

TEMA 7 PARTE I.
CREACIÓN, MODIFICACIÓN Y ELIMINACIÓN DE OBJETOS CON SQL

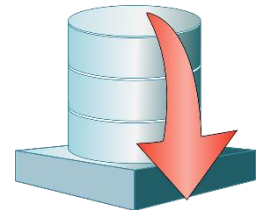
7.1. INTRODUCCIÓN A SQL

SQL *Structured Query Language*, actualmente *Database Language Query*, es un lenguaje de base de datos empleado para:

- Creación y manipulación de las estructuras de una base de datos.
- Administración de los datos
- Ejecución de sentencias complejas diseñadas para transformar los datos almacenados en información útil.

7.2. CARACTERÍSTICAS PRINCIPALES DEL SQL

- De estructura sencilla, fácil de entender, al ser un lenguaje de alto nivel.
- Diseñado para trabajar con conjunto de datos.
- No procedimental.
- Portable. Existencia de estándares regulados que permiten el uso del lenguaje relativamente independiente al manejador que se utilice.
- Existencia de extensiones empleadas para evitar los problemas que implica la falta de sentencias de control y estructuras que proporciona un lenguaje procedimental. En este caso cada manejador define sus propias extensiones del SQL:
 - PL-SQL en Oracle
 - Transact SQL en SQL Server (Microsoft)
 - SQL-PL En DB2 (IBM)



7.3. CATEGORÍAS DEL SQL

El lenguaje SQL está dividido en diversas categorías clasificadas con base a su uso:

DDL (Data Definition Language)

Lenguaje de definición de datos. Es el lenguaje encargado de la creación, modificación y eliminación de la estructura de los objetos de la base de datos (tablas, índices, vistas, etc).

DML (Data Manipulation Language)

Lenguaje de manipulación de datos. Es el lenguaje que permite realizar las tareas de consulta, modificación y eliminación de los datos almacenados en una base de datos.

DCL (Data Control Language)

Lenguaje de control de datos. Es el lenguaje encargado de configurar y establecer el control de acceso a la base de datos. Incluye instrucciones para definir accesos y privilegios a los distintos objetos de la base de datos.


DQL (Data Query Language)

Lenguaje de consulta de datos. Algunos autores clasifican a la instrucción SELECT como el único elemento de una cuarta categoría del lenguaje SQL Data Query Language (DQL).

Transaction Control

Control de transacciones. Es el lenguaje empleado para crear, y administrar transacciones aplicadas a un conjunto de sentencias DML principalmente.

La siguiente tabla muestra las principales clausulas SQL divididas por categoría:

Lenguaje	Clausulas básicas
DDL (Data Definition Language)	<div> <code>create</code> <code>alter</code> <code>drop</code> <code>rename</code> <code>truncate</code> <code>comment</code> </div> 
DML (Data Manipulation Language)	<div> <code>insert</code> <code>update</code> <code>delete</code> <code>merge</code> </div>
DCL (Data Control Language)	<div> <code>grant</code> <code>revoke</code> </div>
DQL (Data Query Language)	<div> <code>select</code> </div>
Transaction Control	<div> <code>commit</code> <code>rollback</code> <code>savepoint</code> </div>

7.4. BREVE HISTORIA DE LOS ESTÁNDARES SQL

En 1978 el “Committe on Data Systems and Language” (CODASYL) inicia los trabajos para definir el DDL y el DML. En 1986 se convirtieron en estándares aprobados por el American National Standards Institute’s Technical Committee X3H2 (Database) o también llamado NCITS (National Committee for Information Technology Standards).

7.4.1. SQL 86/87

En 1982 se le encargo al X3H2 la estandarización del modelo relacional basado inicialmente en la especificación de modelo IBM SQL/DS, IBM DB2 fue tomado como modelo. En 1984 el modelo fue rediseñado para ser más genérico. EN 1986 fue aprobado como **estándar nacional americano**. ISO (International Organization for Standarization) adoptó al lenguaje en 1987.

7.4.2. SQL 89 (SQL 1.1)

Agrega definiciones menores empleadas para estandarizar el lenguaje para los RDBMS existentes en ese momento. Contiene varias limitantes las cuales son implementadas y definidas de forma particular por cada RDBMS.

7.4.3. SQL 92 (SQL 2)

Considerado como el primer estándar SQL sólido tomado y corregido por el ANSI. La especificación creció aproximadamente en **500 páginas** más con respecto al anterior estándar. Algunas de estas mejoras incluyen:

- Nombrado de constraints
- Soporte para varchars
- National characters
- Case, cast expressions
- Operadores Join
- ALTER TABLE, etc.

7.4.4. SQL 99 (SQL 3)

Desarrollado en conjunto por ISO, ANSI, el estándar fue formado por alrededor de **2000 páginas**. En esta versión se extiende el modelo relacional tradicional para incorporar objetos y tipos de datos complejos, incorpora conceptos de la programación orientada a objetos (POO) como herencia, polimorfismo, encapsulamiento, y polimorfismo.

7.4.5. SQL 2003

El tema principal en este estándar es XML. Agrega funcionalidades como:

- Table functions
- Sequence generators
- Auto generated values
- Identity columns

7.4.6. SQL 2008

Incorpora mejoras a la integración de tipos de dato XML, incluye:

- Binary Data Type
- Soporte para expresiones regulares.
- Vistas materializadas.
- First n, Top n Queries.
- Uso de la cláusula `order by` de manera externa a la definición de cursores, se agrega soporte para `instead of triggers`, se agrega la sentencia `truncate` y la cláusula `fetch`

7.4.7. SQL 2011

- La principal característica de esta versión es la incorporación del concepto de **bases de datos temporales**. Su idea principal es almacenar datos que son considerados válidos o correctos en un periodo de tiempo específico el cual puede ser un periodo del pasado, actual o futuro.
- Este tipo de bases de datos permite almacenar, por ejemplo, la historia de vida de una persona. Un registro puede especificar que una persona habitó de 1990 al 2000 en New York, y otro especifica que a partir del 2001 a la fecha habita en California. Al actualizar la dirección de residencia, los datos anteriores no se eliminan, solo se actualizan los periodos de tiempo: 1990 al 2000 y 2001 a la fecha (se emplea el valor ∞ para especificar que el tiempo fin aun no ocurre).

- Para implementar esta idea, se agregan los conceptos de **Valid Time, Valid-From, Valid-To**.
- Se agrega soporte para manejo de datos temporales (PERIOD FOR), Temporal Primary Keys, Temporal referential integrity constraints.
- Se incorporan mejoras a la cláusula `fetch`.

7.4.8. SQL 2016

- Ofrece alrededor de 44 nuevas funcionalidades, de las cuales 22 de ellas se relacionan con el soporte para manejo de documentos JSON: creación de documentos, modificación, búsquedas.
- Alrededor de 10 nuevas funcionalidades se agregan para el manejo de funciones polimórficas.
- Se agrega soporte para recuperar registros que tengan correspondencia con expresiones regulares (predicados basados en expresiones regulares).
- Parseo y formateo de fechas
- Nuevo tipo de dato: `decfloat`.

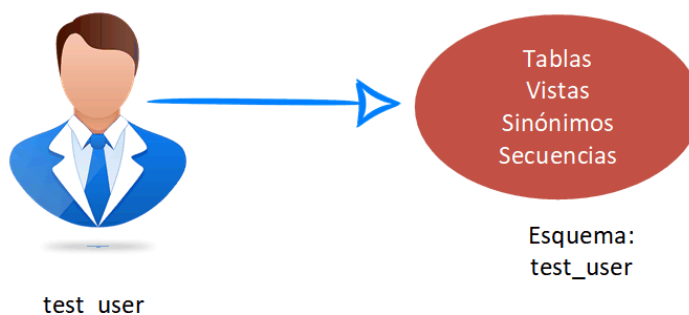
7.5. CREACIÓN DE TABLAS

Antes de realizar cualquier actividad en el RDBMS es necesario haber realizado las siguientes actividades básicas

- Tener un modelo de datos diseñado, revisado y aprobado, listo para ser implementado en algún RDBMS.
- Instalación y configuración del manejador de base de datos considerando los requerimientos tanto funcionales como no funcionales del proyecto.
- Creación de la base de datos.
- Creación de esquemas y/o usuarios.
- Definición de privilegios y control de acceso.

