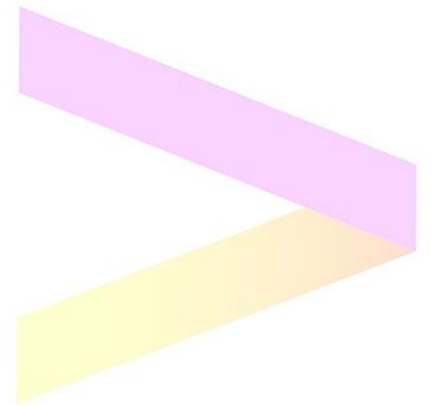


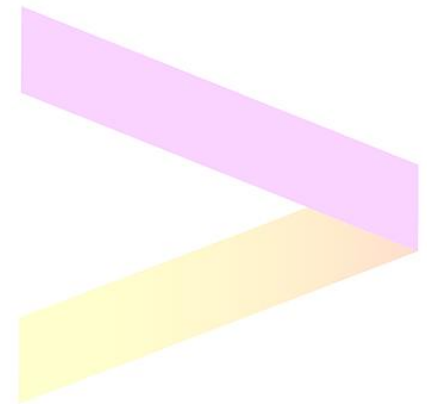
# **Clase 6**

# **SQL Básico**



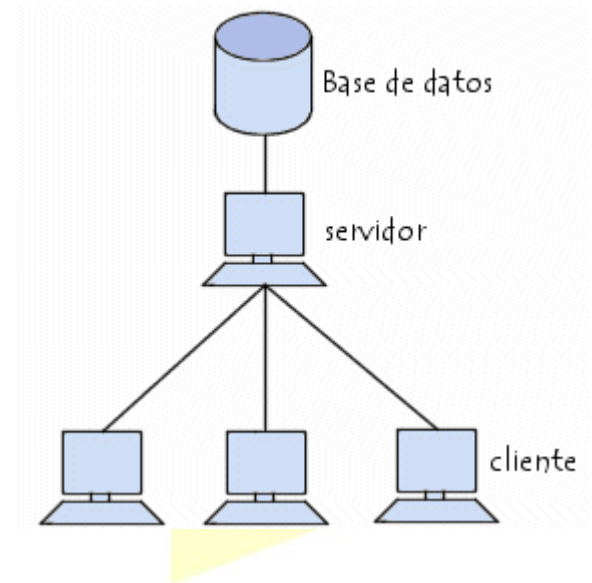
# Temario

- 01-Introduccion a Base de Datos
- 02-Tablas
- 03-Insert
- 04-Select
- 05-Where
- 06-Operadores
- 07-Delete
- 08-Update
- 09-Null
- 10-Is null
- 11-Introduccion a restricciones
- 12-Primary Key
- 13-Unique
- 14-Check
- 15-Foreign Key
- 16-Reglas de Integridad
- 17-Truncate
- 18-Alias
- 19-Order By
- 20-Between
- 21-In
- 22-Like
- 23-Count
- 24-Group By
- 25-Having
- 26-Distinct
- 27-Join
- 28-Union
- 29-Intersection
- 30-Minus
- 31-Subconsultas
- 32-Vistas



# 01 - Introducción a Base de Datos

- Una base de datos es una colección de información organizada de forma que un programa de ordenador pueda seleccionar rápidamente los fragmentos de datos que necesite.
- Las bases de datos tradicionales se organizan por campos, registros y archivos. Un campo es una pieza única de información; un registro es un sistema completo de campos



- Una base de datos almacena su información en tablas, que es la unidad básica de almacenamiento.
- Una tabla es una estructura de datos que organiza los datos en columnas y filas; cada columna es un campo (o atributo) y cada fila, un registro. La intersección de una columna con una fila, contiene un dato específico, un solo valor.

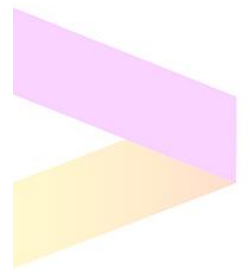
The diagram illustrates a database table structure. A table with 5 rows and 5 columns is shown. The columns are labeled 'Cve. cliente', 'Nombre', 'Direccion', 'Ciudad', and 'Estado'. The rows contain data for five different clients. Annotations include: a red bracket above the table labeled 'tabla'; a red box around the first column labeled 'campo clave'; a blue box around the second and third columns labeled 'campo'; a blue arrow pointing to the second row labeled 'registro'; and a red arrow pointing to the first column labeled 'campo clave'.

Cve. cliente	Nombre	Direccion	Ciudad	Estado
1	Alfredo Godínez	Fresnillo #47	Veracruz	Veracruz
2	Gabriela Mora	El crespo #81	Guadalajara	Jalisco
3	Alejandra Avalos	Casa Mata #1	Morelia	Michoacan
4	Jaime Quintero	Miraflores #23	Uruapan	Michoacan
5	Carlos Miranda	Rio Bravo #95	Matamoros	Tamaulipas

- Cada registro contiene un dato por cada columna de la tabla. Cada campo (columna) debe tener un nombre. El nombre del campo hace referencia a la información que almacenará.
- Cada campo (columna) también debe definir el tipo de dato que almacenará.

### Libros

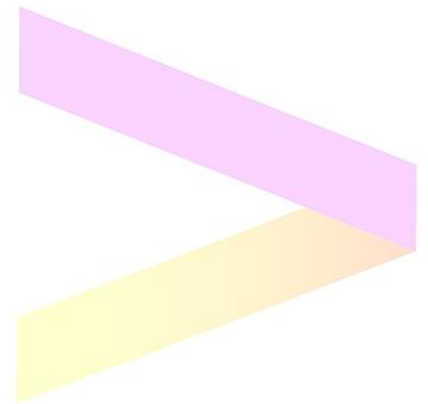
ISBN	Título	Nombre Autor	Apellido Autor	Precio
978-0062511409	El Alquimista	Paulo	Coelho	\$8.46
978-0307744593	Aleph	Paulo	Coelho	\$12.23
978-034580704	El peregrino	Paulo	Coelho	\$12.20



## 02 - TABLAS

- Las tablas forman parte de una base de datos.
- Al crear una tabla debemos resolver qué campos (columnas) tendrá y que tipo de datos almacenarán cada uno de ellos, es decir, su estructura.
- La sintaxis básica y general para crear una tabla es la siguiente:

```
create table NOMBRETABLA(  
  NOMBRECAMPO1 TIPODEDATO,  
  ...  
  NOMBRECAMPON TIPODEDATO  
);
```



Ejemplo:

- Creamos una tabla llamada "usuarios" y entre paréntesis definimos los campos y sus tipos:

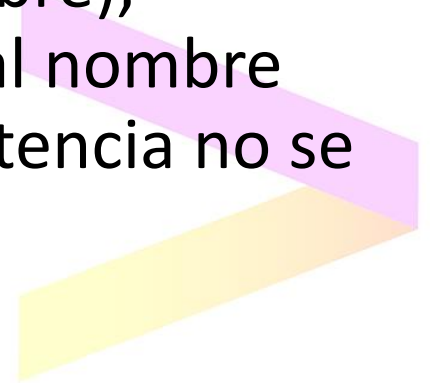
```
create table usuarios(  
  nombre varchar2(30),  
  clave varchar2(10)  
);
```

- Cada campo con su tipo debe separarse con comas de los siguientes, excepto el último.
- Cuando se crea una tabla debemos indicar su nombre y definir al menos un campo con su tipo de dato.
- Cada usuario ocupará un registro de esta tabla, con su respectivo nombre y clave.



## Consideraciones:

- Para nombres de tablas, se puede utilizar cualquier carácter permitido para nombres de directorios, el primero debe ser un carácter alfabético y no puede contener espacios. La longitud máxima es de 30 caracteres.
- Si intentamos crear una tabla con un nombre ya existente (existe otra tabla con ese nombre), mostrará un mensaje indicando que a tal nombre ya lo está utilizando otro objeto y la sentencia no se ejecutará.





# accenture

## DROP TABLE

- Para eliminar una tabla usamos "drop table" junto al nombre de la tabla a eliminar:

```
drop table NOMBRETABLA;
```

- En el siguiente ejemplo eliminamos la tabla "usuarios":

```
drop table usuarios;
```

- Si intentamos eliminar una tabla que no existe, aparece un mensaje de error indicando tal situación y la sentencia no se ejecuta.



# ALTER TABLE

- "alter table" permite modificar la estructura de una tabla. Podemos utilizarla para agregar, modificar y eliminar campos de una tabla.
- Para agregar un nuevo campo a una tabla empleamos la siguiente sintaxis básica:

```
alter table NOMBRETABLA  
add NOMBRENUEVOCAMPO DEFINICION;
```

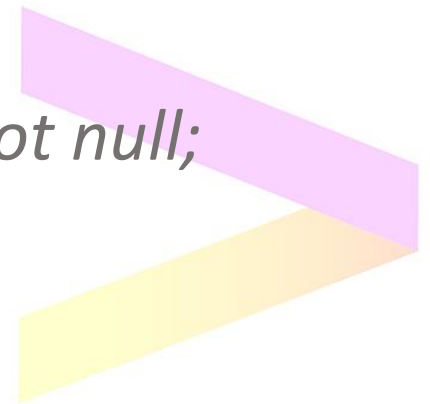


## Ejemplo:

- En el siguiente ejemplo agregamos el campo "cantidad" a la tabla "libros", de tipo number(4), con el valor por defecto cero y que NO acepta valores nulos:

*alter table libros*

***add** cantidad number(4) default 0 not null;*



- Para modificar un campo empleamos la siguiente sintaxis:

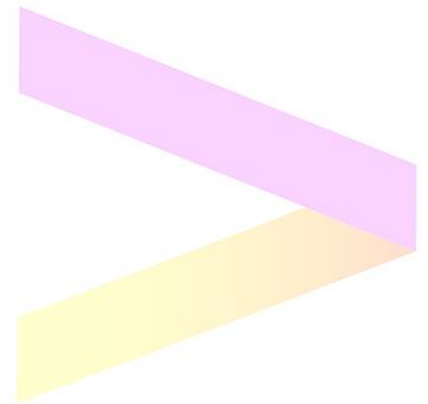
*alter table NOMBRETABLA*

*modify NOMBRECAMPO NUEVADEFINICION;*

- En el siguiente ejemplo modificamos el campo "precio" de la tabla "libros" para que tome valores de 6 dígitos incluyendo 2 decimales y no acepte valores nulos:

*alter table libros*

*modify precio number(6,2) not null;*



- Para eliminar campos de una tabla la sintaxis básica es la siguiente:

*alter table NOMBRETABLA*

*drop column NOMBRECAMPO;*

- En el siguiente ejemplo eliminamos el campo "precio" de la tabla "libros":

*alter table libros*

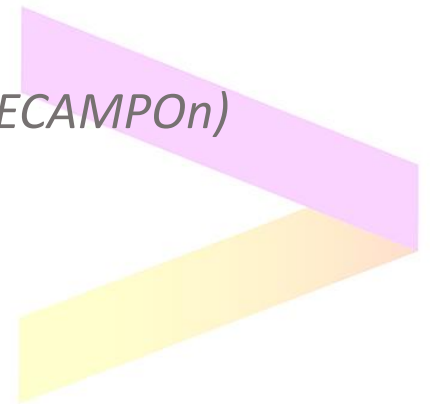
*drop column precio;*

- No pueden eliminarse los campos a los cuales hace referencia una restricción "foreign key".

