2.5.- El lenguaje estándar SQL

- El SQL es un lenguaje estándar de definición y manipulación (y consulta) de bases de datos relacionales.
- El SQL estándar incluye:
  - Características del Álgebra Relacional.
  - Características del Cálculo Relacional de Tuplas.
- La versión que actualmente se encuentra más extendida es el SQL2 (ó SQL-92).

2.5.1.- SQL como lenguaje de definición de datos (DDL)

Instrucciones del SQL para poder definir esquemas relacionales:

- **create schema**: permite dar nombre a un esquema relacional y declarar el usuario que es el creador y propietario de dicho esquema.
- **create domain**: permite definir un nuevo dominio de datos.

ORACLE
- **create table**: define una tabla, su esquema y las restricciones asociadas.

ORACLE
- **create view**: define una vista o relación derivada en el esquema relacional.
- **create assertion**: permite definir restricciones de integridad generales.

ORACLE
- **grant**: permite definir autorización de operaciones sobre objetos de la BD.

Todas estas instrucciones tienen asociada la operación inversa (DROP / REVOKE) y modificación (ALTER).

2.5.1.1.- Definición del Esquema (SQL)

create schema [*esquema*] [authorization *usuario*]

[lista_elemento_esquema];

Un elemento de esquema puede ser uno de los siguientes:

- Definición de dominio.
- Definición de tabla.
- Definición de vista.
- Definición de restricción.
- Definición de privilegio.

Eliminación de la definición de un esquema relacional:

drop schema *esquema* {restrict | cascade};

2.5.1.2.- Definición de Dominios (SQL)

create domain *dominio* [as] tipo_dato

[default {literal | función_sistema | null }]

[definición_restricción_dominio];

Funciones del sistema:

- user
- current_user
- session_user
- current_date
- current_time
- current_timestamp.

2.5.1.2.- Definición de Dominios (SQL)

A un dominio se le puede asociar un conjunto de restricciones:

[constraint *restricción*]
check (expresión_condicional)
[not] deferrable

- *expresión_condicional* permite expresar cualquier condición que debe cumplir siempre el dominio (debe ser CIERTA o INDEFINIDA)
- **deferrable** indica que el sistema ha de comprobar la restricción al finalizar la transacción activa.
- **Not deferrable** indica que el sistema ha de comprobar la restricción después de cada operación de actualización a la base de datos.

2.5.1.2.- Definición de Dominios (SQL). Ejemplo

```
CREATE DOMAIN ángulo AS FLOAT
    DEFAULT 0
    CHECK (VALUE >= 0 AND VALUE < 360)
    NOT DEFERRABLE;
```

Eliminación de un Dominio:

drop domain *dominio* [restrict | cascade]

2.5.1.3.- Definición de Tablas (SQL).

CREATE TABLE tabla
comalista_definición_columna
[comalista_definición_restrición_tabla];

La definición de una columna de una tabla se realiza como sigue:

columna {tipo_dato | *dominio*}
[default {literal | función_sistema | null }]
[lista_definición_restricción_columna]

Las restricciones que se pueden definir sobre las columnas son las siguientes:

- not null: restricción de valor no nulo.
- Definiciones de restricciones de CP, UNI, CAj de una sola columna.
- Definición de restricciones generales con la cláusula check.

2.5.1.3.- Definición de Tablas (SQL).

La cláusula para definir restricciones de tabla es la siguiente:

```
[constraint restricción]
    { primary key (comalista_columna)
      | unique (comalista_columna)
      | foreign key (comalista_columna)
            references tabla[(comalista_columna)]
    [match {full | partial}]                          * NO ORACLE
    [on update [cascade |                              * NO ORACLE
                    set null | set default | no action ]]   * NO ORACLE
    [on delete [cascade |
                    set null | set default | no action ]]   * NO ORACLE
      | check expresión_condicional }
    [comprobación_restricción]
```

- debe ser CIERTA o INDEFINIDA.
- no puede incluir subconsultas ni referencias a otras tablas.