7.5.1. Esquemas y objetos.

- Esquema: Contenedor lógico de objetos.
- En Oracle, al crear un usuario, se le asigna de forma automática su único esquema cuyo nombre es el mismo que el nombre del usuario (Relación 1 -1)
- El esquema contiene todos los objetos y los datos que le pertenecen al usuario

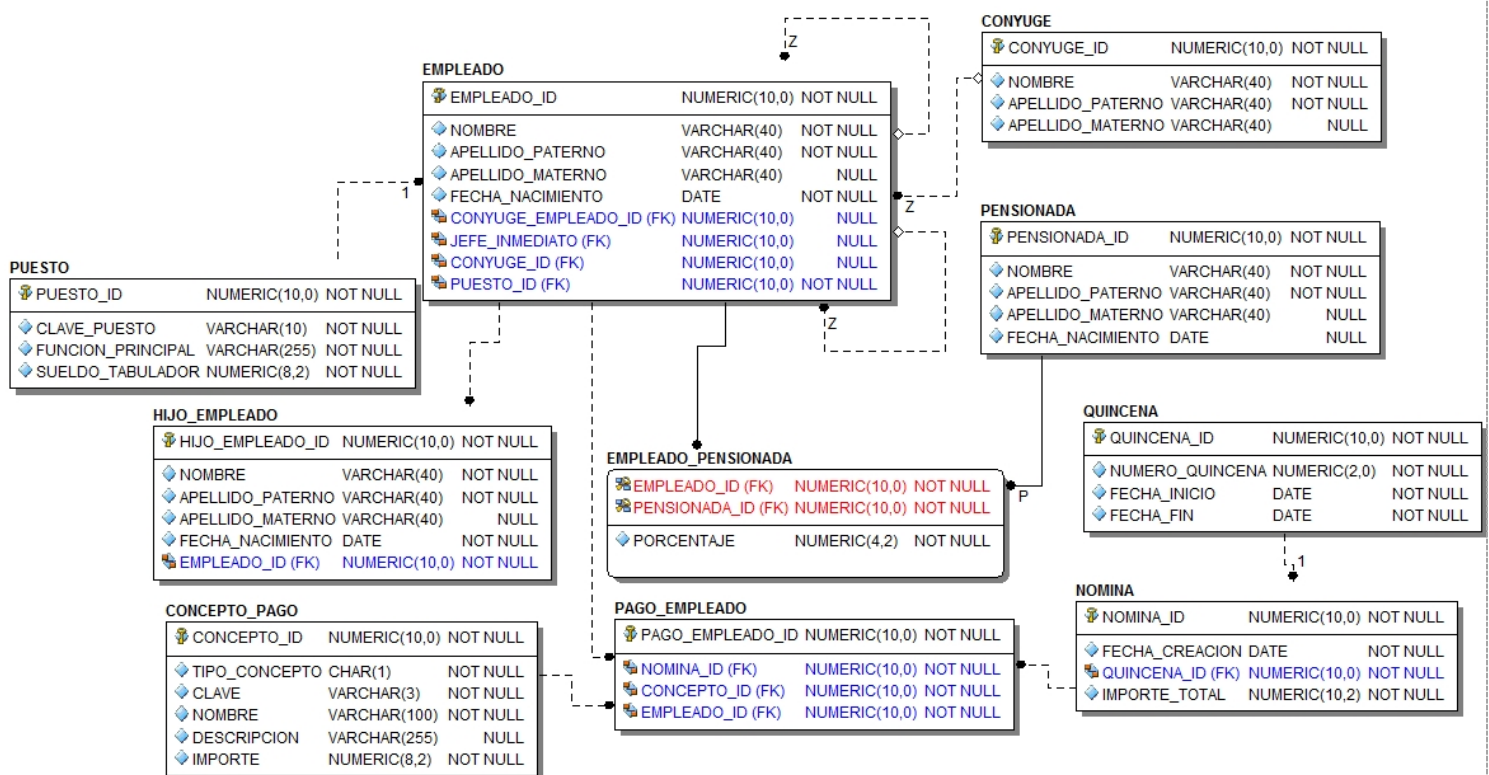


- En una BD productiva, un esquema representa a una **aplicación** y no tanto a un usuario o a una **persona**.

7.5.1.1. Principales tipos de objetos.

- Tablas
- Restricciones
- Índices
- Particiones
- Vistas
- Secuencias
- Sinónimos
- Programas PL/SQL

Para realizar la creación de los principales objetos de la base de datos se considera el siguiente modelo relacional que corresponde a la base de datos de un sistema de nómina.



7.5.2. Sintaxis create statement SQL estándar

La definición de tablas varía considerablemente entre manejador y manejador. Por ejemplo, en Oracle, la sintaxis empleada para definir una tabla está formada aproximadamente de **15 páginas**. Sin embargo, es posible hacer uso de la definición de una tabla empleando SQL estándar.

```

create [{global | local} temporary] table <table_name> (
    <column_name>
    [ <domain_name> | <datatype>[<size1>[, <size2>]]
    [generated [always | by default] as identity <identity_clause>]
    ]
    [<column_constraint>, ...]
    [default <default_value>]
    [collate <collation_name>]
    , ...
    [<table_constraints>]
) [on commit {delete | preserve} rows]

```

Nomenclatura: Variante de la notación **Backus-Naur Form**

Símbolo	Significado
[]	Indica que todos los elementos SQL que se encuentren entre corchetes puede omitirse (contenido opcional).
()	Los paréntesis son elementos propios de la sintaxis SQL
< >	Los símbolos < > deben ser reemplazados por el valor del elemento que representan. Por ejemplo <column_name> debe ser reemplazado por el valor del nombre de la columna de una tabla.
, ...	La coma y los puntos suspensivos, indica que un elemento puede aparecer o repetirse varias veces.
{ }	Las llaves se emplean para agrupar a un conjunto de elementos, normalmente acompañados de expresiones lógicas. Por ejemplo: {<domain_name> <datatype>} significa que una sentencia SQL puede aparecer cualquiera de los 2 elementos del grupo <domain_name> o <datatype>
	Barra vertical (pipe) indica operación lógica OR empleada para indicar las diferentes opciones que pueden aparecer dentro de la sintaxis de una sentencia SQL.

7.5.2.1. Sintaxis ORACLE

En Oracle es posible crear 2 categorías de tablas principales:

- **Relational tables.** Representa la estructura básica (formato tabular) para almacenar datos.
- **Object tables.** Los tipos de datos de cada columna son tipos de datos personalizados: **Object data types**. Para efectos del curso, se revisan solo tablas relacionales.

Ejemplo:

```

create [ global temporary] table [< schema >.]<table_name>(
    <column_name>{
    [
        <datatype>[<size1>[, <size2>]]
        [default <default_value>]
        [encrypt <encryption_specs>]
        [<column_constraint>, ...]
    ] |
    as < expression > virtual, ...
    [<table_or_column_constraint>, ...]
    [physical_properties]
) [on commit {delete | preserve} rows]

```

En [este enlace](#) puede revisarse la sintaxis completa de esta instrucción.

Ejemplo:

Crear siguiente tabla empleado empleando sintaxis SQL estándar y en Oracle

En SQL estándar:

```
create table empleado (
  nombre varchar(40),
  apellido_paterno varchar(40),
  apellido_materno varchar(40),
  fecha_nacimiento date,
  tipo_empleado char(1),
  sueldo_base numeric(8,2),
  foto blob,
  titulado boolean
);
```

EMPLEADO

◆ NOMBRE	VARCHAR(40)	NULL
◆ AP_PATERNO	VARCHAR(40)	NULL
◆ AP_MATERNO	VARCHAR(40)	NULL
◆ FECHA_NACIMIENTO	DATE	NULL
◆ TIPO_EMPLEADO	CHAR(1)	NULL
◆ SUELDO_BASE	NUMERIC(8,2)	NULL
◆ FOTO	VARBINARY/BLOB(max)	NULL
◆ TITULADO	BIT	NULL

En Oracle:

```
create table empleado (
  nombre varchar2(40),
  apellido_paterno varchar2(40),
  apellido_materno varchar2(40),
  fecha_nacimiento date,
  tipo_empleado char(1),
  sueldo_base number(8,2),
  foto blob,
  titulado number(1,0)
);
```

- Para agregar comentarios a una tabla:

```
comment on table empleado is '<comentario>';
```

- En la práctica todas las tablas generalmente definen una llave primaria, sin embargo, no es requisito indispensable. Es posible crear tablas sin llave primaria.

