Universidad Nacional Autónoma de Honduras

Ingeniería en Sistemas

Bases de Datos I

**Resumen Unidad II**

Ing. Erick Vladimir Reyes Marín

Contenido

[Sentencias SQL 3](#_Toc423380942)

[Sentencias DDL (Data Definition Language) 3](#_Toc423380943)

[Sentencias DML 4](#_Toc423380944)

[Consulta de informacion (QL) 5](#_Toc423380945)

[Gestión de transacciones 6](#_Toc423380946)

[Gestión de fechas 6](#_Toc423380947)

[Campos auto numéricos, Secuencias 8](#_Toc423380948)

[Algebra Relacional 9](#_Toc423380949)

[Selección (Select) 9](#_Toc423380950)

[Proyección (Project) 9](#_Toc423380951)

[Producto cartesiano (Cross-Product) 10](#_Toc423380952)

[NaturalJoin 11](#_Toc423380953)

[Cruces (Joins) Convencionales (INNER JOIN, NATURAL JOIN) 12](#_Toc423380954)

[Usuario de pruebas HR (Human Resources) 13](#_Toc423380955)

[LEFT JOIN e RIGHT JOIN 14](#_Toc423380956)

[Recomendaciones al realizar consultas cruzadas: 14](#_Toc423380957)

Recomendaciones al momento de diseñar bases de datos:

* Evitar utilizar acentos, símbolos, ñ
* Nombres descriptivos
* Cambiar nombres de los campos de las llaves foráneas
* Tipos de datos para una llave primaria y llaves foráneas.
* Orden de los campos.
* Tamaño de los Varchar
* Faltan campos
* Tablas intermedias

# Sentencias SQL

## Sentencias DDL (Data Definition Language)

Las sentencias DDL son las que se utilizan para definir, modificar o eliminar estructuras o elementos físicos dentro de la base de datos.

A continuación se definen algunas sentencias DML básicas:

Crear una tabla

CREATE TABLE NOMBRE\_TABLA

(

CAMPO1 TIPO\_DATO [MODIFICADORES],

CAMPO2 TIPO\_DATO [MODIFICADORES],

CAMPON TIPO\_DATO [MODIFICADORES],

CONSTRAINT NOMBRE\_PK PRIMARY KEY

(

CAMPOS\_LLAVE\_PRIMARIA

)

ENABLE

);

Sentencia DDL para agregar una nueva columna:

ALTER TABLE NOMBRE\_TABLA

ADD (CAMPO1 TIPO\_DATO [MODIFICADORES], ... CAMPON TIPO\_DATO [MODIFICADORES]);

Eliminar una columna:

ALTER TABLE NOMBRE\_TABLA

DROP COLUMN CAMPO;

Modificar una columna:

ALTER TABLE NOMBRE\_TABLA

MODIFY (CAMPO TIPO\_DATO\_NUEVO);

Eliminar una tabla:

DROP TABLE NOMBRE\_TABLA.

## Sentencias DML

INSERT INTO TBL\_CLIENTES (

CODIGO\_CLIENTE,

NOMBRE\_CLIENTE,

APELLIDO\_CLIENTE,

GENERO

) VALUES (2, 'Maria', 'Ruiz', 'F');

INSERT INTO NOMBRE\_TABLA(CAMPOS) VALUES(VALORES)

Actualizar un registro:

UPDATE NOMBRE\_TABLA

SET CAMPO1=NUEVO\_VALOR1,

CAMPO2=NUEVO\_VALOR2,

....

CAMPON=NUEVO\_VALOR\_N

WHERE CONDICION;

UPDATE TBL\_CLIENTES

SET APELLIDO\_CLIENTE = 'Juarez'

WHERE CODIGO\_CLIENTE = 2

Eliminar un registro:

DELETE FROM NOMBRE\_TABLA

WHERE CONDICIONES;

Gestion de transacciones (Activa)

Unicamente aplica a instrucciones DML

\*Hacer efectiva una transaccion

commit;

\*Deshacer los cambios hasta el último commit

rollback;

## Consulta de informacion (QL)

Instrucciones cuya principal funcionalidad es consultar informacion para ser mostrada o ser gestionada por el usuario desde un aplicativo.

Consulta basica

SELECT \*

FROM NOMBRE\_TABLA;

Es recomendable escribir siempre los nombres de los campos que se necesitan.

