**Qué tamaño puede tener un number en SQL?**

En Oracle Database, el tipo de datos **NUMBER** tiene una estructura flexible que permite almacenar valores numéricos con precisión y escala variables. Aquí están los detalles clave sobre su capacidad:

**📏 Tamaño y Límites del Tipo NUMBER en Oracle**

| **Parámetro** | **Valor** | **Explicación** |
| --- | --- | --- |
| **Precisión máxima** | 38 dígitos significativos | Máximo total de dígitos (enteros + decimales). |
| **Escala máxima** | De -84 a 127 | Dígitos después del punto decimal. Escala negativa redondea a múltiplos de 10. |
| **Rango de valores** | ±1.0E-130 a ±9.999E125 | Valores extremos admitidos. |
| **Almacenamiento** | 1 a 22 bytes | Depende de la precisión/escala definida. |

**🔍 Ejemplos Prácticos**

**1. Sin especificar precisión/escala:**

sql

NUMBER -- Almacena hasta 38 dígitos, escala variable (ej: 123456.789)

**2. Con precisión y escala fijas:**

sql

NUMBER(10) -- Enteros hasta 10 dígitos (ej: 1234567890)

NUMBER(10,2) -- 8 enteros + 2 decimales (ej: 12345678.99)

NUMBER(5,-2) -- Redondea a centenas (ej: 12300 en lugar de 12345)

**3. Casos comunes en tu esquema:**

sql

-- Para IDs (auto-incrementales)

Id NUMBER GENERATED ALWAYS AS IDENTITY -- Equivale a NUMBER(38)

-- Para prioridades (valores pequeños)

Priority NUMBER(2) DEFAULT 1 -- Admite valores de -99 a 99

-- Para valores monetarios

Price NUMBER(10,2) -- Ej: 99999999.99

**⚠️ Consideraciones Importantes**

1. **Rendimiento**:
   * Usar NUMBER(10) es más eficiente que NUMBER si sabes el límite de dígitos.
   * Para claves primarias/foreign keys, especifica una precisión adecuada (ej: NUMBER(10)).
2. **Alternativas para casos específicos**:
   * **Enteros pequeños**: INT o NUMBER(10) (alias de NUMBER(38) en Oracle).
   * **Decimales científicos**: BINARY\_FLOAT/BINARY\_DOUBLE (menos precisión, pero más rápido).
3. **Comparación con otros motores**:
   * En **MySQL/MariaDB**: INT (4 bytes), BIGINT (8 bytes).
   * En **SQL Server**: INT (4 bytes), DECIMAL(19,0) similar a NUMBER(19).

**📌 Recomendación para tu Esquema**

En tus tablas actuales:

sql

-- Mejora para IDs (si esperas menos de 10^9 registros)

Id NUMBER(10) GENERATED ALWAYS AS IDENTITY PRIMARY KEY

-- Prioridad puede ser incluso NUMBER(1) si solo usas valores 1-9

Priority NUMBER(1) DEFAULT 1 CHECK (Priority BETWEEN 1 AND 9)

**💡 Dato Curioso**

Oracle almacena los NUMBER en formato **científico comprimido**:

* El número 123.45 se guarda como 1.2345 x 10^2.
* Por eso soporta rangos tan amplios sin consumir siempre 38 dígitos de espacio.

¿Necesitas ayuda para ajustar los tipos de datos en otro contexto? 😊

This response is AI-generated, for reference only.

Para poner un tamaño a los id que fuese entero pero lo más grande posible que le pongo

Para definir un campo de tipo **entero** con el **tamaño máximo posible** en Oracle Database (ideal para IDs autoincrementales o claves primarias), usa la siguiente sintaxis:

**🔢 Opción Recomendada (Máximo soportado)**

sql

ID NUMBER(38) GENERATED ALWAYS AS IDENTITY PRIMARY KEY

**Explicación:**

* **NUMBER(38)**:
  + Es el **máximo valor entero** que Oracle puede almacenar en este tipo de dato.
  + **Rango**: Desde -10^38 + 1 hasta 10^38 - 1 (es decir, números de **hasta 38 dígitos**).
  + Equivale a **1.0E+37** (un 1 seguido de 37 ceros).

**📌 Alternativas (equivalentes)**

Si prefieres usar alias semánticos (Oracle los convierte internamente a NUMBER(38)):

sql

ID INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY -- Recomendado para legibilidad

-- o

ID INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY

**⚠️ Consideraciones importantes**

1. **Rendimiento**:
   * Aunque NUMBER(38) permite valores enormes, en la práctica es raro necesitar más de NUMBER(19) (equivalente a **BIGINT** en otros motores).
   * Para bases de datos con **billones de registros**, NUMBER(19) es más que suficiente (hasta 9,223,372,036,854,775,807).
2. **Espacio en disco**:
   * Oracle almacena los números de forma variable, pero un NUMBER(38) puede ocupar hasta **22 bytes**.
   * Si usas NUMBER(10), ocupa solo **5 bytes**.
3. **Autoincremento seguro**:
   * Con GENERATED ALWAYS AS IDENTITY, Oracle gestiona automáticamente los valores sin riesgo de desbordamiento.