2.5.1.3.- Ejemplo: Proveedor-Piezas-Suministro

d_cod_pieza: tira(4)
d_cod_proy: tira(4)
d_dni: entero (positivo)

Proveedor(dni: d_dni, nombre: tira(40), dirección: tira(25), ciudad: tira(30))
CP: {dni}
VNN: {nombre}

Pieza(código: d_cod_pieza, desc: tira(40), color: tira(20), peso: real)
CP: {código}

Suministro(dni: d_dni, código: tira(4), precio: real)
CP: {dni, código}
CAj: {dni} → Proveedor
CAj: {código} → Pieza

Restricciones de integridad:

R1) Px: Pieza ∀Px: Pieza (Px.color='rojo' → Px.peso>100)

R2) Px: Pieza, Sx: Suministro ∀Px: Pieza (∃Sx: Suministro (Sx.código=Px.código))

2.5.1.3.- Ejemplo: Proveedor-Piezas-Suministro (SQL)

```
create schema Almacén
    authorization pepe
    create domain d_cod_pieza as char(4)
    create domain d_cod_proy as char(4)
    create domain d_dni as integer check value>0
    create table Proveedor    ( dni          d_dni     primary key,
                                nombre       varchar(40)     not null,
                                dirección    char(25),
                                ciudad       char(30)                )
    create table Pieza        ( código       d_cod_pieza     primary key,
                                desc         varchar(40),
                                color        char(20),
                                peso         float,
                                constraint r1 check (color<>'rojo' or peso>100))
    create table Suministro   ( dni          d_dni,
                                código       d_cod_pieza     references Pieza,
                                precio       float,
                                primary key (dni, código),
                                foreign key (dni) references Proveedor(dni) );
```

⇐ R1

¿Y R2?

2.5.1.3.- Definición de Tablas (SQL). Cláusula MATCH

- completa (**match full**): en cada tupla de *R* la clave ajena *CA* tiene el valor nulo o no lo tiene, en cada una de sus columnas. En el segundo caso, ha de existir una fila en la tabla *S* cuyo valor en las columnas de *CU* sea idéntico.
- parcial (**match partial**): en cada tupla de *R* la clave ajena *CA* tiene el valor nulo en cada una de sus columnas, o ha de existir una fila en la tabla *S*, de forma que para las columnas de la clave ajena *CA* que no tienen valor nulo, el valor en las columnas correspondientes de *CU* es idéntico.
- débil (**no se incluye cláusula match**): en cada tupla de *R* si la clave ajena *CA* no tiene el valor nulo, en cada una de sus columnas, ha de existir una fila en la tabla *S* tal que el valor en coincida en todas las columnas.

ORACLE

2.5.1.3.- Modificación de Definición de Tablas (SQL).

Para modificar la definición de una tabla:

```
alter table tabla_base
    {add [column] definición_columna
     | alter [column] columna
            {set default {literal | función_sistema | null }
             | drop default}
     | drop [column] columna {restrict | cascade}  };
```

En ORACLE cambian algunas cosas

Para eliminar una tabla del esquema relacional:

```
drop table tabla_base {restrict | cascade};
```

2.5.1.4.- Definición de Restricciones (SQL)

create assertion *restricción*

check (expresión_condicional)

[comprobación_restricción];

La condición debe ser CIERTA.

2.5.1.4.- Ejemplo: Proveedor-Piezas-Suministro (SQL)

La restricción R2 :

R2) Px: Pieza, Sx: Suministro ∀Px : Pieza (∃Sx : Suministro(Sx) (Sx.código=Px.código))

se define mediante una restricción general:

```
create assertion R2 check
not exists(select * from Pieza P
        where not exists(select *
                from Suministro S
                where P.código=S.código));
```

Eliminación de una Restricción

DROP ASSERTION *restricción*

2.5.2.- SQL como lenguaje de manipulación de datos.

- El SQL como lenguaje de manipulación de datos incorpora:
  - La sentencia de consulta SELECT: Integración de las perspectivas lógica y algebraica.
  - Las sentencias de actualización de datos: INSERT, DELETE y UPDATE.

2.5.2.1- La sentencia SELECT

SELECT

- Permite recuperar información almacenada en una base de datos.
- Su sintaxis es:

```
5  select [all | distinct] comalista_ítem_seleccionado | *
1  from tabla
2  [where expresión_condicional]
3  [group by comalista_col]
4  [having expresión_condicional]
6  [order by comalista_referencia_col]
```

2.5.2.1- La sentencia SELECT

3 **select** R1X.A, R2X.B, ... , RnX.AA

1 **from** R1 [AS] R1X, R2 [AS] R2X, ... , Rn [AS] RnX

2 [**where** *F(R1X, R2X, ..., RnX)*]

donde:

- R1, R2, .., Rn son relaciones.
- A, B, ..., AA son atributos de las correspondientes relaciones.
- R1X, R2X, ..., RnX son nombres alternativos (alias).
- F(R1X, R2X, ..., RnX) es una condición.

El resultado es una relación formada por los atributos A, BB, ..., AA de las tuplas de las relaciones R1, R2, ..., Rn para las que F es cierta.

2.5.2.1- La sentencia SELECT.