7.5.3. Organización de almacenamiento.

En Oracle, el almacenamiento de los datos puede organizarse con las siguientes características:

- Heap organized tables
- Index organized tables
- External tables
- Temporary tables.

7.5.3.1. Heap and Index organized tables.

- **Ordinarias** (Heap organized table) Representa el tipo más común en el que los registros no se guardan en algún orden el particular. La tabla creada anteriormente representa una tabla de este tipo.
- **Indexadas** (Indexed- organized tables) Los registros se ordenan con base a los valores de la PK.

7.5.3.2. Tablas externas.

- Son tablas de solo lectura. La definición de su estructura o metadatos son almacenados en el diccionario de datos, pero los datos se encuentran en una fuente externa a la base de datos, por ejemplo, en archivos de texto: archivos CSV, etc.

Ejemplo:

- Crear una tabla externa con la siguiente estructura. Los datos se encuentran en un archivo csv llamado `empleado_ext.csv`

EMPLEADO_EXT

◆ NUM_EMPLEADO	NUMERIC(10,0)	NULL
◆ NOMBRE	VARCHAR(40)	NULL
◆ AP_PATERNO	VARCHAR(40)	NULL
◆ AP_MATERNO	VARCHAR(40)	NULL
◆ FECHA_NACIMIENTO	DATE	NULL



- Observar que todos los campos están definidos como campos opcionales. Este es un requisito para poder crear tablas externas. La razón de este requisito es mantener máxima flexibilidad, en especial para archivos externos que no contienen todos los valores de las columnas.

El siguiente código ilustra la forma en la que se crea la tabla externa y se realiza la lectura de datos desde un archivo csv.

```
--Se requiere del usuario SYS para crear un objeto tipo
--directory y otorgar privilegios.
prompt Conectando como sys
connect sys as sysdba

--Un objeto tipo directory es un objeto que se crea y almacena en el
-- diccionario de datos y se emplea para mapear directorios
-- reales en el sistema de archivos. En este caso tmp_dir es un
-- objeto que apunta al directorio /tmp/bases del servidor
prompt creando directorio tmp_dir
create or replace directory tmp_dir as '/tmp/bases';

--se otorgan permisos para que el usuario jorge0507 de la BD pueda leer
--el contenido del directorio
grant read, write on directory tmp_dir to jorge0507;

prompt Conectando con usuario jorge0507 para crear la tabla externa
connect jorge0507
show user
prompt creando tabla externa
create table empleado_ext (
    num_empleado    number(10, 0),
    nombre          varchar2(40),
    ap_paterno       varchar2(40),
    ap_materno       varchar2(40),
    fecha_nacimiento date
)
organization external (
    --En oracle existen 2 tipos de drivers para parsear el archivo:
    -- oracle_loader y oracle_datapump
```



```

type oracle_loader
default directory tmp_dir
access parameters (
  records delimited by newline
  badfile tmp_dir:'empleado_ext_bad.log'
  logfile tmp_dir:'empleado_ext.log'
  fields terminated by ','
  lrtrim
  missing field values are null
  (
    num_empleado, nombre, ap_paterno, ap_materno,
    fecha_nacimiento date mask "dd/mm/yyyy"
  )
)
location ('empleado_ext.csv')
)
reject limit unlimited;

--Dentro de sqlplus se pueden ejecutar comandos del s.o. empleando '!'
--En esta instrucción se crea el directorio /tmp/bases para
--copiar el archivo csv
prompt creando el directorio /tmp/bases en caso de no existir
!mkdir -p /tmp/bases

-- Asegurarse que el archivo csv se encuentra en elmismo
-- directorio donde se está ejecutando este script.
-- De lo contrario, el comando cp fallará.
prompt copiando el archivo csv a /tmp/bases
!cp empleado_ext.csv /tmp/bases
prompt cambiando permisos
!chmod 777 /tmp/bases

prompt mostrando los datos

col nombre format a20
col ap_paterno format a20
col ap_materno format a20

select * from empleado_ext;

```

- Observar la sección `access parameters`. Aquí se especifican las configuraciones necesarias para acceder al archivo `csv`.
- Los campos `badfile` y `logfile` especifican los nombres de los archivos que contienen información acerca del resultado del parseo del archivo. En el archivo `empleado_ext_bad.log` se guardan los registros que no pudieron ser parseados de forma correcta. En el archivo `empleado_ext.log` se guardan mensajes generales del proceso de parseo. Ambos archivos se guardarán en el directorio `tmp_dir`, el cual apunta a `/tmp/bases`
- Observar la lista de atributos que se especifican entre paréntesis. Esta lista corresponde con la lista de atributos de la tabla externa. El orden corresponde con el orden del archivo `csv`.
- Observar el valor del atributo `location`. Aquí se especifica el nombre del archivo que contiene los datos (datos externos a la base de datos). En este ejemplo el archivo es `empleado_ext.csv` contiene los siguientes datos:

```

100,juan,lopez,martinez,10/10/1987
101,mario,jimenez,perez,23/08/1999

```

- Notar que la tabla externa se crea sin importar si existe o no el archivo de datos. La validación se realiza hasta que se hace una consulta.
- Finalmente, la instrucción `reject limit unlimited` es empleada para validar el número máximo de errores que pueden ocurrir al leer el contenido del archivo csv. Si se excede del valor configurado, toda la instrucción se considera como fallida. En este caso se especifica el valor `unlimited`. Esto significa que se intentará leer todos los renglones del archivo sin importar el número de errores que se hayan encontrado.

Las instrucciones que siguen después de la creación de la tabla se encargan de realizar las siguientes actividades para probar que la tabla externa funciona correctamente.

- Como se indica en los comentarios, en `Sql *Plus` se pueden invocar comandos del sistema operativo anteponiendo el carácter `!`. En este caso, se crea el directorio `/tmp/bases`. Notar que el manejador nunca lo crea. Por esta razón se invoca este comando.
- En la siguiente instrucción se copia el archivo csv que contiene los datos de la tabla al directorio `/tmp/bases`. El archivo se debe encontrar en el mismo directorio donde se ejecuta este código. En su defecto, se puede especificar una ruta absoluta.
- Finalmente se ejecuta la instrucción `select` para visualizar los datos:

`prompt` mostrando los datos

```
col nombre format a20
col ap_paterno format a20
col ap_materno format a20
```

```
select * from empleado_ext;
```

num_empleado	nombre	ap_paterno	ap_materno	fecha_nacimiento
100	juan	lopez	martinez	10/10/87
101	mario	jimenez	perez	23/08/99

- Observar que los datos se muestran como si se encontraran almacenados en la BD, pero en realidad se leen del archivo csv.

7.5.3.3. Tablas temporales:

Una tabla temporal contiene datos que existen únicamente durante la existencia de una **transacción** o de una **sesión**.

- Los datos de una tabla temporal son privados a cada sesión de usuario. Si un usuario se conecta a la base de datos en 2 sesiones diferentes, la sesión 1 no podrá ver los datos que inserta o actualiza la otra sesión.

Existen 2 tipos de tablas temporales:

- Global temporary table
- Private temporary table.

La siguiente tabla muestra sus características y diferencias.

Característica	Global	Privada
Reglas para nombrado	Mismas reglas que las tablas permanentes.	Su nombre debe iniciar con ora\$ptt
Visibilidad de la definición de la tabla	Visible para todas las sesiones.	Visible únicamente para la sesión que crea la tabla
Lugar donde se almacena la definición de la tabla (metadatos)	En disco (dentro del diccionario de datos)	Únicamente en memoria
Tipos	<ul style="list-style-type: none"> Específica a una transacción: <code>on commit preserve rows</code> Específica de la sesión: <code>on commit preserve rows</code> 	<ul style="list-style-type: none"> Específica de la transacción: <code>on commit drop definition</code> Específica de la sesión: <code>on commit preserve definition.</code>

Como se menciona en la tabla anterior, los datos insertados y/o la definición de la tabla temporal desaparecen cuando ocurre alguno de los siguientes 2 eventos:

- Al cerrar sesión del usuario
- Al terminar una transacción (al ejecutar la instrucción `commit`).