**💡 Ejemplo completo en tus tablas**

sql

CREATE TABLE Task (

Id NUMBER(38) GENERATED ALWAYS AS IDENTITY PRIMARY KEY, -- Máximo tamaño

-- Resto de campos...

);

-- Alternativa con alias (misma capacidad)

CREATE TABLE User (

Id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

-- Resto de campos...

);

**🔍 ¿Cuándo usar NUMBER(38) vs NUMBER(19)?**

| **Tipo** | **Rango** | **Uso típico** |
| --- | --- | --- |
| **NUMBER(38)** | ±1E+38 (máximo teórico) | Sistemas legacy o requerimientos excepcionales. |
| **NUMBER(19)** | ±9,223,372,036,854,775,807 (9.2E+18) | Bases de datos masivas modernas (99.9% de casos). |

**🚀 Conclusión**

Para IDs en tus tablas, usa:

sql

Id NUMBER(38) GENERATED ALWAYS AS IDENTITY PRIMARY KEY -- Si quieres el máximo absoluto

-- o

Id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY -- Mejor legibilidad, mismo rango

¿Necesitas ayuda para ajustar esto en un contexto específico de tu aplicación? 😊

**Qué tamaño puede tener un Varchar como máximo y como mínimo?**

En Oracle Database, el tipo de datos **VARCHAR2** (recomendado sobre el obsoleto VARCHAR) tiene los siguientes límites de tamaño:

### 📏 ****Límites de**** VARCHAR2 ****en Oracle****

| **Parámetro** | **Valor (hasta Oracle 12c+)** | **Notas** |
| --- | --- | --- |
| **Máximo estándar** | **4000 bytes** | Límite tradicional en versiones antiguas. |
| **Máximo extendido** | **32767 bytes** (32 KB) | Requiere configuración especial (\*). |
| **Mínimo** | **1 byte** | Cadena vacía ('') ocupa 1 byte. |

### 🔧 ****Configuración para habilitar VARCHAR2 extendido (32 KB)****

1. **A nivel de sesión** (temporal):

sql

 ALTER SYSTEM SET MAX\_STRING\_SIZE = EXTENDED SCOPE = SPFILE;

 **Reinicia la base de datos**:

bash

 SHUTDOWN IMMEDIATE;

STARTUP UPGRADE;

 **Ejecuta el script de migración**:

bash

 @?/rdbms/admin/utl32k.sql

 **Reinicia nuevamente**:

bash

1. SHUTDOWN IMMEDIATE;
2. STARTUP;

### 📌 ****Recomendaciones prácticas****

1. **Para campos cortos** (nombres, emails, códigos):

sql

 VARCHAR2(255) -- Suficiente para la mayoría de casos.

 **Para campos largos** (descripciones, JSON pequeños):

sql

 VARCHAR2(4000) -- Límite estándar sin configuración especial.

 **Para texto muy largo** (documentos, logs):

sql

1. CLOB -- Admite hasta 128 TB (teóricamente).

### ⚠️ ****Diferencias clave****

* **VARCHAR2 vs CHAR**:
  + VARCHAR2 es de longitud variable (ahorra espacio).
  + CHAR rellena con espacios hasta el tamaño definido.
* **VARCHAR2 vs CLOB**:
  + Usa VARCHAR2 para textos < 4000 bytes (mejor rendimiento).
  + Usa CLOB para contenido mayor.

### 💡 ****Ejemplo en tu esquema****

sql

CREATE TABLE User (

Id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

Name VARCHAR2(255) NOT NULL, -- Nombre típico

Email VARCHAR2(320) UNIQUE NOT NULL, -- Máximo según RFC 5321

Password VARCHAR2(60) NOT NULL, -- Suficiente para hash BCrypt

Bio VARCHAR2(4000) -- Descripción larga (estándar)

);

### 🌐 ****Comparativa con otros motores****

| **Base de Datos** | **Tipo Equivalente** | **Máximo** |
| --- | --- | --- |
| **MySQL** | VARCHAR | 65535 bytes |
| **SQL Server** | VARCHAR | 8000 bytes |
| **PostgreSQL** | VARCHAR | 1 GB |

¿Necesitas ayuda para decidir entre VARCHAR2 y CLOB en un caso concreto? 😊

This response is AI-generated, for reference only.

**big int cual sería el número más grande que puede almacenar?**

En Oracle Database, el equivalente a **BIGINT** de otros motores (como MySQL o PostgreSQL) se implementa mediante el tipo NUMBER con precisión específica. Aquí están los detalles clave:

### 🔢 ****Equivalente a BIGINT en Oracle****

#### ****Definición Recomendada****:

sql

NUMBER(19) -- Rango: -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807

* **Este es el estándar** para almacenar enteros grandes (equivalente a un BIGINT en otros sistemas).
* Ocupa **8 bytes** de almacenamiento (igual que en MySQL/SQL Server).

