


FACULTAD DE CIENCIAS Y TECNOLOGÍA

Ingeniería en Ciencias de la Computación
Ingeniería en Tecnologías de la Información y Seguridad




Capítulo VI

SQL

Structured Query Language

Ing. Edgar T. Espinoza R.

Principales tipos de objetos.

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

Sintaxis create statement SQL estándar



La definición de tablas varía considerablemente entre manejador y manejador.

SQL estandar

```
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.

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.

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]`

Ejemplo:

Crear siguiente tabla empleado empleando sintaxis SQL estándar y en Oracle En SQL estándar:



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

SQL estandar

```
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  
);
```

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)  
);
```

.Organización de almacenamiento.



Para agregar comentarios a una tabla:

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

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

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

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.

.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.


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 delete 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).



CREACION 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*.

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

Valores por default en una columna:



El manejador puede asignar valores por default (valores por omisión) a una columna para los casos en los 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  
);
```

Ejemplos:



```
Insert into product (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

CREACION DE CONSTRAINTS.



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, así como la convención de nombrado propuesta para el curso



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: <code>create table empleado(num_empleado number not null);</code> 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, 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**.

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 en 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}

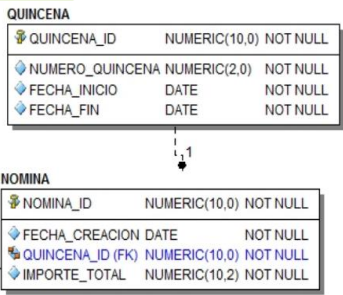
Observar, que los nombres que genera Oracle para identificar a un constraint inician con la palabra *SYS*.

Llave foránea - Sintaxis Oracle:

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*:



```
create table quincena (  
    quincena_id number(10,0) constrain t 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)  
);
```



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**.

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.

Sintaxis en SQL estándar:

```
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'))  
);
```



Creando FK con un constraint a nivel de 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)  
);
```

Ejemplo:



1. Crear una tabla con las siguientes especificaciones, emplear columna constraints:

- ❑ *Observar que ya se especifica el atributo que actuara como PK.*
- ❑ *El nivel del puesto solo debe tener valores 'a','b','c'.*
- ❑ *La clave no debe deplucarse.*
- ❑ *El sueldo nodebe exceder los Bs. 10000.*

2. Crear la tabla Empleado considerando la FK puesto_id, emplear column constraints

3. Aplicando la tecnica de Table Constraints, reescribir el codigo anterior,modificar el nombre de los constraints y de las tablas con base al siguiente diagrama:

4. Crear la tabla Empleado considerando la FK puesto_id, emplear column constraints

¿Preguntas?