Usos de una tabla temporal global.

- Se emplean principalmente en casos donde se desea mantener datos en memoria y que no necesariamente se desea que se guarden de forma permanente después de cierto tiempo.
- Por ejemplo: Un carrito de compras. Mientras el usuario está en sesión, el usuario puede insertar datos a una tabla. Al término de su transacción o de su sesión, algunos registros pudieran ser copiados hacia una tabla permanente y todos los datos de la tabla temporal son eliminados de forma automática.
- Se emplean en especial para el cálculo de estadísticas y generación de reportes en donde se requiere consultar cantidades grandes de datos. Los datos se leen de múltiples tablas permanentes y se almacenan en una sola tabla temporal, por ejemplo, aplicando un proceso de denormalización. Una vez que la tabla temporal se ha poblado se realizan los cálculos.

Ejemplos tablas temporales globales:

```
create global temporary table empleado_temp_commit (
    nombre varchar2(100),
    email varchar2(200)
) on commit delete rows;

insert into tabla_temporal (nombre,email) values(
    'jorge rodriguez','jorgerdc@gmail.com');

select * from tabla_temporal;

nombre          email
-----
jorge rodriguez jorgerdc@gmail.com

commit;
```

Al hacer commit, los datos desaparecen de la tabla temporal:

```
select * from tabla_temporal;
```

```
nombre          email
```

```
-----
```

En general, la cláusula `on commit` puede tener la siguiente forma:

```
on commit <preserve|delete> rows
```

Ejemplo:

```
create global temporary empleado_temp_preserve (
  nombre varchar2(100),
  email varchar2(200)
) on commit preserve rows;
```

- Con la instrucción `preserve` los registros siguen existiendo a pesar de haber realizado múltiples commits. Los datos permanecerán, pero al salir de sesión estos desaparecerán.
- Instrucciones como `commit`, `rollback`, `savepoint`, se emplean para realizar el manejo de transacciones en una base de datos. Su uso es una de las características más importantes que debe ofrecer cualquier manejador principalmente porque garantizan que la base de datos no caiga en un estado inconsistente cuando se realizan múltiples operaciones y por alguna razón alguna(s) de esas operaciones no se ejecutaron de manera adecuada, (o todo o nada). Este concepto se revisará a detalle en el tema 8.

Usos de una tabla temporal privada.

- En casos donde aplicaciones almacenan datos temporales en tablas transitorias, es decir, se crean, por lo general se insertan datos una sola vez, se consultan un par de veces y se eliminan al finalizar la transacción o de la sesión.
- Cuando se tienen sesiones abiertas por tiempo indefinido y se desea crear diferentes tablas por cada transacción.
- Cuando la creación de una tabla NO debe crear una nueva transacción o terminar una existente.
 - Si una sentencia DDL, por ejemplo, `create table` es incluida como parte de las instrucciones de una transacción, dicha sentencia DDL provocará que la transacción termine (esto se revisará en el tema 8). Sin embargo si se crea una tabla temporal privada, la transacción no será terminada de forma implícita.
- Cuando diferentes sesiones asociadas a un mismo usuario deben emplear el mismo nombre para crear tablas con estructura diferente.
- Cuando se requiere crear una tabla para propósitos de modo de lectura únicamente.

Ejemplo:

```
--creando la tabla privada y temporal.
create private temporary table ora$ptt_calculo_nomina_txn(
  nomina_id number(10,0),
  fecha_calculo date,
  importe_aproximado number(20,2)
) on commit drop definition;
```

```
--insertando algunos datos
insert into ora$ptt_calculo_nomina_txn(nomina_id, fecha_calculo,
    importe_aproximado)
values(1, sysdate, 56780405.23);
```

```
--mostrando los datos
select * from ora$ptt_calculo_nomina_txn;
```

nomina_id	fecha_calculo	importe_aproximado
1	22-11-2020 23:58:58	56780405.23

```
--realizando commit
commit;
```

```
--verificando que la tabla ya no existe.
select * from ora$ptt_calculo_nomina_txn;
```

```
ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
Error at Line: 24 Column: 15
```

Si se desea conservar la tabla hasta que la sesión termine, la definición de la tabla cambia de la siguiente manera:

```
create private temporary table ora$ptt_calculo_nomina_session(
    nomina_id number(10,0),
    fecha_calculo date,
    importe_aproximado number(20,2)
) on commit preserve definition;
```

7.6. CREACIÓN DE COLUMNAS.

En general para crear columnas se especifica el nombre, el tipo de dato y su longitud. En algunos tipos de datos la longitud se determina de forma automática. Por ejemplo, columnas con tipo de dato date.

Las siguientes secciones muestran algunas características que pueden ser asociadas al momento de crear una columna. Algunas de estas funcionalidades son particulares de Oracle.

7.6.1. Columnas virtuales (sólo para Oracle).

Las columnas virtuales son útiles en especial para los atributos derivados. El valor del atributo solo se calcula y no se persiste en la base de datos. Para crear una tabla con uno o más campos virtuales se emplea la palabra `virtual` en la definición de la columna.

Sintaxis:

```
<column_name>[datatype] [generated always] as <expression> [virtual]
```

Ejemplo:

```

create table promedio (
  calificacion_uno number(4,2),
  calificacion_dos number(4,2),
  calificacion_tres number(4,2),
  promedio generated always as (
    (calificacion_uno + calificacion_dos + calificacion_tres)/3) virtual
);

insert into promedio (calificacion_uno,calificacion_dos,calificacion_tres)
values (7.9,8.4,10);

select * from promedio;

```

calificacion_uno	calificacion_dos	calificacion_tres	promedio
7.9	8.4	10	8.766666666666666

- Observar que el campo `promedio` nunca se persiste en la base de datos.
- La sentencia `select` no incluye el valor para el promedio, ya que este es un campo derivado.
- El manejador se encarga de calcular su valor al momento de recuperar su valor evaluando la expresión que aparece en la definición del campo.
- No es posible emplear un campo virtual para calcular el valor de otro campo virtual.

7.6.2. Valores por default en una columna:

El manejador puede asignar valores por default (valores por omisión) a una columna para los casos en los que no se especifique un valor al momento de realizar una inserción. Se emplea la instrucción `default` seguido del valor a asignar.

Ejemplo:

Sintaxis Oracle:

```

create table producto (
  producto_id number(10,0),
  tipo char(1) default 'A',
  nombre varchar2(10) not null,
  fecha_creacion date default sysdate
);

```

En este ejemplo, el manejador asignará el valor que genera la función `sysdate` cuando no se proporcione un valor para el campo `nombre`. Para el caso del campo `tipo`, el valor por defecto será 'A'.

Ejemplos:

```

insert into producto(producto_id,nombre) values(1,'lap-top');

insert into producto(producto_id,tipo,nombre,fecha_creacion)
values (2,'b','mouse',to_date('01/01/2010 12:00:00',
'dd/mm/yyyy hh24:mi:ss'));

```

```
select * from producto;
```

```
producto_id  tipo  nombre  fecha_creacion
-----
1            A    LAP-TOP  2010-10-27 23:18:21
2            B    MOUSE   2010-01-01 12:00:00
```

- En la primera instrucción `insert`, se omiten ambos valores, y en el segundo se especifican otros diferentes a los valores por default. Observar la diferencia en los datos almacenados.
- ¿Qué valor tendría el campo `fecha_creación` si se inserta el siguiente registro:

```
insert into producto(producto_id,nombre,fecha_creacion)
values(1,'lap-top',null);
```

- El valor de la fecha de creación será nula a pesar de existir un valor por default definido.