### 📏 ****Límites Exactos****

| **Tipo** | **Valor Mínimo** | **Valor Máximo** | **Equivalente en Otros Motores** |
| --- | --- | --- | --- |
| **NUMBER(19)** | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 | BIGINT (MySQL), INT8 (PostgreSQL) |
| **INTEGER** | -2,147,483,648 | 2,147,483,647 | INT (estándar) |

### 💡 ****¿Por qué NUMBER(19) y no NUMBER(38)?****

* **NUMBER(19)** es suficiente para:
  + 9.2 trillones de registros (¡más que suficiente para 99.9% de aplicaciones!).
  + Compatibilidad con sistemas externos que usan BIGINT (ej: APIs, aplicaciones).
* **NUMBER(38)** permite hasta 38 dígitos, pero:
  + Consume más espacio (hasta 22 bytes).
  + Es excesivo para la mayoría de casos reales.

### 🛠️ ****Ejemplo Práctico****

Para IDs autoincrementales grandes:

sql

CREATE TABLE mega\_table (

id NUMBER(19) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

data VARCHAR2(4000)

);

### ⚠️ ****Notas Importantes****

1. **Oracle no tiene un tipo BIGINT nativo**, pero NUMBER(19) es el equivalente funcional.
2. **Para claves primarias**, es mejor usar NUMBER(19) que NUMBER(38) por rendimiento.
3. **Si usas herramientas de migración**, pueden convertir automáticamente BIGINT de otros motores a NUMBER(19) en Oracle.

### 🌐 ****Comparativa con Otros Motores****

| **Base de Datos** | **Tipo BIGINT** | **Máximo Valor Positivo** |
| --- | --- | --- |
| **Oracle** | NUMBER(19) | 9,223,372,036,854,775,807 |
| **MySQL** | BIGINT | 9,223,372,036,854,775,807 |
| **PostgreSQL** | BIGINT/INT8 | 9,223,372,036,854,775,807 |
| **SQL Server** | BIGINT | 9,223,372,036,854,775,807 |

¿Necesitas ayuda para ajustar esto en un escenario específico de tu aplicación? 😊

This response is AI-generated, for reference only.

**bueno pero si el número máximo es 9,223,372,036,854,775,807 number 19 llegaría hasta 9999999999999999999 con lo cual habría números que se quedan fuera, no?**

¡Excelente observación! Tienes toda la razón. Vamos a aclarar este punto importante:

### 🔍 ****Explicación Detallada sobre NUMBER(19) vs. BIGINT****

#### ****1. El "pequeño" error conceptual****

* Efectivamente, **NUMBER(19) puede almacenar hasta 999,999,999,999,999,9999** (un 9 seguido de 18 nueves), que es **mayor** que el límite de 9,223,372,036,854,775,807 de un BIGINT estándar.
* El valor 9,223,372,036,854,775,807 en realidad corresponde al máximo de un **entero con signo de 64 bits** (como el BIGINT de otros motores), que usa **bit 63 para el signo**.

#### ****2. ¿Por qué se recomienda NUMBER(19) como equivalente?****

Oracle no tiene un tipo BIGINT binario nativo, pero:

* **NUMBER(19) cubre todo el rango de BIGINT** y además permite valores más grandes.
* Es **compatible binariamente** con aplicaciones que esperan un BIGINT (siempre que no superes 2^63 - 1).

#### ****3. Tabla Comparativa Exacta****

| **Tipo** | **Máximo con Signo** | **Máximo sin Signo** | **Notas** |
| --- | --- | --- | --- |
| **BIGINT (otros DB)** | 9,223,372,036,854,775,807 | No aplica (con signo) | Usa complemento a 2 |
| **NUMBER(19)** | 999,999,999,999,999,9999 | Igual (Oracle maneja signo aparte) | Decimal exacto |
| **NUMBER(19,0)** | 999,999,999,999,999,9999 | Igual | Equivalente a NUMBER(19) |

### ✅ ****Recomendación Definitiva****

1. **Si necesitas compatibilidad exacta con BIGINT** (ej: migrar desde MySQL):

sql

 NUMBER(19) CHECK (column\_name BETWEEN -9223372036854775808 AND 9223372036854775807)

 **Si solo quieres el máximo rendimiento para IDs**:

sql

1. NUMBER(19) -- Aprovechas el espacio extra sin restricciones

### ⚠️ ****¿Cuándo usar cada uno?****

* **Usa la restricción CHECK** si intercambias datos con sistemas que usen BIGINT de 64 bits.
* **Usa NUMBER(19) simple** si es solo para IDs internos y quieres valores más grandes.

### 💡 ****Ejemplo Práctico****

sql

-- Para compatibilidad estricta con BIGINT

CREATE TABLE transacciones (

transaccion\_id NUMBER(19)

CHECK (transaccion\_id BETWEEN -9223372036854775808 AND 9223372036854775807),

monto NUMBER(19,4)

);

-- Para IDs internos (máximo espacio)

CREATE TABLE logs (

log\_id NUMBER(19) GENERATED ALWAYS AS IDENTITY, -- Hasta 9.9E+18

detalle CLOB

);

### 🌟 ****Dato Curioso****

El valor 9,223,372,036,854,775,807 es **2^63 - 1**, usado en:

* El sistema de archivos **ZFS** (tamaño máximo de archivo).
* **Timestamps** en algunos lenguajes (como el "Year 2038 problem" en sistemas de 32 bits).