Consulta por campos

SELECT CAMPOS

FROM NOMBRE\_TABLA;

Consulta por campos

SELECT CAMPOS

FROM NOMBRE\_TABLA

WHERE CONDICIONES;

Operadores logicos

\*AND

\*OR

\*NOT

\*IN (Retorna lor registros que cumplan con una lista especifica de opciones)

Operadores de comparacion

=

>=

<=

!= o <>

BETWEEN

Consultas con el mismo resultado:

SELECT \*

FROM tbl\_clientes

WHERE nombre\_cliente != 'Maria';

SELECT \*

FROM tbl\_clientes

WHERE nombre\_cliente <> 'Maria';

SELECT \*

FROM tbl\_clientes

WHERE NOT nombre\_cliente = 'Maria';

# Gestión de transacciones

//Pendiente

# Gestión de fechas

|  |  |
| --- | --- |
| **Siglos y años** | |
| CC | Siglo |
| SCC | Siglo. Si es AC (Antes de Cristo), lleva un signo - |
| YYYY | Año, formato de 4 dígitos |
| SYYY | Año, formato de 4 dígitos. Si es AC lleva un signo - |
| YY | Año, formato de 2 dígitos |
| YEAR | Año, escrito en letras y en inglés (por ejemplo, 'TWO THOUSAND TWO') |
| SYEAR | Ídem, pero si es AC lleva el signo - |
| BC | Antes o Después de Cristo (AC o DC) para usar con los anteriores, por ejemplo YYYY BC |
| **Meses** | |
| Q | Trimestre: Ene-Mar=1, Abr-Jun=2, Jul-Sep=3, Oct-Dic=4 |
|  |  |
| MM | Número de mes (1-12) |
| RM | Número de mes en números romanos (I-XII) |
| MONTH | Nombre del mes completo rellenado con espacios hasta 10 espacios (SEPTIEMBRE) |
| FMMONTH | Nombre del mes completo, sin espacios adicionales |
| MON | Tres primeras letras del mes: ENE, FEB,... |
| **Semanas** | |
| WW | Semana del año (1-52) |
| W | Semana del mes (1-5) |
| **Días** | |
| DDD | Día del año (1-366) |
| DD | Día del mes (1-31) |
| D | Día de la semana (1-7) |
| DAY | Nombre del día de la semana rellenado a 9 espacios (MIÉRCOLES) |
| FMDAY | Nombre del día de la semana, sin espacios |
| DY | Tres primeras letras del nombre del día de la semana |
| DDTH | Día (ordinal): 7TH |
| DDSPTH | Día ordinal en palabra, en inglés: SEVENTH |
| **Horas** | |
| HH | Hora del día (1-12) |
| HH12 | Hora del día (1-12) |
| HH24 | Hora del día (1-24) |
| SPHH | Hora del día, en palabra, inglés: SEVEN |
| AM | am o pm, para usar con HH, como 'HH:MI am' |
| PM | am o pm |
| A.M. | a.m. o p.m. |
| P.M. | a.m. o p.m. |
| **Minutos y segundos** | |
| MI | Minutos (0-59) |
| SS | Segundos (0-59) |
| SSSS | Segundos después de medianoche (0-86399) |

# Campos auto numéricos, Secuencias

Mediante las secuencias, Oracle puede proporcionar una lista consecutiva de números unívocos que sirve para simplificar las tareas de programación. La primera vez que una consulta llama a una secuencia, se devuelve un valor predeterminado.

Para crear una secuencia en Oracle mediante SQL utilizaremos el comando create sequence con la siguiente sintaxis:

CREATE SEQUENCE nombre\_secuencia

INCREMENT BY numero\_incremento

START WITH numero\_por\_el\_que\_empezara

MAXVALUE valor\_maximo | NOMAXVALUE

MINVALUE valor\_minimo | NOMINVALUE

CYCLE | NOCYCLE

ORDER | NOORDER

Por ejemplo, si queremos crear una secuencia que empiece en 100 y se incremente de uno en uno utilizaremos la siguiente consulta SQL:

CREATE SEQUENCE secuencia\_id\_cliente

INCREMENT BY 1

START WITH 100

Para utilizar la secuencia, en primer lugar, crearemos una tabla de prueba (para insertar un registro y comprobar que la secuencia anterior funciona correctamente):

CREATE TABLE clientes (

codigo number not null primary key,

nombre varchar2(100) unique not null

)

Para utilizar la secuencia creada en una inserción de fila:

INSERT INTO clientes(codigo,nombre)

VALUES (secuencia\_id\_cliente.NEXTVAL, 'Juan');

Realizamos otra inserción para comprobar que el incremento es de 1:

INSERT INTO clientes(codigo,nombre)

VALUES (secuencia\_id\_cliente.NEXTVAL, Pedro);

# Algebra Relacional

Se define como un conjunto de operaciones que se ejecutan sobre las relaciones (tablas) para obtener un resultado, el cual es otra relación.