7.7. CREACIÓN DE CONSTRAINTS

- Recordando lo revisado en temas anteriores, en especial los conceptos asociados con el modelo Relacional, las restricciones o constraints, se emplean para ayudar a cuidar la integridad de los datos.
- La sintaxis empleada para crear un constraint es similar entre los distintos manejadores:

```
constraint [<nombre_constraint>] <tipo_constraint> <expresión>
```

- Como se puede observar en la sintaxis anterior, cualquier constraint está formado por un nombre, el tipo de constraint y una expresión que depende del tipo.
- Si no se especifica el *nombre*, el manejador **asigna uno por default**, sin embargo, se **recomienda** especificar siempre un nombre que sea significativo y fácil de identificar a dicho constraint.
- Lo anterior es útil para los casos en los que se comete algún error y se produce un error de violación del constraint. El manejador generará un mensaje de error incluyendo el nombre del constraint.

La siguiente tabla muestra un resumen de los tipos de constraints vistos en temas anteriores, así como la convención de nombrado propuesta para el curso y recomendada en la práctica.

Tipo Constraint	Convención de nombrado
<code>not null</code>	Para este tipo de constraint, generalmente se emplea la forma corta y no se especifica un nombre: <pre>create table empleado(num_empleado number not null);</pre> Si se desea emplear la sintaxis antes mencionada, la convención es: <code><nombre_tabla> <nombre_columna> nn</code>
<code>unique</code>	<code><nombre_tabla> <nombre_columna> uk</code>
<code>primary key</code>	<code><nombre_tabla> pk</code>
<code>references,</code> <code>foreign key</code>	<code><nombre_tabla_hija>_<nombre_columna>_fk</code>
<code>check</code>	<code><nombre_tabla> <nombre_columna> chk</code>

- El nombre de la tabla es necesario sobre todo en casos donde existan columnas con el mismo nombre en diferentes tablas.

- Lo anterior se debe a que un constraint es tratado como un objeto más y por lo tanto se requiere que cuente con un nombre único dentro de un mismo esquema.

En SQL, los constraints pueden aparecer en 2 lugares dentro de la definición de una tabla:

- A nivel campo (aparecen como parte de la definición del atributo): **Column constraints**
- A nivel tabla (aparecen como parte de la definición de una tabla posterior a la definición de sus atributos): **Table constraints**.

7.7.1. Column constraints.

Ejemplos:

- Crear la tabla `concepto_pago`. La clave de cada concepto debe ser única, el tipo de concepto solo puede tener los valores A, B y C, y el importe de cada concepto no debe pasar de \$100000.
- Observar que la PK de la tabla corresponde al campo `concepto_id`

CONCEPTO_PAGO		
CONCEPTO_ID	NUMERIC(10,0)	NOT NULL
TIPO_CONCEPTO	CHAR(1)	NOT NULL
CLAVE	VARCHAR(3)	NOT NULL
NOMBRE	VARCHAR(100)	NOT NULL
DESCRIPCION	VARCHAR(255)	NULL
IMPORTE	NUMERIC(8,2)	NOT NULL

Sintaxis SQL estándar:

```
create table concepto_pago (
  concepto_id numeric(10, 0) constraint concepto_pago_pk primary key,
  tipo_concepto char(1) not null constraint cp_tipo_concepto_chk check (
    tipo_concepto in ('A', 'B', 'C')),
  clave varchar(3) not null constraint cp_clave_uk unique,
  descripcion varchar(255),
  importe numeric(8, 2) not null constraint cp_importe_chk check (
    importe < 100000
  )
);
```

- Observar que para el caso del constraint `not null` no fue necesario especificar la palabra `constraint`. Si se desea usar la sintaxis anterior se tendrá:

```
tipo_concepto char(1) constraint cp_tipo_concepto_nn not null
```

La siguiente tabla muestra el contenido del diccionario de datos en Oracle que guarda información de los constraints creados anteriormente para la tabla `concepto_pago`.

Constraint Name	Column Name	Search Condition	Status	Type	Delete Rule	Generated	Condition
CLAVE_UNIQUE	CLAVE	{null}	ENABLED	U	{null}	USER NAME	unique (CLAVE)
CONCEPTO_PAGO_PK	CONCEPTO_ID	{null}	ENABLED	P	{null}	USER NAME	PK (CONCEPTO_ID)
IMPORTE_CHECK	IMPORTE	IMPORTE <100000	ENABLED	C	{null}	USER NAME	{null}
SYS_C0011137	TIPO_CONCEPTO	"TIPO_CONCEPTO" IS NOT NULL	ENABLED	C	{null}	GENERATED NAME	{null}
SYS_C0011138	CLAVE	"CLAVE" IS NOT NULL	ENABLED	C	{null}	GENERATED NAME	{null}
SYS_C0011139	IMPORTE	"IMPORTE" IS NOT NULL	ENABLED	C	{null}	GENERATED NAME	{null}
TIPO_CONCEPTO_CHECK	TIPO_CONCEPTO	TIPO_CONCEPTO IN ('A','B','C')	ENABLED	C	{null}	USER NAME	{null}

Observar, que los nombres que genera Oracle para identificar a un constraint inician con la palabra `SYS`.

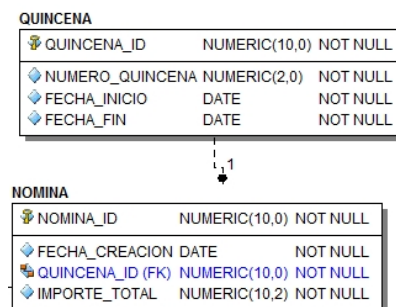
7.7.1.1. Creando llaves foráneas.

Crear la tabla `NOMINA`. Observe que para poder crear la tabla, se requiere crear antes la tabla `QUINCENA`, esto debido a que en la tabla `NOMINA`, `QUINCENA_ID` es una FK con la tabla `QUINCENA`:

Sintaxis Oracle:

```
create table quincena(
    quincena_id number(10,0) constraint quincena_pk
primary key,
    numero_quincena number(2,0) not null,
    fecha_inicio date not null,
    fecha_fin date not null
);
```

```
create table nomina(
    nomina_id number(10,0) constraint nomina_pk primary key,
    fecha_creacion date not null,
    quincena_id not null constraint quincena_id_fk references
    quincena(quincena_id)
);
```



Observar que en la definición del campo `quincena_id` en `nomina`, no se incluye el tipo de dato.

- El uso de column constraints tiene las siguientes limitantes e inconvenientes:
 - La palabra `constraint` no puede aparecer más de una vez en la definición del campo.
 - Restricciones como son llaves primarias o restricciones `Unique` que estén formadas por la definición de más de un atributo no pueden ser definidas a través de esta técnica.
 - La definición del atributo se vuelve más compleja y en algunos casos, complica la lectura del código SQL ya que se está definiendo un campo y un `constraint` a la vez.
- Para resolver estos inconvenientes se emplean `table constraints`.

7.7.2. Creación de constraints a nivel tabla (table constraints).

Son similares a los `constraints` definidos a nivel columna, con la diferencia de que estas se definen a nivel tabla, posterior a la definición de todos los campos de la tabla. Los tipos de `constraints` que pueden definirse a nivel de tabla son:

- `unique`
- `primary key`
- `foreign key`
- `check`

Como se mencionó anteriormente, el uso de esta técnica resuelve inconvenientes como son, la creación de más de un `constraint` por atributo, la definición de PKs y `constraints` de tipo `Unique` que hacen uso de más de un atributo a la vez.

Ejemplo:

Sintaxis SQL estándar:

Crear la tabla `venta`, agregar su PK. Considerar que los valores para el campo `tipo_venta` únicamente tendrán los valores `MA` o `EL`.

```
create table venta(
    venta_id numeric(10,0) not null,
    fecha_venta date not null,
    tipo_venta char(3),
    constraint venta_pk primary key(venta_id),
    constraint ve_tipo_venta_chk check(tipo_venta in('MA','EL'))
);
```

Como se puede observar en el código SQL, la definición del constraint ya no se asocia al campo si no a la tabla separándolos por comas.

Ejemplo:

Creando un FK con un constraint a nivel tabla:

```
create table orden_venta(
    orden_id numeric(10,1) not null,
    venta_id numeric(10,0) not null,
    constraint orden_venta_pk primary key(orden_id),
    constraint venta_ov_venta_id_fk foreign key(venta_id)
    references venta(venta_id)
);
```

7.7.3. Creación de Restricciones de integridad referencial (Referential Integrity constraints)

Estos constraints de integridad aplican para los constraints `references` Y `foreign key` . Especifican una acción a ejecutar para los registros de una tabla que tengan relación referencial.