¿Necesitas ayuda para implementar esto en tu esquema de base de datos? 😊

This response is AI-generated, for reference only.

**y no sería mejor Number 18 para estar 100% seguros de que no habrá problemas?**

¡Buena pregunta! Veamos por qué **NUMBER(18) no sería la mejor opción** y cuál es la alternativa más segura para garantizar compatibilidad con BIGINT:

### 🔍 ****¿Por qué NUMBER(18) NO es ideal?****

1. **Límite demasiado ajustado**:
   * NUMBER(18): Máximo = **999,999,999,999,999,999** (9.99E+17)
   * **BIGINT real (8 bytes)**: Máximo = 9,223,372,036,854,775,807 (≈9.22E+18)
   * ¡Te quedarías **sin 922 trillones de valores** comparado con BIGINT!
2. **Problemas de compatibilidad**:
   * Si migras datos desde otros sistemas (MySQL, PostgreSQL, etc.), podrías truncar valores BIGINT válidos.

### ✅ ****Solución 100% Segura: NUMBER(19) con CHECK****

Para garantizar compatibilidad exacta con BIGINT **sin perder valores válidos**:

sql

CREATE TABLE ejemplo (

id NUMBER(19)

CHECK (id BETWEEN -9223372036854775808 AND 9223372036854775807),

-- Resto de campos...

);

#### Beneficios:

* **Mantienes el rango completo** de BIGINT (2^63).
* **Oracle optimiza el almacenamiento**: Aunque defines NUMBER(19), solo usará los bytes necesarios para cada valor.
* **Compatibilidad garantizada** con aplicaciones que esperan un entero de 64 bits.

### 📊 ****Tabla Comparativa****

| **Tipo** | **Máximo Positivo** | **¿Cubre BIGINT?** | **Almacenamiento** |
| --- | --- | --- | --- |
| NUMBER(18) | 999,999,999,999,999,999 | ❌ No | ~8 bytes |
| NUMBER(19) | 9,999,999,999,999,999,999 | ✅ Sí (con CHECK) | 8-10 bytes |
| NUMBER(19,0) | Igual que NUMBER(19) | ✅ Sí | 8-10 bytes |

### 💡 ****Recomendación Final****

* **Usa NUMBER(19)** si:
  + Quieres máximos compatibilidad y rango.
  + No te preocupa superar el límite de BIGINT (9.22E+18).
* **Añade CHECK** solo si:
  + Intercambias datos con sistemas externos que usen BIGINT estricto.
  + Quieres evitar valores superiores a 2^63 por diseño.

### 🛠️ ****Ejemplo Práctico****

sql

-- Opción 1: Máximo rango sin restricciones

CREATE TABLE usuarios (

user\_id NUMBER(19) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

nombre VARCHAR2(100)

);

-- Opción 2: Compatibilidad estricta con BIGINT

CREATE TABLE transacciones (

tx\_id NUMBER(19)

CHECK (tx\_id BETWEEN -9223372036854775808 AND 9223372036854775807),

monto NUMBER(18,2)

);

### ⚠️ ****¿Y si necesitas valores mayores?****

Si realmente necesitas superar el límite de BIGINT (ej: para claves astronómicas), usa:

sql

NUMBER(38) -- Hasta 1E+38 (pero con mayor sobrecarga de almacenamiento)

¿Te preocupa algún escenario específico donde esto podría fallar? ¡Déjame saber para ajustar la solución! 😊

**Como protejo el campo Password para que ni si quiera yo pueda verlo en mi BBDD. Tengo que tocar algo en los ajustes de la bbdd o lo haré más adelante en el código, para el caso de oracle?**

Para proteger los passwords en tu base de datos Oracle de forma que **ni siquiera tú (como DBA) puedas verlos**, debes implementar un enfoque de seguridad multicapa. Aquí está la estrategia completa:

### 🔒 ****Solución Profesional para Passwords en Oracle****

#### ****1. Hash + Salt (En el Código de la Aplicación)****

* **Nunca almacenes contraseñas en texto plano**.
* Usa algoritmos fuertes como **PBKDF2, bcrypt o Argon2** (NestJS/Angular lo implementan):

typescript

* // Ejemplo en NestJS (antes de guardar)
* import \* as bcrypt from 'bcrypt';
* const salt = await bcrypt.genSalt();
* const hashedPassword = await bcrypt.hash(plainPassword, salt);

Almacena solo hashedPassword en la base de datos.

#### ****2. Cifrado Transparente (TDE) - Nivel BD****

* **Activa Transparent Data Encryption** para la columna:

sql

* CREATE TABLE "User" (
* Id NUMBER GENERATED ALWAYS AS IDENTITY,
* Password VARCHAR2(200) ENCRYPT USING 'AES256' NO SALT, -- Oracle 21c+
* ...
* );
  + Requiere configuración previa del **Wallet de Oracle**.
  + **Ventaja**: El dato se cifra en disco, pero es transparente para consultas válidas.