## 03 - INSERT

- Un registro es una fila de la tabla que contiene los datos propiamente dichos. Cada registro tiene un dato por cada columna (campo). Nuestra tabla "usuarios" consta de 2 campos, "nombre" y "clave".
- Al ingresar los datos de cada registro debe tenerse en cuenta la cantidad y el orden de los campos.
- La sintaxis básica y general es la siguiente:

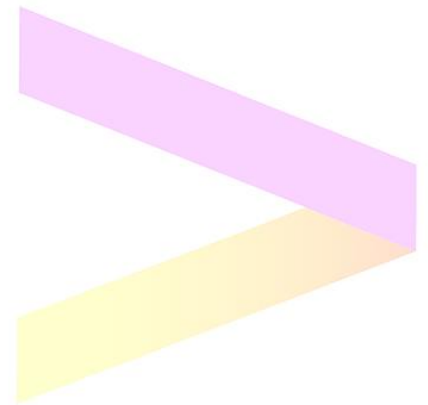
```
insert into NOMBRETABLA (NOMBRECAMPO1, ..., NOMBRECAMPOn)  
values (VALORCAMPO1, ..., VALORCAMPOn);
```



## Ejemplo

- En el siguiente ejemplo se agrega un registro a la tabla "usuarios", en el campo "nombre" se almacenará "Horacio" y en el campo "clave" se guardará "payaso":

```
insert into usuarios (nombre, clave)  
values ('Javier', 'payaso');
```



# accenture

- Es importante ingresar los valores en el mismo orden en que se nombran los campos: En el siguiente ejemplo se lista primero el campo "clave" y luego el campo "nombre" por eso, los valores también se colocan en ese orden:

```
insert into usuarios (clave, nombre)  
values ('River','Juan');
```

- Si ingresamos los datos en un orden distinto al orden en que se nombraron los campos, no aparece un mensaje de error y los datos se guardan de modo incorrecto.





## 04 - SELECT

- Para ver los registros de una tabla usamos "select".
- La sintaxis básica y general es la siguiente:


*select \* from NOMBRETABLA;*

- El asterisco (\*) indica que se seleccionan todos los campos de la tabla.



- Podemos especificar el nombre de los campos que queremos ver, separándolos por comas:

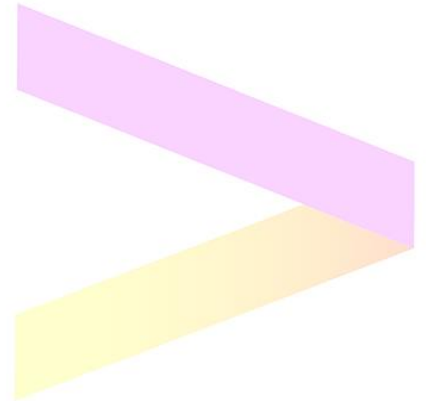
*select titulo, autor from libros;*

- La lista de campos luego del "select" selecciona los datos correspondientes a los campos nombrados. En el ejemplo anterior seleccionamos los campos "titulo" y "autor" de la tabla "libros", mostrando todos los registros.
- 
- A decorative graphic in the bottom right corner consisting of two overlapping chevron shapes pointing to the right. The top shape is light purple and the bottom shape is light yellow.

## 05 - WHERE

- Existe una cláusula, "where" con la cual podemos especificar condiciones para una consulta "select". Es decir, podemos recuperar algunos registros, sólo los que cumplan con ciertas condiciones indicadas con la cláusula "where". Por ejemplo, queremos ver el usuario cuyo nombre es "Marcelo", para ello utilizamos "where" y luego de ella, la condición:

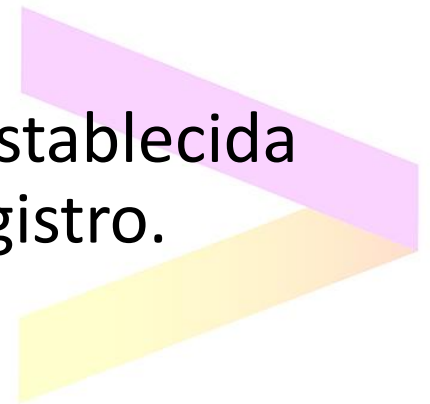
```
select nombre, clave  
from usuarios  
where nombre='Marcelo';
```



- La sintaxis básica y general es la siguiente:

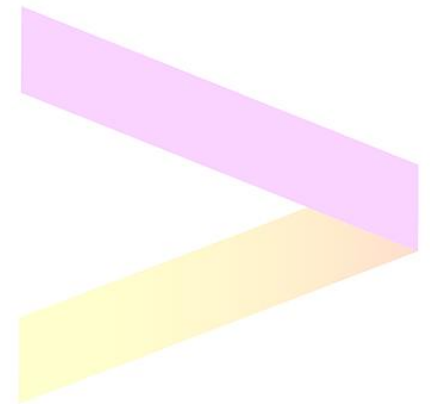
```
select NOMBRECAMPO1, ..., NOMBRECAMPOn  
from NOMBRETABLA  
where CONDICION;
```

- Si ningún registro cumple la condición establecida con el "where", no aparecerá ningún registro.



## 06 - Operadores

- Los operadores son símbolos que permiten realizar operaciones matemáticas, concatenar cadenas, hacer comparaciones.
- Oracle reconoce de 4 tipos de operadores:
  - A) relacionales (o de comparación)
  - B) aritméticos
  - C) de concatenación
  - D) lógicos



## A) Relacionales

- Los operadores relacionales (o de comparación) nos permiten comparar dos expresiones, que pueden ser variables, valores de campos, etc.
- Hemos aprendido a especificar condiciones de igualdad para seleccionar registros de una tabla; por ejemplo:

```
select *from libros  
where autor='Borges';
```

- Utilizamos el operador relacional de igualdad.

- Los operadores relacionales vinculan un campo con un valor para que Oracle compare cada registro (el campo especificado) con el valor dado.
- Los operadores relacionales son los siguientes:

= igual

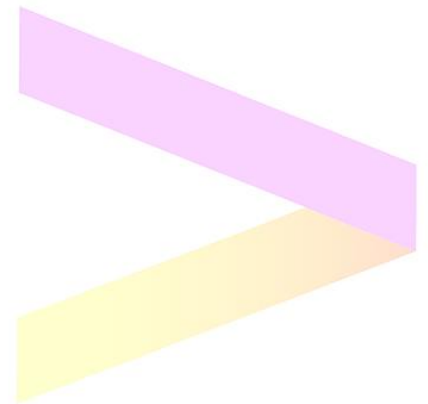
<> distinto

> mayor

< menor

>= mayor o igual

<= menor o igual



- Podemos seleccionar los registros cuyo autor sea diferente de "Borges", para ello usamos la condición:

```
select * from libros  
where autor<>'Borges';
```

- Podemos comparar valores numéricos. Por ejemplo, queremos mostrar los títulos y precios de los libros cuyo precio sea mayor a 20 pesos:

```
select titulo, precio  
from libros  
where precio>20;
```

- Queremos seleccionar los libros cuyo precio sea menor o igual a 30:

```
select *from libros  
where precio<=30;
```

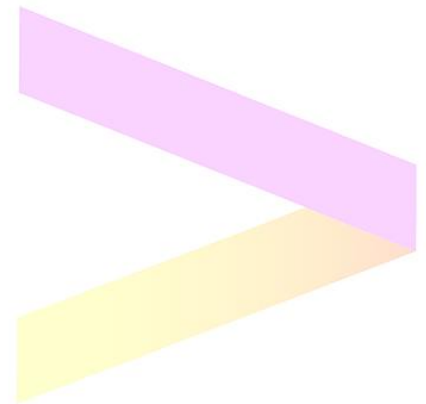




## B) Operador Aritmético

- Los operadores aritméticos permiten realizar cálculos con valores numéricos.
- Son: multiplicación (\*), división (/), suma (+) y resta (-).
- Si queremos saber el precio de cada libro con un 10% de descuento podemos incluir en la sentencia los siguientes cálculos:

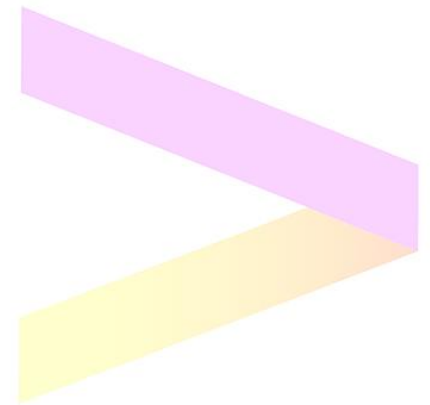
```
select titulo,precio,  
precio-(precio*0.1)  
from libros;
```



## C) Operador de Concatenación

- El operador que se utiliza para concatenar es el string '||' (doble barra vertical).

```
SELECT 'Mi nombre completo es ' || nombre || ' ' ||  
apellidos  
FROM usuarios WHERE id_usuario=8;
```



## D) Operadores Logicos

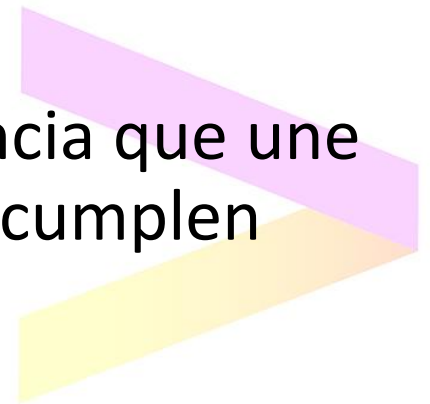
- Los operadores lógicos son los siguientes:
  - and, significa "y",
  - or, significa "y/o",
  - not, significa "no", invierte el resultado
  - (), paréntesis
- Los operadores lógicos se usan para combinar condiciones.



- Si queremos recuperar todos los libros cuyo autor sea igual a "Borges" y cuyo precio no supere los 20 pesos, necesitamos 2 condiciones:

```
select *from libros  
where (autor='Borges') and  
(precio<=20);
```

- Los registros recuperados en una sentencia que une dos condiciones con el operador "and", cumplen con las 2 condiciones.



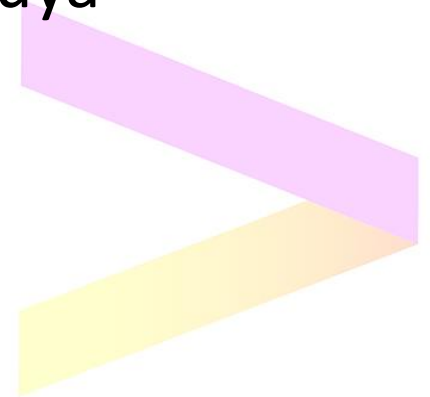
# accenture

- Queremos ver los libros cuyo autor sea "Borges" y/o cuya editorial sea "Planeta":

```
select *from libros  
where autor='Borges' or  
editorial='Planeta';
```

- Queremos recuperar los libros que NO cumplan la condición dada, por ejemplo, aquellos cuya editorial NO sea "Planeta":

```
select *from libros  
where not editorial='Planeta';
```



## 07 - Delete

- Para eliminar los registros de una tabla usamos el comando "delete".
- Sintaxis básica:

*delete from NOMBRETABLA;*

- Por ejemplo, queremos eliminar aquel registro cuyo nombre de usuario es "Marcelo":

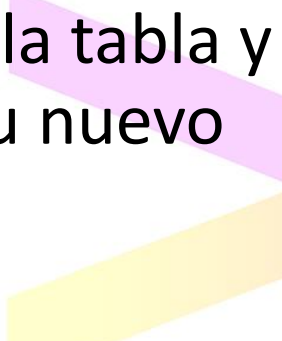
*delete from usuarios  
where nombre='Marcelo';*



## 08 - UPDATE

- Decimos que actualizamos un registro cuando modificamos alguno de sus valores.
- Para modificar uno o varios datos de uno o varios registros utilizamos "update" (actualizar).
- Sintaxis básica:

*update NOMBRETABLA set CAMPO=NUEVOVALOR;*

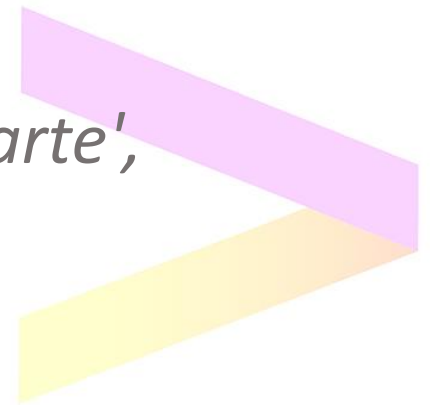
- Utilizamos "update" junto al nombre de la tabla y "set" junto con el campo a modificar y su nuevo valor.
  - El cambio afectará a todos los registros.
- 
- A large, stylized arrow pointing to the right. The arrow is composed of two overlapping shapes: a light purple one on top and a light yellow one on the bottom.