Las acciones que se pueden aplicar a los registros referenciados son: eliminarlos, o modificar el valor de la FK.

Sintaxis SQL estándar:

```
[on {delete | update}
    {no action | cascade | set null | set default | restrict} ]
```

`no action`: Es la opción por default, el manejador genera un error si un usuario trata de eliminar el registro o actualizar el valor de la PK la cual está siendo referenciada por otras tablas. Al error que se genera se le llama “Violación de restricción de integridad”.

`cascade`: Propaga las operaciones de `delete` o `update` aplicadas en la tabla padre. Si se elimina el registro, se eliminan sus referencias. Si se actualiza la PK, se actualizan las FKs .

`set default`: Les asigna el valor configurado en la definición de la tabla.

`set null`: Al aplicar una operación de `update` o `delete` se actualiza el valor de la PK a NULL

`restrict`: Similar a `no action` con la siguiente diferencia: `no action` verifica hasta el final de la sentencia SQL, mientras que `restrict` verifica al momento, prohíbe cualquier violación de integridad.

Sintaxis Oracle:

La opción `update` no está soportada, solo `delete`, con las siguientes variantes:

```
[on delete {no action | cascade | set null}]
```

`set default`, `restrict` no son soportadas en Oracle.

Los constraints de integridad referencial se especifican posterior a la definición del constraint `references` o `foreign key`

Ejemplo:

```
create table direccion(
    direccion_id number(10,0) not null constraint direccion_pk primary key,
    calle varchar(100) not null
);

create table cliente(
    cliente_id number(10,0) not null constraint cliente_pk primary key,
    nombre varchar(100) not null,
    direccion_id not null constraint direccion_id_fk references
        direccion(direccion_id) on delete cascade
);
```

Al intentar eliminar un registro de la tabla `direccion` (tabla padre), se eliminan también sus registros referenciados en `cliente` (tabla hija).

```
insert into direccion (direccion_id,calle) values(1,'avenida central');
insert into cliente (cliente_id,nombre,direccion_id) values(1,'juan',1);

delete from direccion where direccion_id=1;
```

7.8. GENERACIÓN DE VALORES SECUENCIALES.

Empleados normalmente para la generación de valores únicos y consecutivos utilizados en las llaves primarias de las tablas.

Principales estrategias de generación de valores

- Identity clause (SQL Server, DB2, Oracle a partir de la versión 12c)
- Auto increment (MySQL)
- Sequences (Oracle, PostgreSQL)

7.8.1. Identity clause

Empleada para generar columnas de identidad (`identity columns`). En una columna de identidad (típicamente corresponde con la PK de la tabla), el manejador generará un nuevo valor único cada vez que se realice una inserción en la tabla correspondiente.

Sintaxis SQL estándar.

```
<column_name>
[ <domain_name> | <datatype>[<size1>[, <size2>]]
    [generated [always | by default] as identity <identity_clause>]
]
```

<identity_clause> está formada por las siguientes opciones:

```
[
  [start with <integer>]
  [increment by <integer>]
  [minvalue <integer>]
  [maxvalue <integer>]
  [{cycle | nocycle}]
]
```

- start With: Especifica el valor inicial
- increment by: Útil para generar incrementos mayores a 1, por ejemplo, valores en secuencias de 2 en 2, etc.
- maxvalue: Valor máximo que puede ser generado. Si se especifica no cycle, se generará un error al alcanzar el límite. ¡Si se especifica cycle, el siguiente valor a mostrar será el configurado por minvalue!
- minvalue: cómo se mencionó anteriormente, se emplea como siguiente valor al alcanzar el máximo valor secuencial cíclico.

Ejemplo:

```
create table order (
  order_id integer generated always as identity
  start with 1
  increment by 1
  minvalue 1
  maxvalue 1000
  nocycle,
  clave_orden varchar(4) not null
);
```

Ejemplo:

Insertando registros:

```
insert into orden(clave_orden) values('ade3');
```

7.8.1.1. Identity clause en Oracle, a partir de la versión 12c

Sintaxis:

```
generated
[ always | by default [ on null ] ]
as identity [ ( identity_options ) ]
```

- identity_options se refieren a las mismas opciones revisadas con la sintaxis estándar: start with, increment by, etc.
- always obliga el uso de valores auto incrementales. Esto significa que, si en una sentencia insert se hace referencia a la columna definida como identity, se generará un error.

Ejemplo:

```
--Antes de crear la tabla, el usuario debe tener privilegios para crear
-- valores secuenciales:
connect sys as sysdba
grant create sequence to jorge;

connect jorge/jorge
create table empleado_identity (
    empleado_id number generated always as identity,
    nombre varchar2(50)
);

-- esta instrucción si funciona
insert into empleado_identity(nombre) values ('Juan');

-- esta instrucción no funciona, se está haciendo referencia a empleado_id
insert into empleado_identity(empleado_id,nombre) values (2,'Juan');
--ERROR at line 1:
--ORA-32795: no se puede insertar una columna de identidad siempre generada
```

- `set default` permite hacer uso de valores proporcionados en la sentencia `insert`. Es decir, si la sentencia no contiene un valor para la columna de identidad, el valor se auto genera. Si se especifica un valor, se usa dicho valor, y se especifica un valor nulo, se genera un error ya que una columna `identity` no puede tener valores nulos.

Ejemplo:

```
create table empleado_identity_default (
    empleado_id number generated by default as identity,
    nombre varchar2(50)
);

-- esta instrucción si funciona
insert into empleado_identity_default(nombre) values ('Juan');

-- esta instruccion también funciona
insert into empleado_identity_default(empleado_id,nombre) values
(2,'Juan');

-- esta instruccion no funciona
insert into empleado_identity_default(empleado_id,nombre) values
(null,'Juan');

--ERROR at line 1:
--ORA-01400: no se puede realizar una insercion NULL en
--("JORGE"."EMPLEADO_IDENTITY_DEFAULT"."EMPLEADO_ID")
```

- `by default on` Permite la generación de un valor en caso que la columna se esté referenciando con un valor nulo. Notar que en ningún caso se genera un valor nulo.

Ejemplo:

```
create table empleado_default_on_null (
    empleado_id number generated by default on null as identity,
    nombre varchar2(20)
```

```
);

-- esta instrucción si funciona
insert into empleado_default_on_null(nombre)
values ('Juan 1');

-- esta instruccion también funciona
insert into empleado_default_on_null(empleado_id,nombre)
values (3,'Juan 3');

-- esta instruccion tambien funciona
insert into empleado_default_on_null(empleado_id,nombre)
values (null,'Juan 2');

-- contenido de la tabla (observar los valores generados.
select * from empleado_default_on_null;
```

```
EMPLEADO_ID NOMBRE
```

```
-----
1 Juan 1
3 Juan 3
2 Juan 2
```

- Detrás de escenas, Oracle hace uso de una secuencia para generar los valores de las columnas identity.
- Esto se puede validar consultando el diccionario de datos:

```
col object_name format a20
```

```
select object_name, object_type
from   user_objects
where  object_type = 'SEQUENCE';
```

```
OBJECT_NAME          OBJECT_TYPE
-----
ISEQ$$_93364         SEQUENCE
ISEQ$$_93362         SEQUENCE
ISEQ$$_93360         SEQUENCE
```

7.8.2. Auto increment columns:

Empleada para generar valores consecutivos, similar a Identity

Ejemplo:

MySQL

```
create table order(
    order_id integer not null auto_increment
);
```

7.8.3. Secuencias

La finalidad de una secuencia es similar a la de una columna tipo identity. La diferencia principal es que la secuencia es totalmente independiente a la definición de la tabla que la use. Múltiples usuarios pueden generar consecutivos de una misma secuencia de forma independiente.