#### ****3. Enmascaramiento Dinámico (Opcional)****

* Para que ni siquiera los DBA vean valores reales:

sql

* ALTER TABLE "User" MODIFY (
* Password VARCHAR2(200) MASK FUNCTION 'ocultar\_password()'
* );

Crea la función de enmascaramiento.

#### ****4. Restricción de Acceso Directo****

* **Revoca permisos** de lectura directa:

sql

* REVOKE SELECT ON "User" FROM PUBLIC;
* GRANT SELECT (Id, Name, Email) ON "User" TO roles\_seguros;
  + Usa Vistas o Procedimientos para acceder controladamente.

### 🛡️ ****Protección Adicional****

* **Auditoría**:

sql

 AUDIT SELECT, UPDATE ON "User" BY ACCESS;

 **Política de Contraseñas**:

sql

* CREATE PROFILE app\_profile LIMIT
* PASSWORD\_LIFE\_TIME 90
* FAILED\_LOGIN\_ATTEMPTS 5;
* ALTER USER a3tt PROFILE app\_profile;

### 📌 ****¿Dónde Implementar Cada Capa?****

| **Capa** | **Tecnología** | **Responsable** |
| --- | --- | --- |
| Hash + Salt | Código (NestJS/Angular) | Devs |
| Cifrado (TDE) | Oracle | DBA |
| Enmascaramiento | Oracle | DBA |
| Permisos | Oracle | DBA/DevOps |

### ⚠️ ****Errores Comunes que Evitar****

1. **Usar MD5/SHA-1**: Son obsoletos para contraseñas.
2. **Cifrar (en lugar de hashear)**: El cifrado puede revertirse.
3. **No usar salt**: Permite ataques con rainbow tables.

### 💡 ****Ejemplo Completo****

sql

-- Tabla (Oracle 21c+)

CREATE TABLE "User" (

Id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

Email VARCHAR2(255) UNIQUE,

PasswordHash VARCHAR2(200) ENCRYPT USING 'AES256' NO SALT

);

-- Código NestJS (TypeORM)

@Entity()

export class User {

@Column({ name: 'PasswordHash', length: 200 })

passwordHash: string;

async setPassword(password: string) {

this.passwordHash = await bcrypt.hash(password, 10);

}

}

### 🔐 ****Recomendación Final****

* **Lo crítico es el hashing en la aplicación** (aunque no uses TDE o enmascaramiento).
* **Oracle Advanced Security** (TDE) es útil para cumplir normas como GDPR, pero requiere licencia Enterprise.

¿Necesitas ayuda para configurar el wallet de cifrado o implementar bcrypt en NestJS? 😊

This response is AI-generated, for reference only.

**Estoy creando la tabla desde la ventana de oracle sql developer y no veo nada para encriptar, donde debería estar?**

En Oracle SQL Developer, la opción para cifrar columnas directamente en la interfaz gráfica está **oculta en versiones estándar**, pero puedes habilitarla o usar comandos SQL manuales. Aquí te explico cómo hacerlo:

### 🔍 ****Opción 1: Usar la Interfaz Gráfica (Si está disponible)****

1. **Crea la tabla normalmente** en SQL Developer:
   * Haz clic derecho en **"Tablas"** → **"Nueva Tabla"**.
   * Define las columnas (ID, EMAIL, etc.).
2. **Para la columna PASSWORD**:
   * En la pestaña **"Columnas"**, selecciona la columna.
   * Busca la pestaña **"Opciones Avanzadas"** o **"Seguridad"** (depende de la versión).
   * Marca **"Encrypt"** y elige el algoritmo (ej: AES256).

(Nota: Esta opción solo aparece si tienes Oracle Advanced Security habilitado y usando versión Enterprise).

### 🔧 ****Opción 2: Comandos SQL Manuales (Recomendado para Oracle XE)****

Si no ves la opción gráfica, ejecuta esto en una ventana SQL:

sql

-- 1. Asegúrate de tener el wallet de cifrado configurado (como SYSDBA):

ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "TuPassword123";

-- 2. Crea la tabla con cifrado transparente (TDE):

CREATE TABLE "User" (

Id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

Email VARCHAR2(255) UNIQUE NOT NULL,

Password VARCHAR2(200) ENCRYPT USING 'AES256' NO SALT, -- ¡Cifrado aquí!

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- 3. Verifica el cifrado:

SELECT column\_name, encryption\_alg

FROM all\_encrypted\_columns

WHERE table\_name = 'User';

### ⚠️ ****Limitaciones en Oracle Express Edition (XE)****

* **Oracle XE no incluye TDE (Transparent Data Encryption)**. En este caso:
  1. **Usa hashing en la aplicación** (NestJS/Angular) como explicamos antes.
  2. **Alternativa**: Cifra manualmente con DBMS\_CRYPTO (requiere código PL/SQL).