- Usando WHERE:

```
update usuarios set clave='Boca'  
where nombre='Federicolopez';
```

- También podemos actualizar varios campos en una sola instrucción:

```
update usuarios set nombre='Marceloduarte',  
clave='Marce'  
where nombre='Marcelo';
```



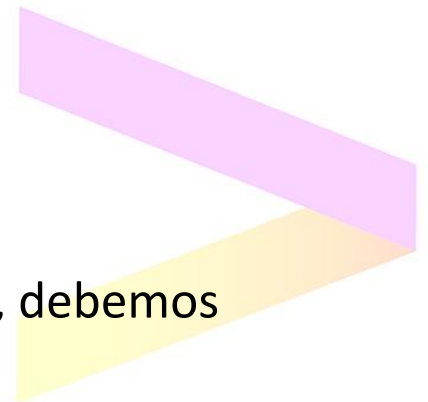


## 09 - NULL

- "null" significa "dato desconocido" o "valor inexistente".
- A veces, puede desconocerse o no existir el dato correspondiente a algún campo de un registro. En estos casos decimos que el campo puede contener valores nulos.
- Veamos un ejemplo. Tenemos nuestra tabla "libros". El campo "titulo" no debería estar vacío nunca, igualmente el campo "autor". Para ello, al crear la tabla, debemos especificar que tales campos no admitan valores nulos:

```
create table libros(  
  titulo varchar2(30) not null,  
  autor varchar2(20) not null,  
  editorial varchar2(15) null,  
  precio number(5,2) );
```

- Para especificar que un campo NO admita valores nulos, debemos colocar "not null" luego de la definición del campo.



## 10 – IS NULL

- Para recuperar los registros que contengan el valor "null" en algún campo, no podemos utilizar los operadores relacionales vistos anteriormente: = (igual) y <> (distinto); debemos utilizar los operadores "is null" (es igual a null) y "is not null" (no es null).
- Con la siguiente sentencia recuperamos los libros que contienen valor nulo en el campo "editorial":

```
select *from libros  
where editorial is null;
```



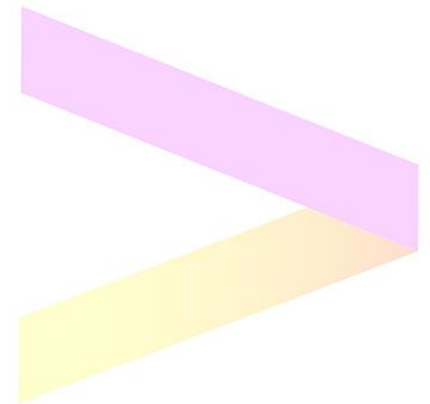
# 11 - Restricciones

- Para asegurar la integridad de los datos almacenados en nuestras tablas, podemos crear restricciones. Estas restricciones las podemos implementar al momento de crear nuestras tablas o de modificarlas, también es necesario señalar que dichas restricciones son objetos propios de la base de datos y por lo tanto requieren de un nombre único compuesto del nombre del esquema al que pertenece y el nombre que lo identifica.
- Podemos agregar un campo a una tabla y en el mismo momento aplicarle una restricción.
- Para agregar un campo y establecer una restricción, la sintaxis básica es la siguiente:

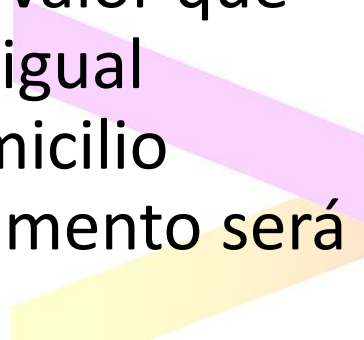
*alter table TABLA*

*add CAMPO DEFINICION*

*constraint NOMBRERESTRICCION TIPO;*



## 12 – Clave Primaria

- Una clave primaria es un campo (o varios) que identifica un solo registro (fila) en una tabla.
  - Para un valor del campo clave existe solamente un registro.
  - Veamos un ejemplo, si tenemos una tabla con datos de personas, el número de documento puede establecerse como clave primaria, es un valor que no se repite; puede haber personas con igual apellido y nombre, incluso el mismo domicilio (padre e hijo por ejemplo), pero su documento será siempre distinto.
- 
- A decorative graphic in the bottom right corner consisting of two overlapping triangles. The top triangle is light purple and the bottom triangle is light yellow, both pointing towards the right.

- Si tenemos la tabla "usuarios", el nombre de cada usuario puede establecerse como clave primaria, es un valor que no se repite; puede haber usuarios con igual clave, pero su nombre de usuario será siempre diferente.
- Podemos establecer que un campo sea clave primaria al momento de crear la tabla o luego que ha sido creada. Vamos a aprender a establecerla al crear la tabla. No existe una única manera de hacerlo, por ahora veremos la sintaxis más sencilla.

Customer ID	Forename	Surname
1	Simon	Jones
2	Emma	Price
3	Laura	Jones
4	Jonathan	Hale
5	Emma	Smith

Simple primary key

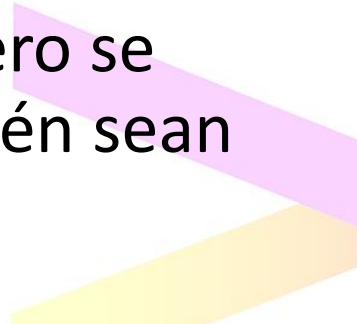
# accenture

- En el siguiente ejemplo definimos una clave primaria, para nuestra tabla "usuarios" para asegurarnos que cada usuario tendrá un nombre diferente y único:

```
create table usuarios(  
  nombre varchar2(20),  
  clave varchar2(10),  
  primary key(nombre) );
```

- Cualquier campo (de cualquier tipo) puede ser clave primaria, debe cumplir como requisito, que sus valores no se repitan ni sean nulos. Por ello, al definir un campo como clave primaria, automáticamente Oracle lo convierte a "not null".

# 13 - UNIQUE

- La restricción "unique" impide la duplicación de claves alternas (no primarias), es decir, especifica que dos registros no puedan tener el mismo valor en un campo. Se permiten valores nulos.
  - Se pueden aplicar varias restricciones de este tipo a una misma tabla, y pueden aplicarse a uno o varios campos que no sean clave primaria.
  - Se emplea cuando ya se estableció una clave primaria (como un número de legajo) pero se necesita asegurar que otros datos también sean únicos y no se repitan (como número de documento).
- 
- A decorative graphic in the bottom right corner consisting of two overlapping chevron shapes pointing to the right. The top chevron is light purple and the bottom one is light yellow.

- La sintaxis general es la siguiente:

*alter table NOMBRETABLA*

*add constraint NOMBRERESTRICCION*

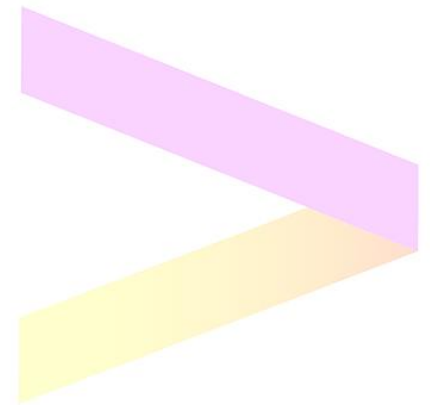
*unique (CAMPO);*

- Ejemplo:

*alter table alumnos*

*add constraint UQ\_alumnos\_documento*

*unique (documento);*





## 14 - CHECK

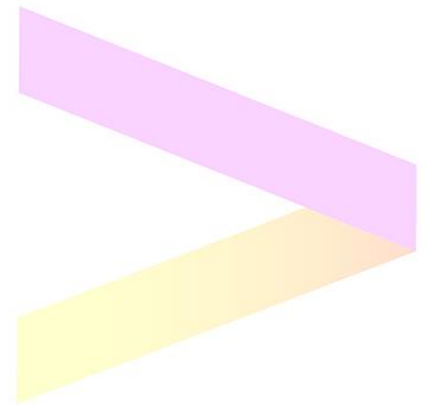
- La restricción "check" especifica los valores que acepta un campo, evitando que se ingresen valores inapropiados.

*La sintaxis básica es la siguiente:*

*alter table NOMBRETABLA*

*add constraint NOMBRECONSTRAINT*

*check CONDICION;*



- Trabajamos con la tabla "libros" de una librería que tiene los siguientes campos: codigo, titulo, autor, editorial, preciomin (que indica el precio para los minoristas) y preciomay (que indica el precio para los mayoristas).
- Los campos correspondientes a los precios (minorista y mayorista) se definen de tipo number(5,2), es decir, aceptan valores entre -999.99 y 999.99. Podemos controlar que no se ingresen valores negativos para dichos campos agregando una restricción "check":

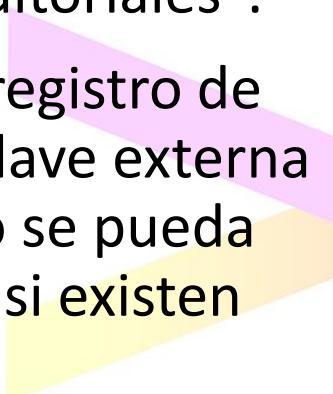
*alter table libros*

*add constraint CK\_libros\_precio\_positivo*

*check (preciomin>=0 and preciomay>=0);*

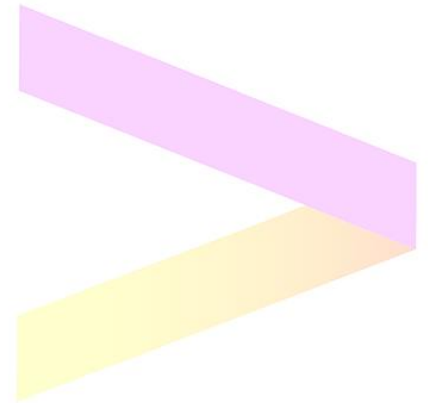
- Este tipo de restricción verifica los datos cada vez que se ejecuta una sentencia "insert" o "update", es decir, actúa en inserciones y actualizaciones.

# 15 – Foreign Key

- Con la restricción "foreign key" se define un campo (o varios) cuyos valores coinciden con la clave primaria de la misma tabla o de otra, es decir, se define una referencia a un campo con una restricción "primary key" o "unique" de la misma tabla o de otra.
  - La integridad referencial asegura que se mantengan las referencias entre las claves primarias y las externas. Por ejemplo, controla que si se agrega un código de editorial en la tabla "libros", tal código exista en la tabla "editoriales".
  - También controla que no pueda eliminarse un registro de una tabla ni modificar la clave primaria si una clave externa hace referencia al registro. Por ejemplo, que no se pueda eliminar o modificar un código de "editoriales" si existen libros con dicho código.
- 
- A decorative graphic in the bottom right corner consisting of two overlapping triangles, one purple and one yellow, pointing towards the right.

