista controlador y sus responsabilidades