Para el caso de Oracle, el valor máximo que puede tener el consecutivo es 10^{27}

Sintaxis Oracle:

```
create sequence [<schema.>]<sequence_name>
[start with <start_value>]
[increment by <increment_value>]
[maxvalue <max_value> | nomaxvalue]
[minvalue <mix_value> | nominvalue]
[cycle | nocycle]
[cache <n> | nocache]
[order | noorder]
```

- **order** Garantiza que los valores se generan en el orden que se solicitaron, en especial en un ambiente concurrente.
- **Cache:** Si esta opción es especificada, el manejador obtendrá <n> valores de la secuencia y los almacenará en cache (memoria). Cuando el usuario requiera hacer uso de un siguiente valor, el manejador obtendrá el siguiente valor secuencial del cache en lugar de obtenerlo de la secuencia. Esta técnica permite mejorar el desempeño, en especial cuando se realiza una carga masiva de datos.

Ejemplo:

Sintaxis Oracle:

```
create sequence my_sequence
start with 500
increment by 10
maxvalue 600
minvalue 0
nocycle;
```

Esta secuencia inicia en 500, cada vez que se invoque la generación del siguiente valor, se incrementa en 10 hasta llegar a 600. La secuencia no es cíclica, lo que significa que, al superar el valor máximo, el manejador generará un error. Para las secuencias cíclicas, la secuencia se reinicia comenzando con el valor especificado en MINVALUE, o con el valor 1, si MINVALUE no se especifica.

En Oracle se emplea la siguiente sintaxis para obtener el siguiente valor de la secuencia.

```
sql> select my_sequence.nextval from dual;
```

```
NEXTVAL
```

```
-----
500
```

Ejemplos:

- Insertando nuevos registros a la tabla ORDEN empleando secuencias en Oracle

```
insert into orden(orden_id,clave_orden)
values (my_sequence.nextval,'0001');
```

- Para visualizar el valor actual de una secuencia sin provocar un incremento se emplea `currval`:

```
sql> select my_sequence.currval from dual;
```

7.9. ÍNDICES.

Como se revisó en temas anteriores, los índices son estructuras de datos que ocupan espacio en disco, similar a una tabla. A nivel general, los índices se emplean para mejorar el desempeño de la base de datos (como se vio en temas anteriores).

Al realizar una consulta, el manejador verifica la existencia de un índice en las columnas que participan en las posibles condiciones que pudiera tener la consulta. Si existe, el manejador puede considerar su empleo para localizar la ubicación física de los datos. Si el índice no existe, se realiza un escaneo completo de la tabla (table full scan).

Un índice en base de datos es similar a un índice de un libro, guarda puntera hacia la ubicación física de los datos. Los índices pueden ser:

- Unique
- Nonunique

7.9.1. Recomendaciones:

- No son eficientes en tablas con pocos registros (Ejemplo: 50 registros)
- En tablas grandes, se deben crear índices únicamente si las condiciones de búsqueda de las sentencias SQL que involucran columnas indexadas regresan un porcentaje pequeño de datos (<15%)
- Útiles en columnas empleadas para realizar “joins”, por lo que es recomendable crear índices en los campos que representan una llave foránea (FKs)
- Un índice causa mayor lentitud en operaciones `INSERT`, `UPDATE`, por lo que, si una tabla contiene o se lanza una gran cantidad de estas operaciones, se deben crear pocos índices en la tabla.

7.9.2. Creación de índices:

Sintaxis Oracle:

```
create [ unique | bitmap ] index [ schema. ] index
  on { cluster_index_clause
      | table_index_clause
      | bitmap_join_index_clause
  }
[ usable | unusable ] ;
```

Ejemplo:

```
create table puesto (
    puesto_id number(10,0) constraint puesto_pk primary key,
    clave_puesto char(2) not null
);
create index pue_puesto_clave_ix on puesto(clave_puesto);
```

7.9.3. Índices Unique

Empleados para verificar la unicidad de los valores de una columna.

Ejemplo:

```
create unique index pue_puesto_clave_iuk on puesto(clave_puesto);

create unique index cliente_direccion_id_iuk on cliente(direccion_id);
```

7.9.4. Índices compuestos.

Llamados también índices concatenados, se caracterizan por estar formados de más de una columna. El orden en el que se definen estas columnas es importante por la siguiente razón:

- Un índice compuesto es útil en los casos donde la cláusula `where` de la sentencia `select` hace referencia a la primera columna del índice. En general, la primera columna que se debe especificar en la lista debe ser la columna que se accede con mayor frecuencia.

Ejemplo:

Considerar la siguiente tabla.

- Suponer que la aplicación accede frecuentemente a los 3 campos mostrados en la figura, suponer que el campo `nombre` tiene una frecuencia alta de valores diferentes, y se tiene una cantidad importante de registros.
- Un índice compuesto que puede ayudar a mejorar los tiempos de acceso a datos es:

EMPLEADO_INDEXADO

◆ NOMBRE	VARCHAR(40)	NOT NULL
◆ CLAVE_PUESTO	VARCHAR(4)	NOT NULL
◆ SALARIO	NUMERIC(10,2)	NOT NULL

```
create index emp_empleados_ix
on empleado_indexado(nombre,clave_puesto,salario);
```

Las sentencias que se pueden beneficiar de este índice son:

- Sentencias que acceden a los 3 campos
- Sentencias que acceden únicamente al campo `nombre`
- Sentencias que acceden al campo `nombre` y a alguno de los otros 2 atributos.

Sentencias que acceden a otros campos diferentes al campo `nombre` no se benefician de este índice, por lo que el manejador puede no hacer uso de él.

Para el caso de un índice compuesto tipo `unique`, la unicidad se verifica considerando las posibles combinaciones de los `n` atributos que formen al índice.

Ejemplo:

```
create index pago_numpagos_iuk
on pago (cliente_id,num_pago);
```

En este ejemplo, el índice valida unicidad considerando las posibles combinaciones de valores entre los campos `cliente_id` y `num_pago`.

7.9.5. Índices basados en funciones

Empleados cuando existen sentencias SQL frecuentes que usan funciones.

Ejemplo:

```

. . . where lower(customer_name) = 'juan'

create unique index idx_cust_name on
customer (lower(customer_name))

```

Este índice puede ser útil en consultas que realicen la búsqueda de personas por su nombre sin importar si el nombre está en minúsculas o en mayúsculas.

Al incorporar la función `lower`, las etiquetas del índice se pasarán a minúsculas. De esta forma se obtiene el beneficio esperado.

Para que la consulta haga uso del índice de forma adecuada, la condición debe incluir exactamente la misma expresión empleada al crear el índice:

```
select * from cliente where lower(customer_name) = 'paco'
```

7.9.6. Generando índices en un tablespace distinto:

```
create unique index idx_cust on customer(customer_id) tablespace index01
```

En este ejemplo, el espacio en disco que genera el índice se almacenará en el tablespace `index01`

7.9.7. Tipos de índices en Oracle.

En general, Oracle ofrece los siguientes tipos de índices. Algunos de ellos no son revisados en este curso.

- B – tree indexes (revisado en el curso).
 - Index Organized Indexes
 - Reverse Key Indexes
 - Descending indexes
 - B-tree cluster indexes
- Bitmap and bitmap join indexes (Revisados en el curso).
 - Function-based indexes (Revisado anteriormente).
 - Application domain indexes

7.9.8. Convenciones de nombrado de índices.

Tipo de índice	Convención
Non unique	<prefijo_tabla>_<nombre_campo>_ix Si en índice es compuesto, usar <nombre_grupo> en lugar de <nombre_campo>
Unique	<prefijo_tabla>_<nombre_campo>_iuk Si en índice es compuesto, usar <nombre_grupo> en lugar de <nombre_campo>

7.10. VISTAS

- Comúnmente llamadas “tablas virtuales”. Las vistas permiten seleccionar registros y columnas de otras tablas y mostrarlas como si los datos estuvieran asociados a la misma tabla.

- Las vistas no requieren almacenamiento en disco.
- La definición de la vista se guarda como una sentencia compilada que obtiene los datos de manera dinámica.

Sintaxis SQL estándar:

```
create view <view name> [(<column_name>, . . .)]
as <query_expression>
[with [cascaded|local] check option]
```

Sintaxis Oracle:

```
create [or replace]
[[no] force]
[ editioning | editionable [ editioning ] | noneditionable ]
view [schema.] view
[ ( { alias [ visible | invisible ] [ inline_constraint... ]
  | out_of_line_constraint
  }
  [, { alias [ visible | invisible ] [ inline_constraint... ]
  | out_of_line_constraint
  }
  ]
)
| object_view_clause
| xmltype_view_clause
]
[ bequeath { current user | definer } ]
as subquery [ subquery_restriction_clause ] ;
```

Ejemplo:

Crear una vista V_EMPLEADO que contenga el nombre, apellidos y clave del puesto de un empleado.