- La siguiente es la sintaxis parcial general para agregar una restricción "foreign key":

```
alter table NOMBRETABLA1  
add constraint NOMBRERESTRICCION  
foreign key (CAMPOCLAVEFORANEA)  
references NOMBRETABLA2  
(CAMPOCLAVEPRIMARIA);
```



- Para agregar una restricción "foreign key" al campo "codigoeditorial" de "libros", tipeamos:

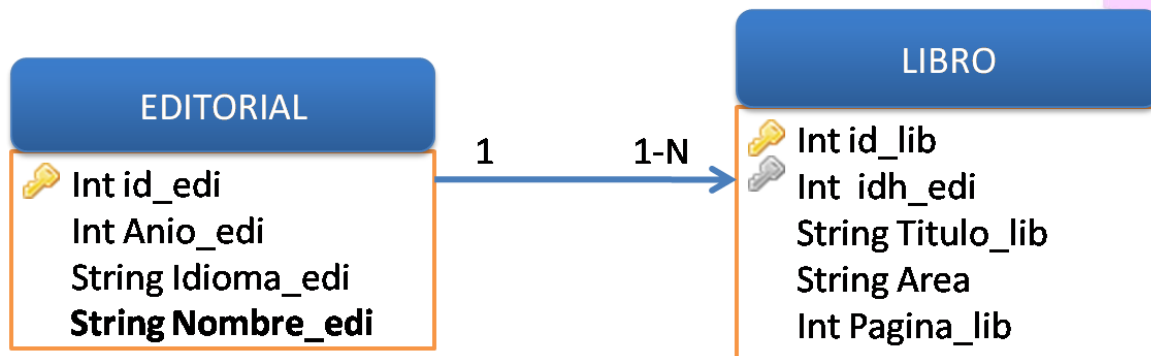
```
alter table libro
```

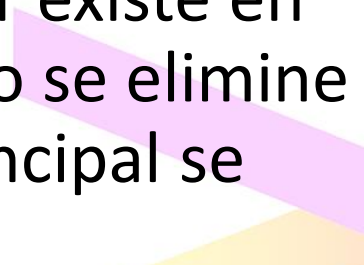
```
add constraint FK_libros_codigoeditorial
```

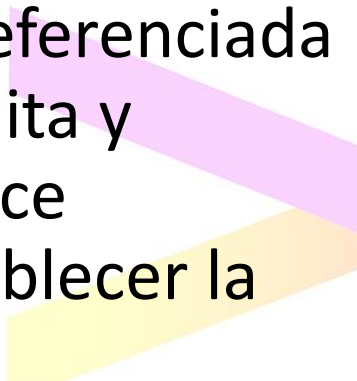
```
foreign key (idh_edi)
```

```
references editoriales(id_edi);
```

## Modelo Relacional

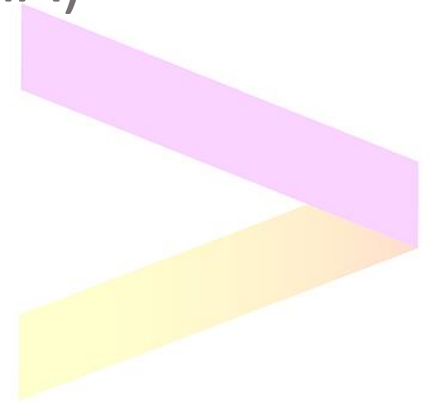


- La restricción "foreign key" tiene la cláusula "on delete", que son opcionales. Esta cláusula especifica cómo debe actuar Oracle frente a eliminaciones en las tablas referenciadas en la restricción.
  - Las opciones para estas cláusulas son las siguientes:
  - - "set null": indica que si eliminamos un registro de la tabla referenciada (TABLA2) cuyo valor existe en la tabla principal (TABLA1), dicho registro se elimine y los valores coincidentes en la tabla principal se seteen a "null".
- 
- A decorative graphic in the bottom right corner consisting of two overlapping chevron shapes pointing right. The top one is light purple and the bottom one is light yellow.

- - "cascade": indica que si eliminamos un registro de la tabla referenciada en una "foreign key" (TABLA2), los registros coincidentes en la tabla principal (TABLA1), también se eliminan; es decir, si eliminamos un registro al cual una clave foránea referencia, dicha eliminación se extiende a la otra tabla (integridad referencial en cascada).
  - - "no action": es la predeterminada; indica que si se intenta eliminar un registro de la tabla referenciada por una "foreign key", Oracle no lo permita y muestre un mensaje de error. Se establece omitiendo la cláusula "on delete" al establecer la restricción.
- 
- A large, stylized arrow pointing to the right, composed of a light purple upper section and a light yellow lower section.

- La sintaxis completa para agregar esta restricción a una tabla es la siguiente:

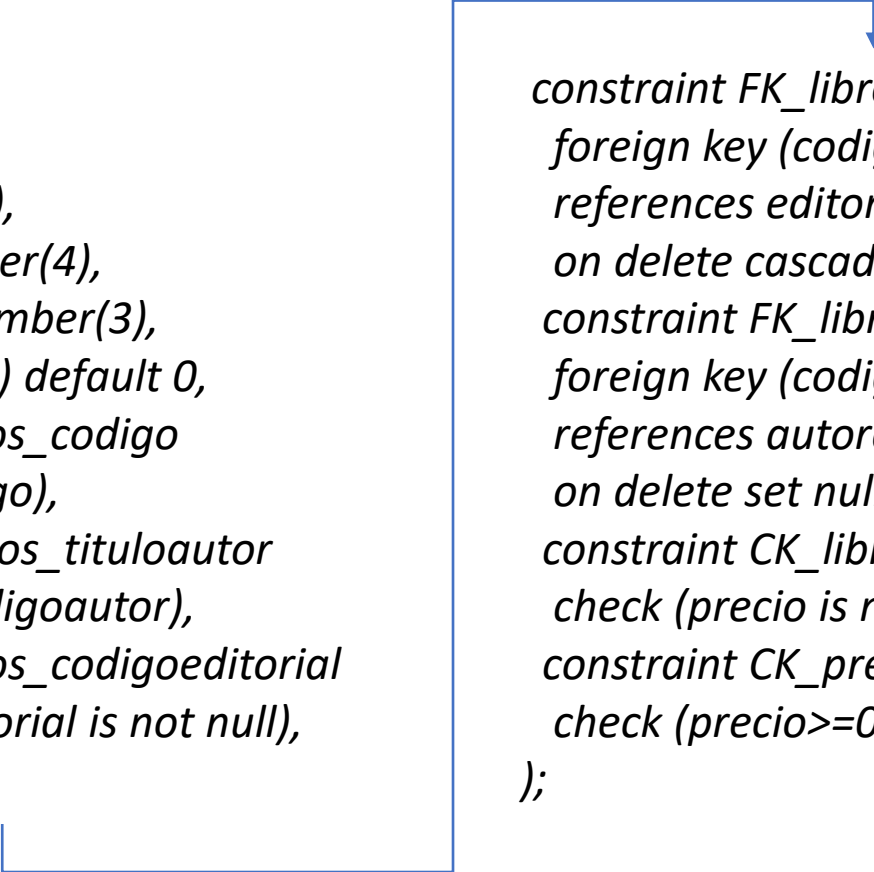
```
alter table TABLA1  
add constraint NOMBRERESTRICCION  
foreign key (CAMPOCLAVEFORANEA)  
references TABLA2(CAMPOCLAVEPRIMARIA)  
on delete OPCION;
```






# Ejemplo de repaso

```
create table libros(  
  codigo number(5),  
  titulo varchar2(40),  
  codigoautor number(4),  
  codigoeditorial number(3),  
  precio number(5,2) default 0,  
  constraint PK_libros_codigo  
    primary key (codigo),  
  constraint UQ_libros_tituloautor  
    unique (titulo,codigoautor),  
  constraint CK_libros_codigoeditorial  
    check (codigoeditorial is not null),
```



```
  constraint FK_libros_editorial  
    foreign key (codigoeditorial)  
      references editoriales(codigo)  
      on delete cascade,  
  constraint FK_libros_autores  
    foreign key (codigoautor)  
      references autores(codigo)  
      on delete set null,  
  constraint CK_libros_precioonulo  
    check (precio is not null) disable,  
  constraint CK_precio_positivo  
    check (precio>=0)  
);
```



# 16 - Reglas de Integridad

- **Integridad de dominio:**
  - **Datos Requeridos:** establece que una columna tenga un valor no NULL. Se define efectuando la declaración de una columna es NOT NULL cuando la tabla que contiene las columnas se crea por primera vez, como parte de la sentencia CREATE TABLE.
  - **Chequeo de Validez:** cuando se crea una tabla cada columna tiene un tipo de datos y asegura que solamente los datos del tipo especificado sean ingresados en la tabla.
- **Integridad de entidad:** establece que la clave primaria de una tabla debe tener un valor único para cada fila de la tabla; si no, la base de datos perderá su integridad. Se especifica en la sentencia CREATE TABLE. El DBMS comprueba automáticamente la unicidad del valor de la clave primaria con cada sentencia INSERT Y UPDATE. Un intento de insertar o actualizar una fila con un valor de la clave primaria ya existente fallará.

- **Integridad referencial:** asegura la integridad entre las claves foráneas y primarias (relaciones padre/hijo). Existen cuatro actualizaciones de la base de datos que pueden corromper la integridad referencial:
  - La inserción de una fila hijo se produce cuando no coincide la clave foránea con la clave primaria del padre.
  - La actualización en la clave foránea de la fila hijo, donde se produce una actualización en la clave ajena de la fila hijo con una sentencia UPDATE y la misma no coincide con ninguna clave primaria.
  - La supresión de una fila padre, con la que, si una fila padre -que tiene uno o más hijos- se suprime, las filas hijos quedarán huérfanas.
  - La actualización de la clave primaria de una fila padre, donde si en una fila padre, que tiene uno o más hijos se actualiza su llave primaria, las filas hijos quedarán huérfanas.

## 17 – Truncate Table

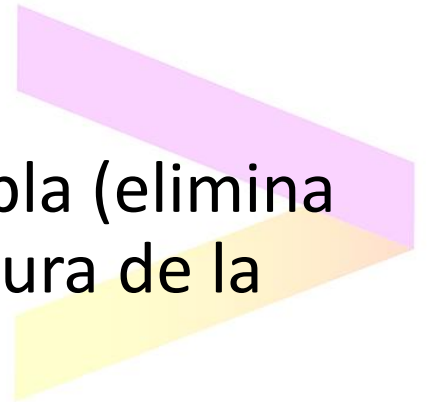
- Aprendimos que para borrar todos los registro de una tabla se usa "delete" sin condición "where".
- También podemos eliminar todos los registros de una tabla con "truncate table". Sintaxis:

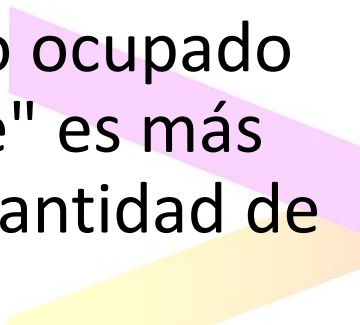
`truncate table NOMBRETABLA;`

- Por ejemplo, queremos vaciar la tabla "libros", usamos:

`truncate table libros;`

- La sentencia "truncate table" vacía la tabla (elimina todos los registros) y conserva la estructura de la tabla.