Ejemplo con DBMS\_CRYPTO (para XE):

sql

-- Crear función de cifrado (ejecutar como SYSDBA)

CREATE OR REPLACE FUNCTION encrypt\_password(p\_password IN VARCHAR2)

RETURN RAW IS

l\_key RAW(32) := UTL\_I18N.STRING\_TO\_RAW('MiClaveSecreta32Bytes123456789012', 'AL32UTF8');

l\_encrypted RAW(2000);

BEGIN

l\_encrypted := DBMS\_CRYPTO.ENCRYPT(

src => UTL\_I18N.STRING\_TO\_RAW(p\_password, 'AL32UTF8'),

typ => DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,

key => l\_key

);

RETURN l\_encrypted;

END;

/

-- Uso en INSERTs

INSERT INTO "User" (Email, Password)

VALUES ('user@example.com', encrypt\_password('contraseña'));

### 📌 ****¿Dónde Encontrar las Opciones en SQL Developer?****

1. **Para TDE (Enterprise)**:
   * Herramientas → Administración de Bases de Datos → Cifrado Transparente.
2. **Para DBMS\_CRYPTO (XE)**:
   * Ejecuta los scripts manualmente en una ventana SQL.

### 💡 ****Consejo Clave****

* **La mejor práctica es combinar**:
  1. **Hashing en la app** (bcrypt/PBKDF2).
  2. **Cifrado en BD** (TDE o DBMS\_CRYPTO) solo si es estrictamente necesario.

Si no ves las opciones de cifrado en tu SQL Developer, confírmame:

1. ¿Usas **Oracle Enterprise o Express Edition (XE)**?
2. ¿Tienes permisos de SYSDBA para configurar el wallet?

¡Con eso puedo darte instrucciones más precisas! 😊

This response is AI-generated, for reference only.

**Vale entonces simplemente me aseguraré de poner medidas una vez me meta en el código.**

¡Exacto! **Enfocarse en el código de la aplicación (NestJS/Angular) es la estrategia más segura y práctica**, especialmente si usas Oracle XE (que no soporta cifrado nativo avanzado). Aquí tienes el plan de acción resumido:

### 🔒 ****Protección de Contraseñas en el Código (NestJS/Angular)****

1. **Hashing con Bcrypt** (Recomendado):

typescript

 // En tu servicio de usuarios (NestJS)

import \* as bcrypt from 'bcrypt';

async createUser(email: string, plainPassword: string) {

const salt = await bcrypt.genSalt(10); // 10 rondas de hashing

const hashedPassword = await bcrypt.hash(plainPassword, salt);

await this.userRepository.save({

email,

password: hashedPassword, // Almacena solo el hash

});

}

 **Verificación al iniciar sesión**:

typescript

1. async validateUser(email: string, plainPassword: string) {
2. const user = await this.userRepository.findOne({ email });
3. if (user && await bcrypt.compare(plainPassword, user.password)) {
4. return user; // Contraseña válida
5. }
6. return null; // Inválido
7. }

### 📌 ****Qué Almacenar en Oracle****

| **Campo** | **Tipo** | **Ejemplo Valor Hash (bcrypt)** |
| --- | --- | --- |
| EMAIL | VARCHAR2(255) | usuario@example.com |
| PASSWORD | VARCHAR2(60) | $2a$10$N9qo8uLOickgx2ZMRZoMy... (60 chars) |

### ⚠️ ****Errores Comunes que Evitar****

* **No uses** algoritmos débiles como MD5/SHA-1.
* **No almacenes** el salt por separado (bcrypt lo incluye en el hash).
* **No cifres** contraseñas (el hashing es irreversible; el cifrado no).

### 🛡️ ****Protección Adicional (Opcional pero Recomendado)****

1. **Protege las consultas SQL**:
   * Usa **TypeORM** con parámetros preparados para evitar inyecciones.

typescript

1. // Correcto (seguro)
2. this.userRepository.findOne({ email });
3. **Auditoría**:
   * Registra intentos fallidos de inicio de sesión.

### 🔍 ****¿Por qué es suficiente el hashing en el código?****

* **Bcrypt** está diseñado para ser:
  + **Lento computacionalmente** (dificulta ataques por fuerza bruta).
  + **Resistente a rainbow tables** (usa salt automático).
* Oracle solo guarda el hash **sin poder revertirlo**.

### 💡 ****Ejemplo de Entidad TypeORM****

typescript

import { Entity, Column, PrimaryGeneratedColumn } from 'typeorm';

@Entity()

export class User {

@PrimaryGeneratedColumn()

id: number;

@Column({ length: 255, unique: true })

email: string;

@Column({ length: 60 }) // Longitud para hash bcrypt

password: string;

}

### 📚 ****Recursos Clave****

* [Bcrypt npm](https://www.npmjs.com/package/bcrypt)
* [OWASP Password Storage](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html)

Con este enfoque, **ni tú ni ningún DBA podrán ver las contraseñas reales**, incluso si acceden directamente a la base de datos. ¡Es la forma más robusta y estándar en la industria! 😊

¿Necesitas ayuda para implementar el servicio de autenticación en NestJS?