## Selección (Select)

Este operador se aplica a una relación R produciendo una nueva relación con un subconjunto de tuplas de R. Las tuplas de la relación resultante son las que satisfacen una condición C sobre algún atributo de R. Es decir selecciona **filas (tuplas)** de una tabla según un cierto criterio C. El criterio C es una expresión condicional, similar a las declaraciones del tipo “if”, es “booleana” esto quiere decir que para cada tupla de R toma el valor Verdad(true) o Falso(false).

(relación = tabla)

(tupla = registro)

Para representar Select en álgebra relacional se utiliza la letra griega sigma σ. Por lo tanto, si utilizamos la notación σc R queremos decir que se aplica la condición C a cada tupla de R. Si la condición es Verdad true, dicha tupla pertenecerá al resultado y si es Falsa false, dicha tupla no será seleccionada. El esquema de la relación resultante es el mismo esquema R, se muestran los atributos en el mismo orden que se usan en la tabla R.

Ejemplo:

**Algebra relacional:**

σedad>30 Ingenieros

**SQL:**

SELECT \* FROM INGENIEROS WHERE EDAD > 30;

## Proyección (Project)

El operador Proyección se utiliza para producir una nueva relación desde R. Esta nueva relación contiene sólo algunos de los atributos de R, es decir, realiza la selección de algunas de las **columnas** de una tabla R.

**Notación en Álgebra Relacional**

Project en Álgebra Relacional se representa por la letra griega **pi**: π(A1,...,An)R

Ejemplo:

**Algebra relacional:**

π(id,nombre)Ingenieros

**SQL:**

SELECT ID, NOMBRE FROM INGENIEROS;

## Producto cartesiano (Cross-Product)

En teoría de conjuntos, el producto cartesiano de dos conjuntos es una operación que resulta en otro conjunto cuyos elementos son todos los pares ordenados que pueden formarse tomando el primer elemento del par del primer conjunto, y el segundo elemento del segundo conjunto. En el Álgebra Relacional se mantiene esta idea con la diferencia que R y S son relaciones, entonces los miembros de R y S son tuplas, que generalmente consisten de más de un componente, cuyo resultado de la vinculación de una tupla de R con una tupla de S es una tupla más larga, con un componente para cada uno de los componentes de las tuplas constituyentes. Es decirCross-product define una relación que es la concatenación de cada una de las filas de la relación R con cada una de las filas de la relación S.

**Notación en Álgebra Relacional**

Para representar Cross-product en Álgebra Relacional se utiliza la siguiente terminología:

R×S

Ejemplo:

**Algebra Relacional:**

RxS

**SQL**

SELECT \* FROM R,S;

--Siendo R y S dos tablas.

## NaturalJoin

Este operador se utiliza cuando se tiene la necesidad de unir relaciones vinculando sólo las tuplas que coinciden de alguna manera. NaturalJoin une sólo los pares de tuplas de R y S que sean comunes. Más precisamente una tupla r de R y una tupla s de S se emparejan correctamente si y sólo si r y s coinciden en cada uno de los valores de los atributos comunes, el resultado de la vinculación es una tupla, llamada **joined tuple**. Entonces, al realizar NaturalJoin se obtiene una relación con los atributos de ambas relaciones y se obtiene combinando las tuplas de ambas relaciones que tengan el mismo valor en los atributos comunes.

**Notación en Álgebra Relacional**

Para denotar NaturalJoin se utiliza la siguiente simbología: R⊳⊲S.

Ejemplo:

Algebra relacional:

USUARIO ⊳⊲ TIPO\_USUARIO

SQL:

SELECT \*

FROM TBL\_USUARIOS

INNER JOIN TBL\_TIPO\_USUARIO

ON (TBL\_USUARIOS.CODIGO\_TIPO\_USUARIO = TBL\_TIPO\_USUARIO.CODIGO\_TIPO\_USUARIO)

Más información sobre algebra relacional:

<http://csrg.inf.utfsm.cl/~jfuentes/_build/html/lectures/week1/lecture3.html>

<http://www.unirioja.es/cu/arjaime/Temas/04.Anexo.pdf>

# Cruces (Joins) Convencionales (INNER JOIN, NATURAL JOIN)

A continuación se presenta tres scripts cuyo resultado es el mismo. El resultado y el rendimiento a nivel de Oracle es el indiferente. Debe utilizar un INNER JOIN cuando este seguro que la llave primaria y la llave foránea no permiten campos nulos, de no ser así puede existir perdida de información.