- La diferencia con "drop table" es que esta sentencia elimina la tabla, no solamente los registros, "truncate table" la vacía de registros.
  - La diferencia con "delete" es la siguiente, al emplear "delete", Oracle guarda una copia de los registros borrados y son recuperables, con "truncate table" no es posible la recuperación porque se libera todo el espacio en disco ocupado por la tabla; por lo tanto, "truncate table" es más rápido que "delete" (se nota cuando la cantidad de registros es muy grande).
- 
- A decorative graphic in the bottom right corner consisting of two overlapping triangles pointing to the right. The top triangle is light purple and the bottom triangle is light yellow.

## 18 - Alias

- Una manera de hacer más comprensible el resultado de una consulta consiste en cambiar los encabezados de las columnas. Por ejemplo, tenemos la tabla "libros" con un campo "cantidad" (entre otros) en el cual se almacena la cantidad de libros en stock; queremos que al mostrar la información de dicha tabla aparezca como encabezado del campo "cantidad" el texto "stock", para ello colocamos un alias de la siguiente manera:

```
select titulo,  
cantidad as stock,  
precio  
from libros;
```

- Para reemplazar el nombre de un campo del encabezado por otro, se coloca la palabra clave "as" seguido del texto del encabezado.



## 19 – Order By

- Podemos ordenar el resultado de un "select" para que los registros se muestren ordenados por algún campo, para ello usamos la cláusula "order by".
- La sintaxis básica es la siguiente:

```
select *from NOMBRETABLA  
order by CAMPO;
```

- Aparecen los registros ordenados alfabéticamente por el campo especificado.



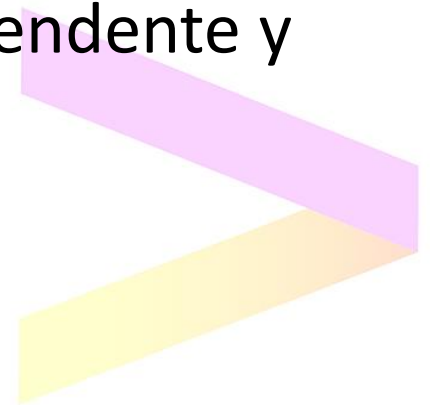
# accenture

- Por defecto, si no aclaramos en la sentencia, los ordena de manera ascendente (de menor a mayor). Podemos ordenarlos de mayor a menor, para ello agregamos la palabra clave "desc":

```
select * from libros  
order by editorial desc;
```

- Incluso, podemos ordenar en distintos sentidos, por ejemplo, por "titulo" en sentido ascendente y "editorial" en sentido descendente:

```
select * from libros  
order by titulo asc, editorial desc;
```





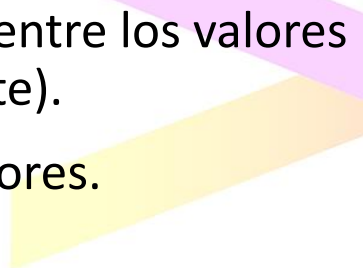
## 20 - BETWEEN

- Hasta ahora, para recuperar de la tabla "libros" los libros con precio mayor o igual a 20 y menor o igual a 40, usamos 2 condiciones unidas por el operador lógico "and":

```
select *from libros  
where precio>=20 and  
precio<=40;
```

- *Podemos usar "between" y así simplificar la consulta:*

```
select *from libros  
where precio between 20 and 40;
```

- Averiguamos si el valor de un campo dado (precio) está entre los valores mínimo y máximo especificados (20 y 40 respectivamente).
  - "between" significa "entre". Trabaja con intervalo de valores.
  - Este operador no tiene en cuenta los valores "null".
- 
- A large, stylized arrow pointing to the right, composed of two overlapping triangles. The top triangle is light purple and the bottom triangle is light yellow.

## 21 - IN

- Se utiliza "in" para averiguar si el valor de un campo está incluido en una lista de valores especificada.
- En la siguiente sentencia usamos "in" para averiguar si el valor del campo autor está incluido en la lista de valores especificada (en este caso, 2 cadenas).
- Hasta ahora, para recuperar los libros cuyo autor sea 'Paenza' o 'Borges' usábamos 2 condiciones:

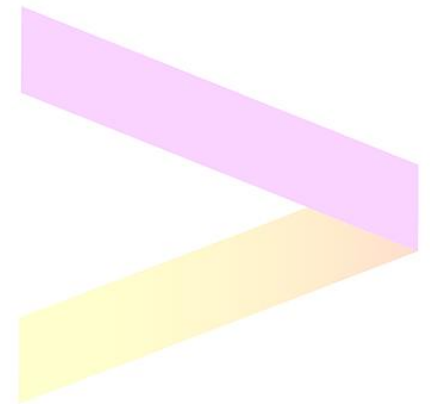
```
select *from libros
```

```
where autor='Borges' or autor='Paenza';
```

- Podemos usar "in" y simplificar la consulta:

```
select *from libros
```

```
where autor in('Borges','Paenza');
```



- Para recuperar los libros cuyo autor no sea 'Paenza' ni 'Borges' usábamos:

```
select *from libros  
where autor<>'Borges' and  
autor<>'Paenza';
```

- También podemos usar "in" anteponiendo "not":

```
select *from libros  
where autor not in ('Borges','Paenza');
```

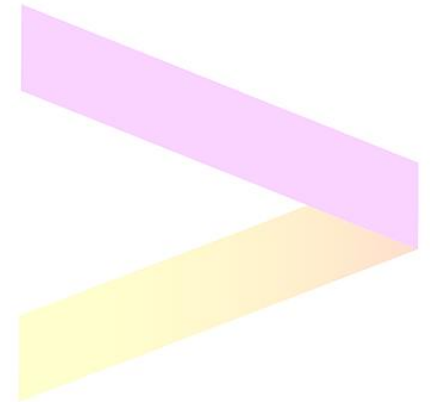


## 22 - LIKE

- Existe un operador relacional que se usa para realizar comparaciones exclusivamente de cadenas, "like" y "not like".
- El operador igual ("=") nos permite comparar cadenas de caracteres, pero al realizar la comparación, busca coincidencias de cadenas completas, realiza una búsqueda exacta.
- Imaginemos que tenemos registrados estos 2 libros:

"El Aleph", "Borges";

"Antologia poetica", "J.L. Borges";



- Entonces, podemos comparar trozos de cadenas de caracteres para realizar consultas. Para recuperar todos los registros cuyo autor contenga la cadena "Borges" debemos tipear:

```
select *from libros
```

```
where autor like "%Borges%";
```

- El símbolo "%" (porcentaje) reemplaza cualquier cantidad de caracteres (incluyendo ningún carácter). Es un carácter comodín. "like" y "not like" son operadores de comparación que señalan igualdad o diferencia.
- Para seleccionar todos los libros que comiencen con "M":

```
select *from libros
```

```
where titulo like 'M%';
```

A decorative graphic consisting of two overlapping chevron shapes pointing to the right. The top chevron is light purple and the bottom one is light yellow.

## 23 - COUNT

- Imaginemos que nuestra tabla "libros" contiene muchos registros. Para averiguar la cantidad sin necesidad de contarlos manualmente usamos la función "count()":

```
select count(*)  
from libros;
```

- La función "count()" cuenta la cantidad de registros de una tabla, incluyendo los que tienen valor nulo.
- Para contar los registros que tienen precio (sin tener en cuenta los que tienen valor nulo), usamos la función "count()" y en los paréntesis colocamos el nombre del campo que necesitamos contar:

```
select count(precio)  
from libros;
```



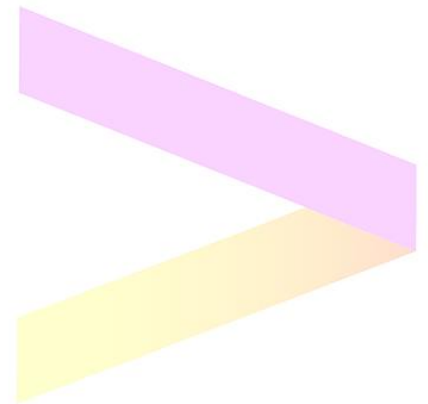
## 24 – Group By

- Podemos generar valores de resumen para un solo campo, combinando las funciones de agregado con la cláusula "group by", que agrupa registros para consultas detalladas.
- Queremos saber la cantidad de libros de cada editorial, podemos tipear la siguiente sentencia:

```
select count(*) from libros  
where editorial='Planeta';
```

- y repetirla con cada valor de "editorial":

```
select count(*) from libros  
where editorial='Emece';  
select count(*) from libros  
where editorial='Paidos';
```



- Pero hay otra manera, utilizando la cláusula "group by":

```
select editorial, count(*)
```

```
from libros
```

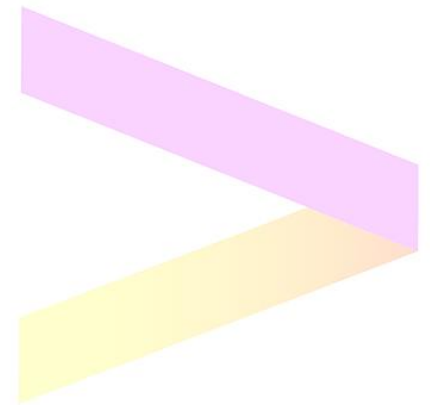
```
group by editorial;
```

- La instrucción anterior solicita que muestre el nombre de la editorial y cuente la cantidad agrupando los registros por el campo "editorial". Como resultado aparecen los nombres de las editoriales y la cantidad de registros para cada valor del campo.
- Los valores nulos se procesan como otro grupo.



- Entonces, para saber la cantidad de libros que tenemos de cada editorial, utilizamos la función "count()", agregamos "group by" (que agrupa registros) y el campo por el que deseamos que se realice el agrupamiento, también colocamos el nombre del campo a recuperar; la sintaxis básica es la siguiente:

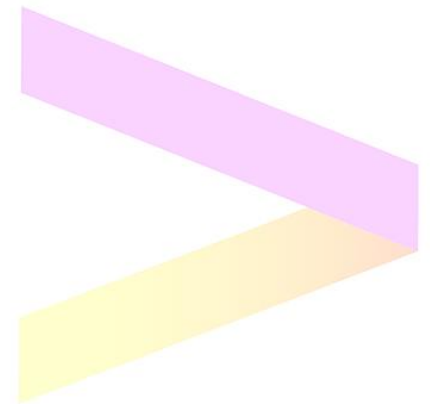
```
select CAMPO, FUNCIONDEAGREGADO  
from NOMBRETABLA  
group by CAMPO;
```



## 25 - HAVING

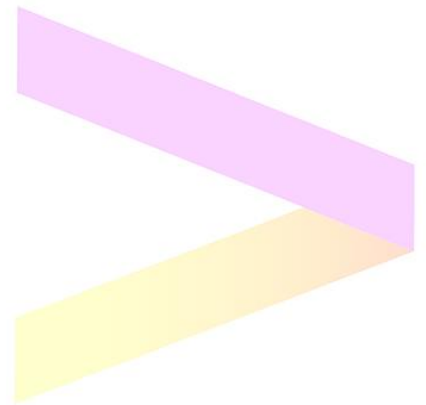
- Así como la cláusula "where" permite seleccionar (o rechazar) registros individuales; la cláusula "having" permite seleccionar (o rechazar) un grupo de registros.
- Si queremos saber la cantidad de libros agrupados por editorial usamos la siguiente instrucción ya aprendida:

```
select editorial, count(*)  
from libros  
group by editorial;
```



- Si queremos saber la cantidad de libros agrupados por editorial pero considerando sólo algunos grupos, por ejemplo, los que devuelvan un valor mayor a 2, usamos la siguiente instrucción:

```
select editorial, count(*) from libros  
group by editorial  
having count(*)>2;
```



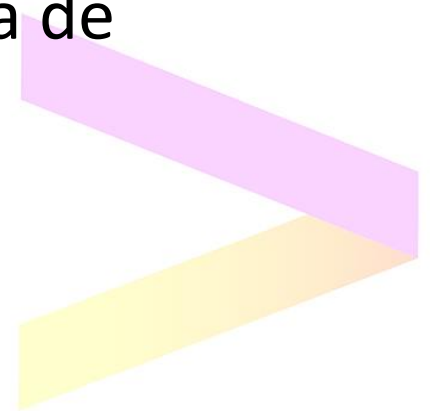
## 26 - DISTINCT

- Con la cláusula "distinct" se especifica que los registros con ciertos datos duplicados sean obviadas en el resultado. Por ejemplo, queremos conocer todos los autores de los cuales tenemos libros, si utilizamos esta sentencia:


```
select autor from libros;
```

- Aparecen repetidos. Para obtener la lista de autores sin repetición usamos:

```
select distinct autor from libros;
```



## 27 - JOIN

- Un join es una operación que relaciona dos o más tablas para obtener un resultado que incluya datos (campos y registros) de ambas; las tablas participantes se combinan según los campos comunes a ambas tablas.
  - Hay tres tipos de combinaciones:
    - 1) combinaciones internas (inner join o join),
    - 2) combinaciones externas y
    - 3) combinaciones cruzadas.
  - También es posible emplear varias combinaciones en una consulta "select", incluso puede combinarse una tabla consigo misma.
- 
- A decorative graphic consisting of two overlapping triangles pointing to the right. The top triangle is light purple and the bottom triangle is light yellow.