**CREATE TABLE USERS ( ID NUMBER(19) GENERATED ALWAYS AS IDENTITY INCREMENT BY 1 START WITH 0 NOT NULL , NAME VARCHAR2(30) NOT NULL , EMAIL VARCHAR2(255) NOT NULL , PASSWORD VARCHAR2(255) NOT NULL , CONSTRAINT USERS\_PK PRIMARY KEY ( ID ) ENABLE ) Start With No puede ser menor que MINVALUE que significa este error?**

El error **"START WITH no puede ser menor que MINVALUE"** ocurre porque estás intentando iniciar una secuencia de identidad con un valor (START WITH 0) que es inferior al valor mínimo permitido por defecto en Oracle (que es **1** para columnas de identidad). Aquí está la explicación y solución detallada:

### 🔍 ****Causa del Error****

1. **Las columnas GENERATED ALWAYS AS IDENTITY** en Oracle tienen un **valor mínimo implícito de 1**.
2. Al definir START WITH 0, violas esta restricción automática.

### ✅ ****Solución (Elige una opción)****

#### ****Opción 1: Usar**** START WITH 1 ****(Recomendado)****

sql

CREATE TABLE USERS

(

ID NUMBER(19) GENERATED ALWAYS AS IDENTITY INCREMENT BY 1 START WITH 1 NOT NULL, -- Cambiado a 1

NAME VARCHAR2(30) NOT NULL,

EMAIL VARCHAR2(255) NOT NULL,

PASSWORD VARCHAR2(255) NOT NULL,

CONSTRAINT USERS\_PK PRIMARY KEY (ID) ENABLE

);

#### ****Opción 2: Definir manualmente el rango (Si necesitas valores desde 0)****

sql

CREATE TABLE USERS

(

ID NUMBER(19) GENERATED ALWAYS AS IDENTITY

INCREMENT BY 1

START WITH 0 -- Ahora sí permitido

MINVALUE 0 -- Estableces el mínimo manualmente

NOT NULL,

NAME VARCHAR2(30) NOT NULL,

EMAIL VARCHAR2(255) NOT NULL,

PASSWORD VARCHAR2(255) NOT NULL,

CONSTRAINT USERS\_PK PRIMARY KEY (ID) ENABLE

);

### 📌 ****¿Por qué Oracle no permite START WITH 0 por defecto?****

* Oracle sigue el estándar SQL que asume que las claves primarias son **valores positivos**.
* El valor 0 podría confundirse con un valor "no inicializado" en algunos contextos.

### 💡 ****Consejos Adicionales****

1. **Para IDs autoincrementales**, es mejor comenzar desde 1 (estándar en bases de datos).
2. **Si necesitas valores negativos**, debes definir explícitamente:

sql

 MINVALUE -9999999999999999999

MAXVALUE 9999999999999999999

 **Verifica los valores actuales** de una secuencia:

sql

1. SELECT sequence\_name, min\_value, max\_value
2. FROM user\_sequences;

### ⚠️ ****Errores Relacionados****

Si ves **"START WITH no puede ser mayor que MAXVALUE"**, aplica la misma lógica:

sql

START WITH 1000

MAXVALUE 9999999999999999999

### Ejemplo Completo con Rango Personalizado

sql

CREATE TABLE USERS

(

ID NUMBER(19) GENERATED ALWAYS AS IDENTITY

INCREMENT BY 1

START WITH 0

MINVALUE 0

MAXVALUE 9999999999999999999

NOT NULL,

-- Resto de campos...

);

¿Necesitas que los IDs comiencen desde 0 por algún requisito específico de tu aplicación? 😊

**Quiero que mi USER\_ID sea clave foránea. Usando el menú de oracle sql developer, no me ha quedadoe muy claro si eso tengo que configurarlo en índices o restricciones, me puedes explicar como hacerlo?**

En Oracle SQL Developer, las **claves foráneas (foreign keys)** se configuran en la sección de **"Restricciones" (Constraints)**, no en Índices. Aquí te explico cómo hacerlo paso a paso desde la interfaz gráfica:

### 🔑 ****Pasos para Crear una Clave Foránea (FK) en SQL Developer****

#### ****1. Abre el diálogo de la tabla****

* Haz clic derecho sobre la tabla que tendrá la FK (ej: TASKS) → **"Editar"**.

#### ****2. Ve a la pestaña "Restricciones"****

* Selecciona la pestaña **"Restricciones"** en el editor de la tabla.

#### ****3. Crea una nueva restricción****

* Haz clic en el botón **"Añadir Restricción"** (icono +).
* Configura los campos:
  + **Nombre**: TASKS\_USER\_FK (usa un nombre descriptivo).
  + **Tipo**: Selecciona **"Clave Foránea"** (Foreign Key).
  + **Tabla de referencia**: Elige la tabla USERS.
  + **Columna local**: Selecciona USER\_ID (en tu tabla actual).
  + **Columna referenciada**: Selecciona ID (de la tabla USERS).