Utilizando INNER JOIN

SELECT A.NOMBRE\_USUARIO,

A.CORREO,

B.TIPO\_USUARIO,

B.DESCRIPCION

FROM TBL\_USUARIOS A

INNER JOIN TBL\_TIPO\_USUARIO B

ON (A.CODIGO\_TIPO\_USUARIO = B.CODIGO\_TIPO\_USUARIO);

Utilizando NATURAL JOIN: En el caso de utilizar este tipo de JOIN los campos deben llamarse igual en ambas tablas.

SELECT A.NOMBRE\_USUARIO,

A.CORREO,

B.TIPO\_USUARIO,

B.DESCRIPCION

FROM TBL\_USUARIOS A

NATURAL JOIN TBL\_TIPO\_USUARIO B;

Utilizando un producto cartesiano con un criterio en la cláusula WHERE

SELECT A.NOMBRE\_USUARIO,

A.CORREO,

B.TIPO\_USUARIO,

B.DESCRIPCION

FROM tbl\_usuarios A,

tbl\_tipo\_usuario B

WHERE A.codigo\_tipo\_usuario = B.codigo\_tipo\_usuario;

## Usuario de pruebas HR (Human Resources)

Oracle viene por defecto con un esquema de pruebas llamado HR, el cual contiene un conjunto de tablas relacionadas a un sistema de Recursos Humanos, dicho esquema por defecto esta bloqueado por lo cual se define a continuacion el script para desbloquearlo y asignar un password:

Sustituya la palabra PASSWORD por la constraseña de su preferencia:

ALTER USER HR IDENTIFIED BY "PASSWORD" ACCOUNT UNLOCK;

Ejemplo de cruces de la tabla de empleados y trabajos:

INNER JOIN:

SELECT first\_name ||' '|| last\_name as name,

email,

job\_title

FROM employees A

INNER JOIN jobs B

ON (A.job\_id = B.job\_id);

NATURAL JOIN:

SELECT first\_name ||' '|| last\_name as name,

email,

job\_title

FROM employees A

NATURAL JOIN jobs B;

Producto cartesiano aplicando la cláusula WHERE

SELECT first\_name ||' '|| last\_name as name,

email,

job\_title

FROM employees A,

jobs B

WHERE A.job\_id = B.job\_id;

# LEFT JOIN e RIGHT JOIN

Con la sentencia INNER JOIN existe perdida de información cuando uno de los campos que especifica en la sentencia ON o el campo que hace la relación (llave foránea) es nulo o tiene un valor que no existe (sin integridad referencial). Los registros que no coinciden no son tomados en cuenta al momento de utilizar un INNER JOIN, esto en muchas ocasiones puede ser catastrófico ya que puede haber pérdida de información valiosa.

Para evitar este problema se utiliza la sentencia LEFT JOIN o RIGHT JOIN de tal forma que se incluye los registros de la tabla de la derecha o de la izquierda (dependiendo del join) de tal forma que no exista perdida de registros para los que no coincidan con la llave primaria y la llave foránea.

Ejemplo:

Intente probar este script con un INNER JOIN y luego con un LEFT JOIN y podrá ver que existe perdida de información utilizando el INNER JOIN mientras que con el otro se conserva la misma cantidad de registros.

SELECT A.first\_name||' '||A.last\_name as empleado,

b.first\_name||' '||b.last\_name as jefe

FROM EMPLOYEES A --EMPLEADOS

LEFT JOIN EMPLOYEES B --JEFES

ON (A.MANAGER\_ID = B.EMPLOYEE\_ID);

Cruce de tres tablas utilizando LEFT JOIN

SELECT A.first\_name||' '||A.last\_name as empleado,

B.first\_name||' '||B.last\_name as jefe,

C.first\_name||' '||C.last\_name as super\_jefe

FROM EMPLOYEES A --EMPLEADOS

LEFT JOIN EMPLOYEES B --JEFES

ON (A.MANAGER\_ID = B.EMPLOYEE\_ID)

LEFT JOIN EMPLOYEES C --SUPER JEFES

ON (B.MANAGER\_ID = C.EMPLOYEE\_ID);

## Recomendaciones al realizar consultas cruzadas:

1. Consultar cada tabla por aparte para ver la información.

2. Al hacer el cruce, verificar si hay pérdida o duplicidad de información

3. La cantidad de registros resultantes debe coincidir con la cantidad de registros de la tabla principal.