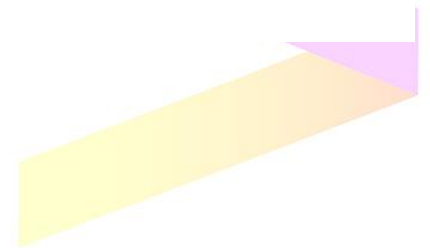
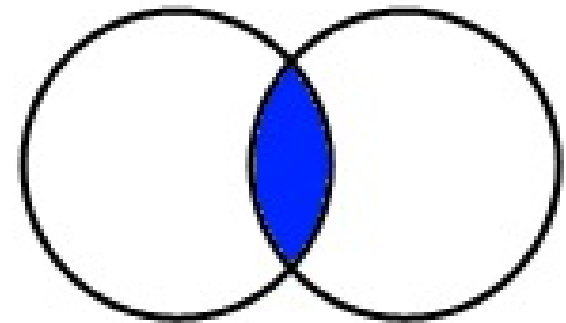
- La combinación interna emplea "join", que es la forma abreviada de "inner join". Se emplea para obtener información de dos tablas y combinar dicha información en una salida.
- La sintaxis básica es la siguiente:

```
select CAMPOS  
from TABLA1  
join TABLA2  
on CONDICIONdeCOMBINACION;
```

- Ejemplo:

```
select *from libros  
join editoriales  
on codigoeditorial=editoriales.codigo;
```

**INNER JOIN**

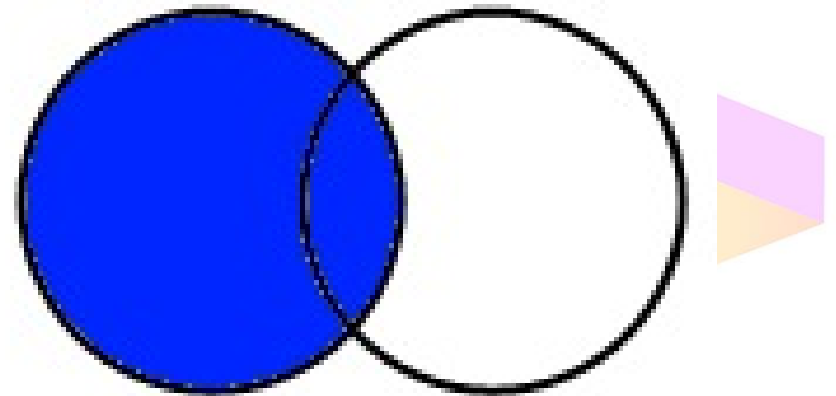


## Left Join

- Un "left join" se usa para hacer coincidir registros en una tabla (izquierda) con otra tabla (derecha); si un valor de la tabla de la izquierda no encuentra coincidencia en la tabla de la derecha, se genera una fila extra (una por cada valor no encontrado) con todos los campos correspondientes a la tabla derecha seteados a "null". La sintaxis básica es la siguiente:

```
select CAMPOS  
from TABLAIZQUIERDA  
left join TABLADERECHA  
on CONDICION;
```

### LEFT JOIN

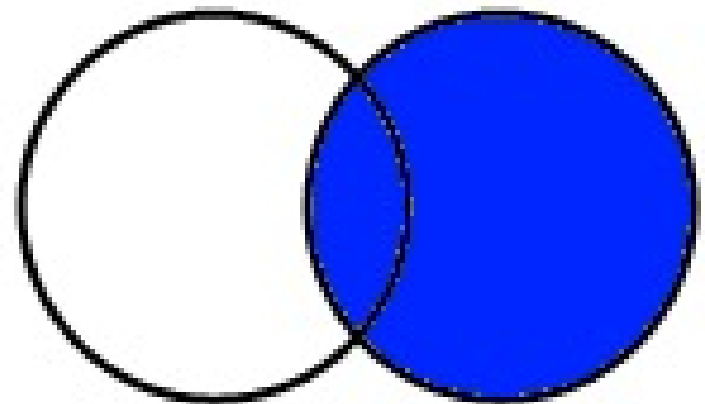


## Right Join

- Una combinación externa derecha ("right outer join" o "right join") opera del mismo modo sólo que la tabla derecha es la que localiza los registros en la tabla izquierda.

```
select CAMPOS  
from TABLAIZQUIERDA  
right join TABLADERECHA  
on CONDICION;
```

### RIGHT JOIN

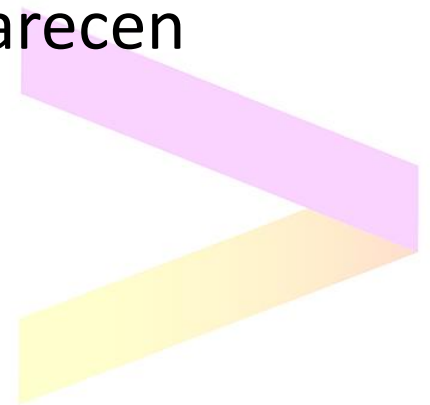




# accenture

## Full Join

- Una combinación externa completa ("full join") retorna todos los registros de ambas tablas. Si un registro de una tabla izquierda no encuentra coincidencia en la tabla derecha, las columnas correspondientes a campos de la tabla derecha aparecen seteadas a "null", y si la tabla de la derecha no encuentra correspondencia en la tabla izquierda, los campos de esta última aparecen conteniendo "null".



- Veamos un ejemplo:

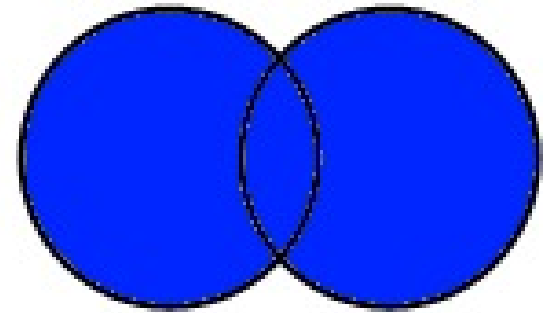
```
select titulo, nombre  
from editoriales e
```

```
full join libros l
```

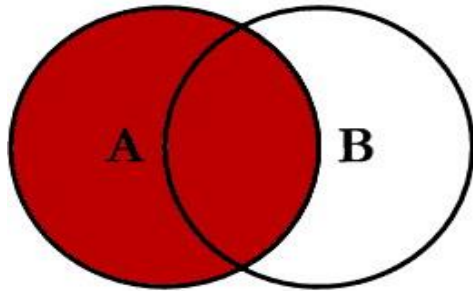
```
on codigoeditorial = e.codigo;
```

- La salida del "full join" precedente muestra todos los registros de ambas tablas, incluyendo los libros cuyo código de editorial no existe en la tabla "editoriales" y las editoriales de las cuales no hay correspondencia en "libros".

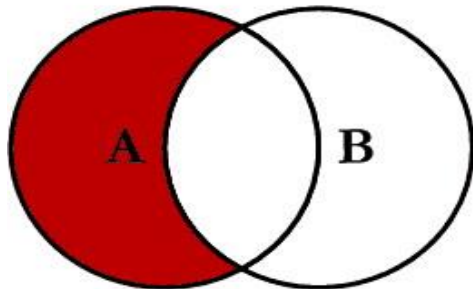
## FULL OUTER JOIN



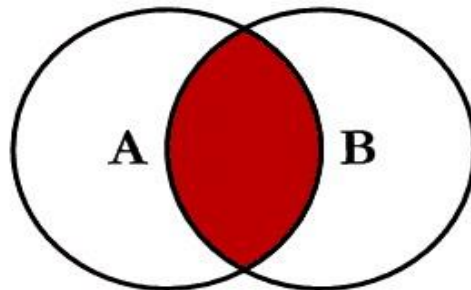
# SQL JOINS



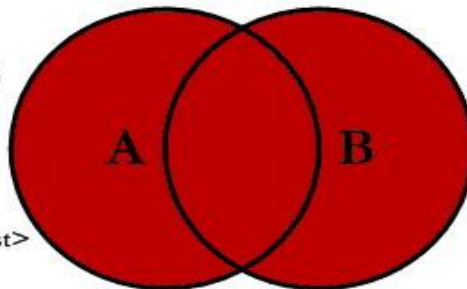
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



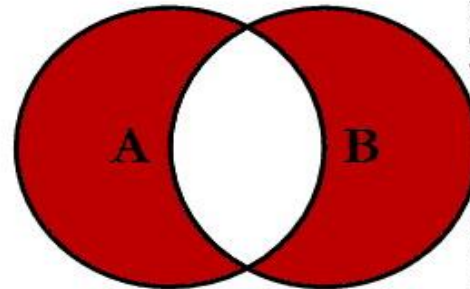
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



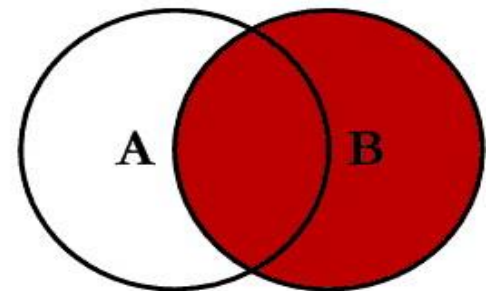
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



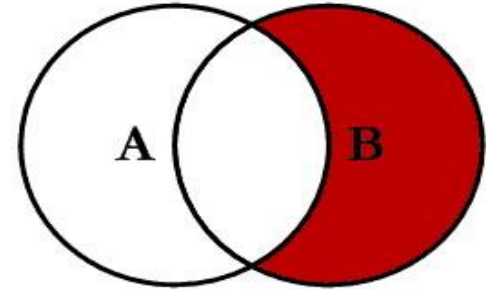
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

## 28 - UNION

- Union devuelve el resultado combinado de dos consultas.
- La sintaxis para unir dos consultas con el operador "union" es la siguiente:

*CONSULTA1*

*union*

*CONSULTA2;*

- Las consultas DEBEN tener el mismo numero de valores retornados y los valores deben ser del mismo tipo.