#### ****4. Opciones avanzadas (opcional)****

* **On Delete**: Elige qué hacer si se borra un usuario:
  + CASCADE: Borra tareas asociadas.
  + SET NULL: Pone USER\_ID en NULL (si la columna lo permite).
  + RESTRICT (por defecto): Impide borrar usuarios con tareas.

#### ****5. Aplica los cambios****

* Haz clic en **"Aceptar"** y luego en **"Guardar"**.

### 📜 ****Código SQL Generado (para referencia)****

SQL Developer generará un comando como este:

sql

ALTER TABLE TASKS

ADD CONSTRAINT TASKS\_USER\_FK

FOREIGN KEY (USER\_ID)

REFERENCES USERS(ID)

ON DELETE RESTRICT; -- O CASCADE/SET NULL

### 🖼️ ****Captura de Pantalla (Ubicación en SQL Developer)****

<https://i.imgur.com/JyQ5b3p.png>  
(La ubicación exacta puede variar según versión, pero busca "Constraints" o "Restricciones").

### 💡 ****Consejos Clave****

1. **Nomenclatura**: Usa nombres descriptivos como [TABLA]\_[COLUMNA]\_FK.
2. **Índices automáticos**: Oracle crea un índice para FKs automáticamente, pero puedes añadir uno manualmente si mejora el rendimiento.
3. **Verificación**: Tras crearla, ejecuta una consulta para confirmar:

sql

1. SELECT \* FROM USER\_CONSTRAINTS
2. WHERE TABLE\_NAME = 'TASKS' AND CONSTRAINT\_TYPE = 'R';

### ⚠️ ****Errores Comunes****

* **ORA-02270**: La columna referenciada (ID en USERS) debe ser **PRIMARY KEY o UNIQUE**.
* **ORA-02291**: Violación de integridad (insertas un USER\_ID que no existe en USERS).

### 🔍 ****¿Índices o Restricciones?****

* **Restricciones**: Definen reglas de integridad (PK, FK, CHECK).
* **Índices**: Mejoran el rendimiento de búsquedas (se crean automáticamente para PKs, pero son opcionales para FKs).

¿Necesitas ayuda para crear una FK específica o resolver algún mensaje de error? 😊

**CREATE TABLE TASK ( ID NUMBER(19) GENERATED ALWAYS AS IDENTITY INCREMENT BY 1 START WITH 1 NOT NULL , NAME VARCHAR2(255) DEFAULT Inserta Nombre , DESCRIPTION VARCHAR2(4000) DEFAULT Inserta Descripcion , STATE NUMBER(1) DEFAULT 0 NOT NULL , USER\_ID NUMBER(19) NOT NULL , CATEGORY\_ID NUMBER(19) NOT NULL , CONSTRAINT TASK\_PK PRIMARY KEY ( ID ) ENABLE ) Se ha encontrado un error: ORA-00907: falta el paréntesis derecho. Fallo de la sentencia SQL, Qué le pasa?**

El error **ORA-00907: falta el paréntesis derecho** ocurre porque hay un problema de sintaxis en tu sentencia CREATE TABLE. En tu caso, hay dos errores clave:

**Problemas en tu SQL:**

1. **Los valores DEFAULT deben estar entre comillas** (ya que son cadenas de texto).
   * ❌ DEFAULT Inserta Nombre → ✅ DEFAULT 'Inserta Nombre'.
   * ❌ DEFAULT Inserta Descripcion → ✅ DEFAULT 'Inserta Descripcion'.
2. **Falta una coma** después de la definición de CATEGORY\_ID NUMBER(19) NOT NULL antes de la cláusula CONSTRAINT.

**Sentencia corregida:**

sql

CREATE TABLE TASK

(

ID NUMBER(19) GENERATED ALWAYS AS IDENTITY INCREMENT BY 1 START WITH 1 NOT NULL,

NAME VARCHAR2(255) DEFAULT 'Inserta Nombre',

DESCRIPTION VARCHAR2(4000) DEFAULT 'Inserta Descripcion',

STATE NUMBER(1) DEFAULT 0 NOT NULL,

USER\_ID NUMBER(19) NOT NULL,

CATEGORY\_ID NUMBER(19) NOT NULL,

CONSTRAINT TASK\_PK PRIMARY KEY (ID) ENABLE

);

**Detalles de los cambios:**

1. **Comillas en DEFAULT**:
   * DEFAULT Inserta Nombre → DEFAULT 'Inserta Nombre'.
   * Oracle interpreta Inserta Nombre como un objeto (ej: columna o función), no como texto. Las comillas indican que es un valor literal.
2. **Coma después de CATEGORY\_ID**:
   * Antes de CONSTRAINT, debe haber una coma para separar la última columna de la definición de la clave primaria.

**Consejos para evitar errores en Oracle:**

* Usa herramientas como **SQL Developer** o **PL/SQL Developer**, que resaltan errores de sintaxis.
* Si usas scripts, prueba primero con sentencias simples y luego ve añadiendo complejidad.

¿Necesitas ajustar algo más en la tabla? 😊