A. Definición de la tabla PUESTO

```
create table puesto(
    puesto_id          number(10,0)    not null,
    clave_puesto       varchar2(10)     not null,
    funcion_principal   varchar2(255)    not null,
    sueldo_tabulador    number(8, 2)     not null,
    constraint puesto_pk primary key (puesto_id)
);
```

B. Definición de la tabla CONYUGE

```
create table conyuge(
    conyuge_id          number(10, 0)    not null,
```

```

nombre          varchar2(40)      not null,
apellido_paterno varchar2(40)      not null,
apellido_materno varchar2(40),
constraint conyuge_pk primary key (conyuge_id)
);

```

C. Definición de la tabla EMPLEADO

```

create table empleado(
    empleado_id      number(10, 0)    not null,
    nombre           varchar2(40)      not null,
    apellido_paterno varchar2(40)      not null,
    apellido_materno varchar2(40),
    fecha_nacimiento date              not null,
    conyuge_empleado_id number(10, 0),
    jefe_inmediato   number(10, 0),
    conyuge_id       number(10, 0),
    puesto_id        number(10, 0)    not null,
    constraint empleado_pk primary key (empleado_id),
    constraint jefe_inmemdiado_fk foreign key (jefe_inmediato)
    references empleado(empleado_id),
    constraint conyuge_emp_fk foreign key (conyuge empleado_id)
    references empleado(empleado_id),
    constraint conyuge_fk foreign key (conyuge_id)
    references conyuge(conyuge_id),
    constraint puesto_fk foreign key (puesto_id)
    references puesto(puesto_id)
);

```

D. Crear la vista.

```

create or replace view v_empleado(
    nombre, apellido_paterno, apellido_materno, clave_puesto
) as select nombre, apellido_paterno, apellido_materno, clave_puesto
from empleado e, puesto p
where e.puesto_id=p.puesto_id;

```

Observar que los campos de la vista, corresponden a una combinación entre los campos de EMPLEADO y los de PUESTO. Se emplea una consulta `select` que define la forma en la que se extraen los datos.

Ejemplo:

Inserción de un puesto y del empleado

```

insert into puesto
(puesto_id, clave_puesto, funcion_principal, sueldo_tabulador)
values (1, 'dg', 'direccion de sistemas', 80000);

```

```

insert into empleado (empleado_id, nombre, apellido_paterno,
    apellido_materno, fecha_nacimiento, puesto_id)

```

```
values (1, 'juan', 'martinez', 'lopez',
        to_date('1980/01/10 10:40:00', 'yyyy/mm/dd hh24:mi:ss'), 1);
```

Leyendo el contenido de la vista:

```
select * from v_empleado;
```

NOMBRE	APELLIDO_PATERNO	APELLIDO_MATERNO	CLAVE_PUESTO
JUAN	MARTINEZ	LOPEZ	DG

7.10.1. Otros beneficios del uso de las vistas:

- Proporcionan niveles adicionales de seguridad ya que pueden ser empleadas para restringir el acceso a ciertas columnas de una o varias tablas. Por ejemplo, suponer que se tiene una tabla llamada *Cliente*, que contiene datos privados o delicados como son: *password*, *num_tarjeta*. Suponer que se desea dar permisos de lectura a un usuario a todos los atributos excepto a estas 2 columnas. Una manera de implementar este requerimiento es:
 - El usuario no tiene permisos para leer directamente la tabla *Cliente*.
 - Se crea una vista *V_CLIENTE* que contiene únicamente las columnas permitidas y se le da acceso de lectura al usuario.
- Como se revisó anteriormente, la vista es útil para ocultar la complejidad de una consulta que involucran múltiples tablas, en especial, ocultar el código requerido para ligar las tablas.
- Pueden ser empleadas para ocultar a una aplicación cambios a las tablas, Por ejemplo, la definición de la tabla puede cambiar, pero la definición de la vista no. De esta forma las aplicaciones no se verán afectadas. Solo bastará con actualizar la definición de la vista en caso de ser necesario.

7.10.2. Otras funcionalidades de las vistas (no revisadas en este curso)

- Es posible hacer operaciones DML a partir de una vista: inserciones, actualizaciones, siempre y cuando la vista cumpla con ciertos requisitos.
- Algunos tipos de vistas en Oracle:
 - Updatable Join Views
 - Object views
 - Materialized views: Empleadas en replicación de datos y en Data warehouses

7.11. SINÓNIMOS

Un sinónimo es un nombre alternativo que se emplea para hacer referencia a algún otro objeto de la base de datos: tablas, vistas, secuencias, procedimientos almacenados, vistas materializadas, clases Java, objetos definidos por el usuario, y otro sinónimo.

Los sinónimos pueden ser empleados en clausulas como: *select*, *insert*, *update*, *delete*, *flashback table*, *explain plan*, y *lock table*

Sintaxis:

```
create [ or replace ] [ editionable | noneditionable ]
  [ public ] synonym
  [ schema. ] synonym
for [ schema. ] object [ @ dblink ] ;
```

7.11.1. Usos principales:

- Permiten simplificar sentencias SQL al emplear nombres más cortos.
- En bases de datos distribuidas permiten implementar el concepto de transparencia de localización.
- Un uso muy común es para ocultar el dueño o nombre del esquema de un objeto cuando este es accedido por otro usuario B (asumiendo que el usuario B tiene los permisos adecuados).

Ejemplo:

- Suponer que se tienen 2 usuarios: `admin` y `guest`. El usuario `admin` define o tiene una tabla llamada `Cliente`.
- El usuario `admin` otorga permisos de lectura al usuario `guest` para que pueda consultar la tabla `Cliente`.
- Una vez otorgados los permisos, el usuario `guest` podrá consultar la tabla de la siguiente manera:

```
select *
from admin.cliente;
```

- Observar que el usuario `guest` debe escribir el nombre del esquema o dueño. Por default, si un objeto no especifica su dueño, el manejador solo buscará al objeto en el esquema del dueño. De no escribir el nombre del esquema, el manejador generará un mensaje de error indicando que el objeto no existe.
- Escribir los nombres de los esquemas suele ser tedioso y aumenta la probabilidad de errores.
- Para resolver el inconveniente, el usuario `guest` puede crear un sinónimo de la siguiente manera:

```
create or replace private synonym cliente for admin.cliente;
```

- Antes de ejecutar la sentencia, el usuario `guest` debe contar con el privilegio `create synonym`
- De esta forma el usuario `guest` puede reescribir la sentencia anterior de la siguiente forma:

```
select *
from cliente;
```

7.11.2. Sinónimos públicos y privados.

- Un sinónimo público es aquel que se crea para uso global de todos los usuarios. Su dueño es un grupo de usuarios llamado `PUBLIC`. Para que un usuario pueda crear un sinónimo público debe contar con el privilegio `create public synonym`
- Un sinónimo privado, como su nombre lo indica, es un objeto que estará disponible solo para el usuario que lo genera. Para que un usuario pueda crear sinónimos privados, deberá contar con el privilegio `create synonym`

Ejemplo:

```
create or replace public synonym cliente for admin.cliente;
```

Con base a lo anterior se pueden tener los siguientes escenarios para el ejemplo mencionado:

- Si el usuario `admin` tiene privilegios para crear sinónimos públicos, puede crear uno para que el usuario `guest` pueda hacer uso de él y consultar a la tabla `cliente`.

- Si el usuario `quest` cuenta con privilegios para crear sinónimos privados, puede crear uno para su uso propio y consultar a la tabla `cliente`.
- Los 2 escenarios son válidos, siempre y cuando el usuario `admin` otorgue privilegios para poder leer el contenido de su tabla `cliente` con la siguiente instrucción:

```
grant select on empleado to quest;
```

7.12. REFERENCIAS.

El contenido de este documento se basa en las fuentes bibliográficas de la asignatura y en la documentación oficial de Oracle 12c : <http://docs.oracle.com/database/121/CNCPT/>