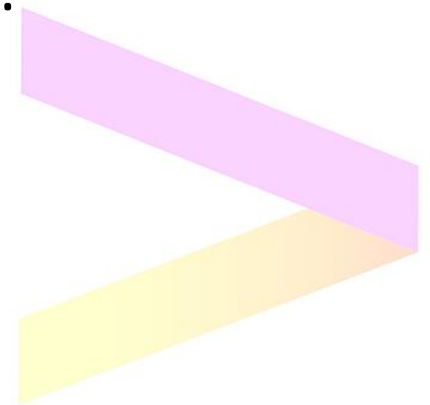


- Veamos un ejemplo. Una academia de enseñanza de idiomas da clases de inglés y frances; almacena los datos de los alumnos que estudian inglés en una tabla llamada "ingles" y los que están inscriptos en "francés" en una tabla denominada "frances".
- La academia necesita el nombre y domicilio de todos los alumnos, de todos los cursos para enviarles una tarjeta de felicitación para el día del alumno.
- Para obtener los datos necesarios de ambas tablas en una sola consulta necesitamos realizar una unión:

*select nombre, domicilio from ingles*

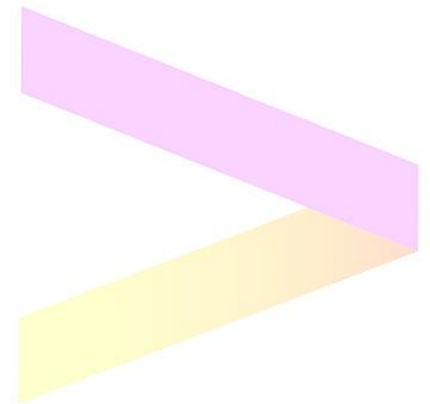
*union*

*select nombre, domicilio from frances;*



- Los encabezados del resultado de una unión son los que se especifican en el primer "select". El operador de conjunto "union" no retorna duplicados; es decir, si un registro de la primer consulta es igual a otro registro de la segunda consulta, tal registro aparece una sola vez. Si queremos que se incluyan TODOS los registros, aún duplicados, debemos emplear "union all":

```
select nombre,domicilio from ingles  
union all  
select nombre,domicilio from frances;
```



## 29 - Intersección

- "Intersect" devuelve la intersección de las consultas involucradas; es decir, el resultado retornará los registros que se encuentran en la primera y segunda consulta (y demás si las hubiere), o sea, los registros que todas las consultas tienen en común.

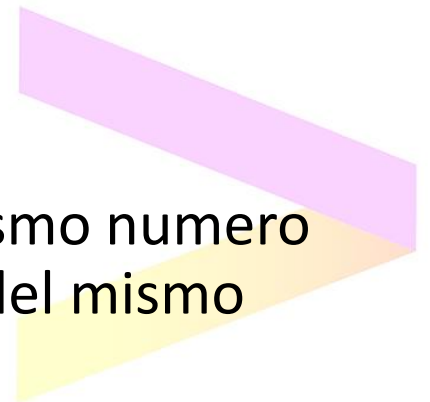
- Sintaxis:

*SENTENCIASELECT1*

*intersect*

*SENTENCIASELECT2;*

- No olvide que las consultas DEBEN tener el mismo numero de valores retornados y los valores deben ser del mismo tipo.



# accenture

- Una academia de enseñanza de idiomas da clases de inglés, frances y portugues; almacena los datos de los alumnos que estudian inglés en una tabla llamada "ingles", los que están inscriptos en "francés" en una tabla denominada "frances" y los que aprenden portugues en la tabla "portugues".
- La academia necesita el nombre y domicilio de todos los alumnos que cursan los tres idiomas para enviarles una tarjeta de descuento.
- Para obtener los datos necesarios de las tres tablas en una sola consulta necesitamos realizar una intresección:

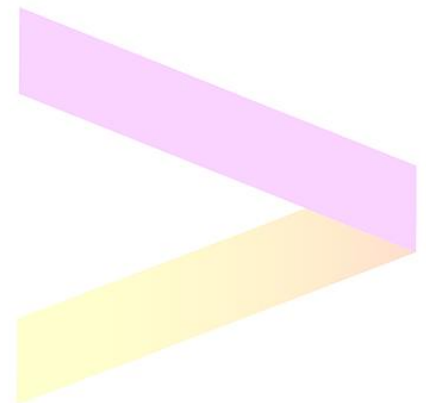
*select nombre, domicilio from ingles*

*intersect*

*select nombre, domicilio from frances*

*intersect*

*select nombre, domicilio from portugues;*





## 30 - MINUS

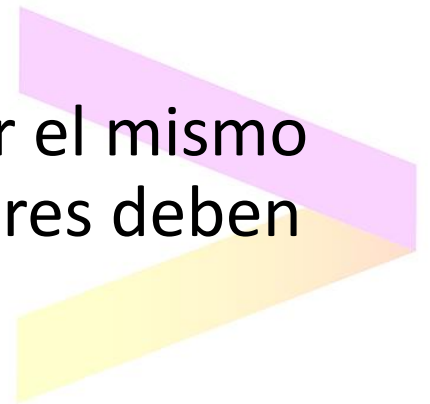
- "minus" (diferencia) devuelve los registros de la primera consulta que no se encuentran en segunda consulta, es decir, aquellos registros que no coinciden. Es el equivalente a "except" en SQL.
- Sintaxis:

*SENTENCIASELECT1*

*minus*

*SENTENCIASELECT2;*

- No olvide que las consultas DEBEN tener el mismo numero de valores retornados y los valores deben ser del mismo tipo.



# > accenture

- Una academia de enseñanza de idiomas da clases de inglés y frances; almacena los datos de los alumnos que estudian inglés en una tabla llamada "ingles" y los que están inscriptos en "francés" en una tabla denominada "frances".
- La academia necesita el nombre y domicilio de todos los alumnos que cursan solamente inglés (no presentes en la tabla "frances") para enviarles publicidad referente al curso de francés. Empleamos el operador "minus" para obtener dicha información:


*select nombre, domicilio from ingles*

*minus*

*select nombre,domicilio from frances;*



## 31 - SUBCONSULTAS

- Una subconsulta (subquery) es una sentencia "select" anidada en otra sentencia "select", "insert", "update" o "delete" (o en otra subconsulta).
  - Las subconsultas se emplean cuando una consulta es muy compleja, entonces se la divide en varios pasos lógicos y se obtiene el resultado con una única instrucción y cuando la consulta depende de los resultados de otra consulta.
  - Generalmente, una subconsulta se puede reemplazar por combinaciones y estas últimas son más eficientes.
- 
- A large, stylized arrow pointing to the right, composed of two overlapping shapes: a light purple one on top and a light yellow one on the bottom.

- Las subconsultas generalmente se incluyen entre paréntesis.
- Puede haber subconsultas dentro de subconsultas.
- Una subconsulta puede reemplazar una expresión. Dicha subconsulta debe devolver un valor escalar (o una lista de valores de un campo).



- Las subconsultas que retornan un solo valor escalar se utiliza con un operador de comparación o en lugar de una expresión:

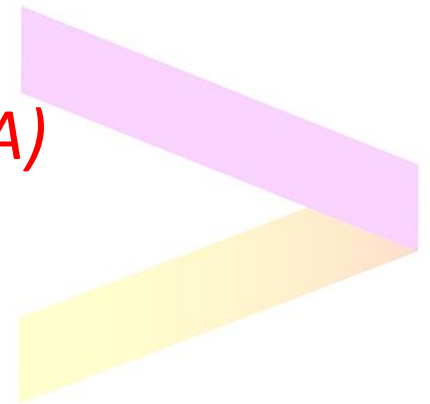
*select CAMPOS*

*from TABLA*

*where CAMPO OPERADOR (SUBCONSULTA);*

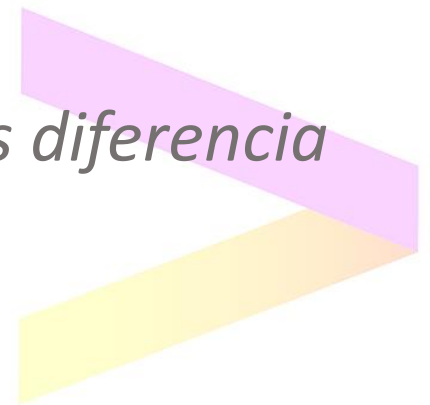
*select CAMPO OPERADOR (SUBCONSULTA)*

*from TABLA;*



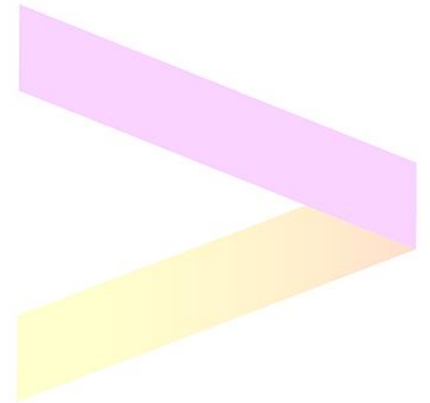
- Si queremos saber el precio de un determinado libro y la diferencia con el precio del libro más costoso, anteriormente debíamos averiguar en una consulta el precio del libro más costoso y luego, en otra consulta, calcular la diferencia con el valor del libro que solicitamos. Podemos conseguirlo en una sola sentencia combinando dos consultas:

```
select titulo,precio,  
precio-(select max(precio) from libros) as diferencia  
from libros  
where titulo='Uno';
```



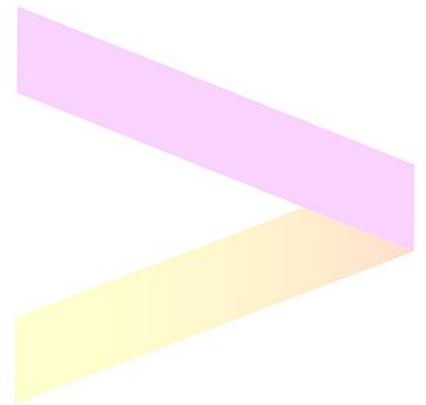
- Queremos saber el título, autor y precio del libro más costoso:

```
select titulo, autor, precio  
from libros  
where precio =  
(select max(precio) from libros);
```



- El resultado de una subconsulta con "in" (o "not in") es una lista. Luego que la subconsulta retorna resultados, la consulta exterior los usa.
- Podemos averiguar si un valor de la consulta externa pertenece o no al conjunto devuelto por una subconsulta empleando "in" y "not in".
- La sintaxis básica es la siguiente:

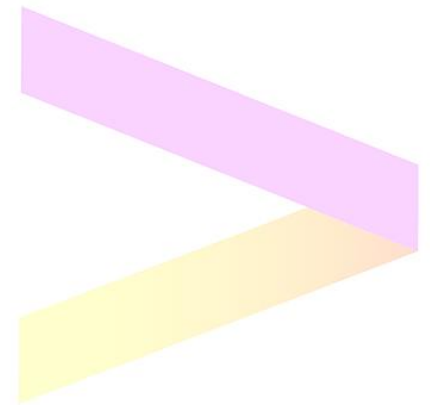
*...where EXPRESION in (SUBCONSULTA);*





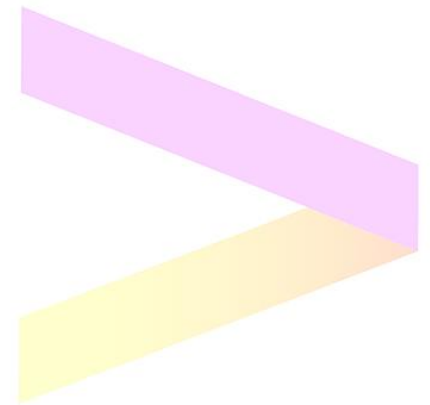
- Este ejemplo muestra los nombres de las editoriales que ha publicado libros de un determinado autor:

```
select nombre  
from editoriales  
where codigo in  
(select codigoeditorial  
from libros  
where autor='Richard Bach');
```



- Podemos reemplazar por un "join" la consulta anterior:

```
select distinct nombre  
from editoriales e  
join libros  
on codigoeditorial=e.codigo  
where autor='Richard Bach';
```



# accenture

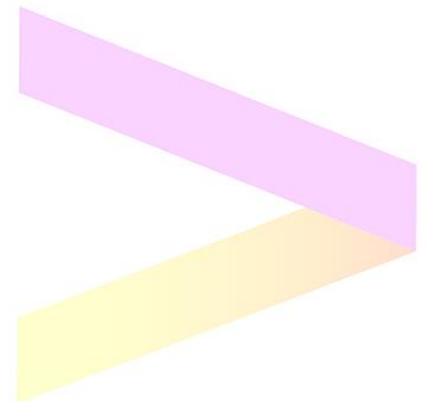
- "any" chequean si alguna fila de la lista resultado de una subconsulta se encuentra el valor especificado en la condición.
- Compara un valor escalar con los valores de un campo y devuelven "true" si la comparación con cada valor de la lista de la subconsulta es verdadera, sino "false".
- El tipo de datos que se comparan deben ser compatibles.
- La sintaxis básica es:

...VALORESCALAR OPERADORDECOMPARACION

**any** (SUBCONSULTA);

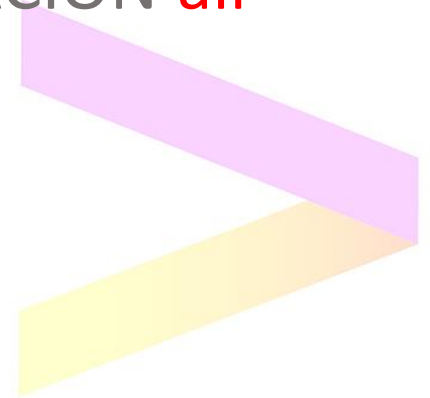
- Queremos saber los títulos de los libros de "Borges" que pertenecen a editoriales que han publicado también libros de "Richard Bach", es decir, si los libros de "Borges" coinciden con ALGUNA de las editoriales que publicó libros de "Richard Bach":

```
select titulo  
from libros  
where autor='Borges' and  
codigoeditorial = any  
(select e.codigo  
from editoriales e  
join libros l  
on codigoeditorial=e.codigo  
where l.autor='Richard Bach');
```



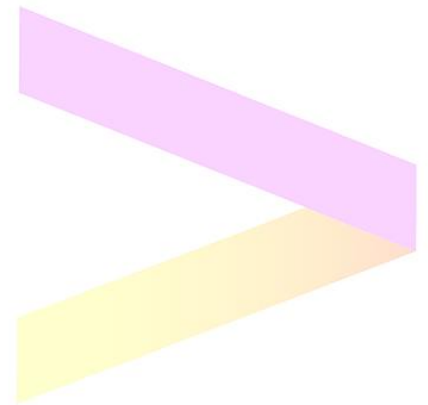
- "all" también compara un valor escalar con una serie de valores. Chequea si TODOS los valores de la lista de la consulta externa se encuentran en la lista de valores devuelta por la consulta interna.
- Sintaxis:

VALORESCALAR OPERADORDECOMPARACION **all**  
(SUBCONSULTA);



- Queremos saber si TODAS las editoriales que publicaron libros de "Borges" coinciden con TODAS las editoriales que publicaron libros de "Richard Bach":

```
select titulo  
from libros  
where autor='Borges' and  
codigoeditorial = all  
(select e.codigo  
from editoriales e  
join libros l  
on codigoeditorial=e.codigo  
where l.autor='Richard  
Bach');
```



- Los operadores "exists" y "not exists" se emplean para determinar si hay o no datos en una lista de valores.
- Estos operadores pueden emplearse con subconsultas correlacionadas para restringir el resultado de una consulta exterior a los registros que cumplen la subconsulta (consulta interior). Estos operadores retornan "true" (si las subconsultas retornan registros) o "false" (si las subconsultas no retornan registros).
- Cuando se coloca en una subconsulta el operador "exists", Oracle analiza si hay datos que coinciden con la subconsulta, no se devuelve ningún registro, es como un test de existencia; Oracle termina la recuperación de registros cuando por lo menos un registro cumple la condición "where" de la subconsulta.



# accenture

- La sintaxis básica es la siguiente:

*... where exists (SUBCONSULTA);*

- En este ejemplo se usa una subconsulta correlacionada con un operador "exists" en la cláusula "where" para devolver una lista de clientes que compraron el artículo "lapiz":

*select cliente,numero*

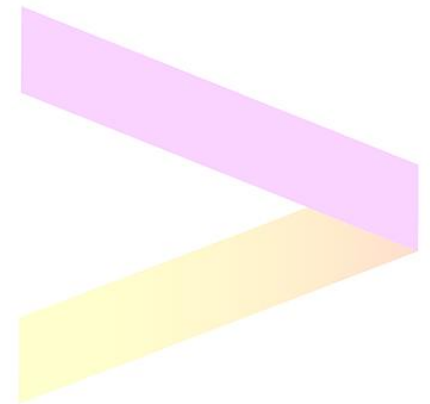
*from facturas f*

*where exists*

*(select \*from Detalles d*

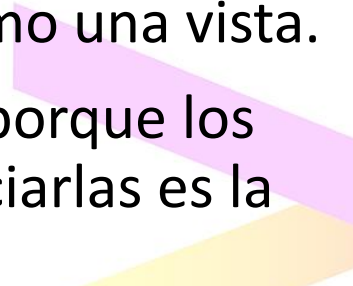
*where f.numero=d.numeroofactura*

*and d.articulo='lapiz');*



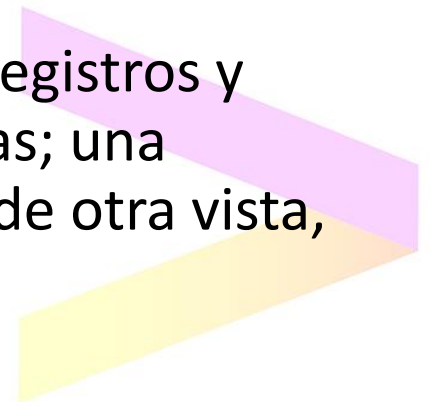


## 32 - VISTAS

- Una vista es un objeto. Una vista es una alternativa para mostrar datos de varias tablas; es como una tabla virtual que almacena una consulta. Los datos accesibles a través de la vista no están almacenados en la base de datos, en la base de datos se guarda la definición de la vista y no el resultado de ella.
  - Entonces, una vista almacena una consulta como un objeto para utilizarse posteriormente. Las tablas consultadas en una vista se llaman tablas base. En general, se puede dar un nombre a cualquier consulta y almacenarla como una vista.
  - Una vista suele llamarse también tabla virtual porque los resultados que retorna y la manera de referenciarlas es la misma que para una tabla.
- 
- A decorative graphic in the bottom right corner consisting of two overlapping chevron shapes pointing to the right. The top chevron is light purple and the bottom one is light yellow.



- Las vistas permiten:
- - simplificar la administración de los permisos de usuario: se pueden dar al usuario permisos para que solamente pueda acceder a los datos a través de vistas, en lugar de concederle permisos para acceder a ciertos campos, así se protegen las tablas base de cambios en su estructura.
- - mejorar el rendimiento: se puede evitar tipear instrucciones repetidamente almacenando en una vista el resultado de una consulta compleja que incluya información de varias tablas.
- Podemos crear vistas con: un subconjunto de registros y campos de una tabla; una unión de varias tablas; una combinación de varias tablas; un subconjunto de otra vista, combinación de vistas y tablas.



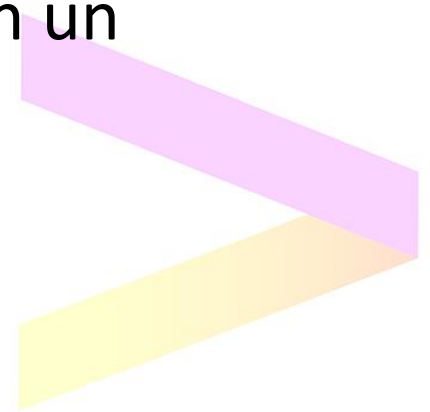
# accenture

- Una vista se define usando un "select".
- La sintaxis básica para crear una vista es la siguiente:

```
create view NOMBREVISTA as  
SUBCONSULTA;
```

- El contenido de una vista se muestra con un "select":

```
select *from NOMBREVISTA;
```

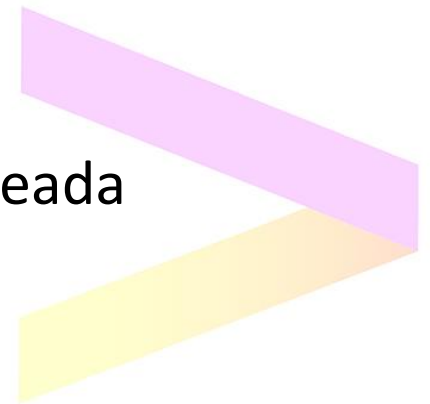


- En el siguiente ejemplo creamos la vista "vista\_empleados", que es resultado de una combinación en la cual se muestran 4 campos:

```
create view vista_empleados as  
select (apellido || ' ' || e.nombre) as nombre, sexo,  
s.nombre as seccion, cantidadhijos  
from empleados e  
join secciones s  
on codigo=seccion;
```

- Para ver la información contenida en la vista creada anteriormente tipeamos:

```
select * from vista_empleados;
```



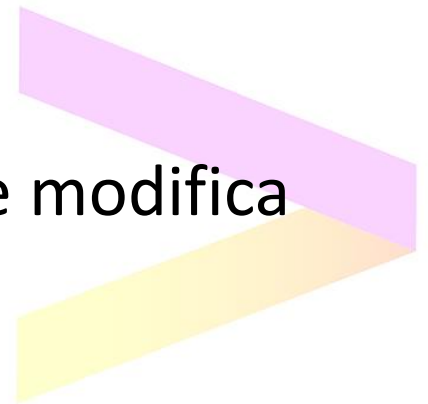
- Podemos realizar consultas a una vista como si se tratara de una tabla:

```
select seccion, count(*) as cantidad  
from vista_empleados;
```

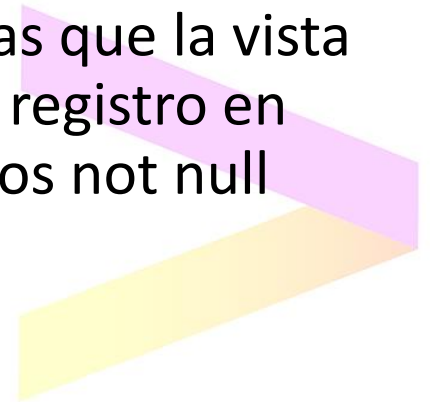
- Para quitar una vista se emplea "drop view":

```
drop view NOMBREVISTA;
```

- Si se modifican los datos de una vista, se modifica la tabla base.



- Se puede insertar, actualizar o eliminar datos de una tabla a través de una vista, teniendo en cuenta lo siguiente, las modificaciones que se realizan a las vistas:
  - no pueden afectar a más de una tabla consultada. Pueden modificarse y eliminarse datos de una vista que combina varias tablas pero la modificación o eliminación solamente debe afectar a una sola tabla.
  - no se pueden cambiar los campos resultado de un cálculo.
  - pueden generar errores si afectan a campos a las que la vista no hace referencia. Por ejemplo, si se ingresa un registro en una vista que consulta una tabla que tiene campos not null que no están incluidos en la vista.



- Con la cláusula "with read only" (sólo lectura) evitamos que se puedan realizar inserciones, actualizaciones y eliminaciones mediante una vista.
- Sintaxis:

```
create view NOMBREVISTA  
as SUBCONSULTA  
with read only;
```