RENOMBRAR

- Para realizar renombramientos en SQL se utiliza la palabra reservada AS.
- Permite el renombramiento de una relación así como de todos sus atributos (OJO: no cambia el esquema de la relación).

EJEMPLOS:

Jugador(nombre:varchar, edad: number, país:varchar)

Jugador AS Tenista ⟹ Renombra la relación *Jugador*

Jugador AS T(nom, ed, pa) ⟹ Renombra la relación *Jugador* y todos sus atributos.

2.5.2.1- La sentencia SELECT.

RENOMBRAR *(Cont)*

- La palabra reservada AS es opcional.
- En el caso de que una consulta haga referencia dos o más veces a una misma tabla, resulta imprescindible realizar un renombramiento.

EJEMPLO:

Jugador(dni:number, nombre:varchar, edad: number, país:varchar)

CP: {dni}

Obtén la lista de pares de nombres de jugadores del mismo país:

```
Select J1.nombre, J2.nombre
from Jugador AS J1, Jugador AS J2
where J1.país = J2.país and J1.dni < J2.dni;
```


2.5.2.1- La sentencia SELECT: La aproximación lógica.

3 **select** R1X.A, R2X.B, ... , RnX.AA

1 **from** R1 [AS] R1X, R2 [AS] R2X, ... , Rn [AS] RnX

2 [**where** *F(R1X, R2X, ..., RnX)*]

donde:

- En el SELECT se indican los atributos que se desean consultar.
- En la componente FROM se declaran variables de tipo tupla.
- WHERE es una fórmula lógica en las que las únicas variables libres son las declaradas en el FROM.
- La fórmula del WHERE se construye siguiendo la sintaxis usual de los lenguajes de 1er orden.

2.5.2.1- La sentencia SELECT: La aproximación lógica.

FORMALIZACIÓN (SINTAXIS):
FÓRMULAS DE LA CLÁUSULA WHERE.

Una **condición** es una expresión que puede ser:

- IS NULL (RX.Ai)
- RX.Ai α SX.Aj
- RX.Ai α *a*

donde:

- α es un operador de comparación ($<, >, \leq, \geq, =, \neq$).
- Ai y Aj son nombre de atributo de las relaciones sobre las que se han definido las variables RX y SX.
- *a* es un valor del dominio asociado al atributo RX.Ai (excepto el nulo).

2.5.2.1- La sentencia SELECT: La aproximación lógica.

Con lo dicho, las **fórmulas** se construyen aplicando las siguientes reglas:

- Toda condición es una fórmula.
- Si ***F*** es una fórmula, entonces ***(F)*** y *NOT* ***F*** son fórmulas.
- Si ***F*** y ***G*** son fórmulas, entonces también lo son ***F OR G***, ***F*** AND ***G***.
- Si ***S*** es una sentencia SELECT, entonces EXISTS(S) es una fórmula.
- Nada más es una fórmula.

2.5.2.1- La sentencia SELECT: La aproximación lógica.

3 **select** R1X.A, R2X.B, ... , RnX.AA

1 **from** R1 [AS] R1X, R2 [AS] R2X, ... , Rn [AS] RnX

2 [**where** *F(R1X, R2X, ..., RnX)*]

La sentencia SELECT devuelve una relación en la que cada tupla de la relación lo forman los valores de los atributos R1X.A, R2X.B, ... , RnX.AA de modo que:

- Estos valores aparecen en las variables R1X, R2X, ..., RnX.
- Por tanto, estos valores aparecen en las extensiones de las relaciones R1, R2, ..., Rn.
- Dichos valores hacen cierta la fórmula *F(R1X, R2X, ..., RnX)*.

2.5.2.1- La sentencia SELECT: La aproximación lógica.

EVALUACIÓN DE FÓRMULAS (SEMÁNTICA).

Valor de verdad de una condición:

- Si F es de la forma RX.Ai α SX.Aj entonces F se evalúa a indefinido si al menos un atributo Ai o Aj tiene el valor nulo en la tupla asignada a RX o a SX, en caso contrario se evalúa al valor de certeza de la condición.
- Si F es de la forma RX.Ai α *a* entonces F se evalúa a indefinido si Ai tiene valor nulo en la tupla asignada a RX, en caso contrario se evalúa al valor de certeza de la comparación.
- Si F es de la forma IS NULL(RX.Ai) entonces F se evalúa a cierto si Ai tiene el valor nulo para la tupla asignada a RX, en caso contrario se evalúa a falso.

2.5.2.1- La sentencia SELECT: La aproximación lógica.

Valor de verdad de una fórmula:

1) Sea *F* una condición, entonces su valor de verdad es el de la condición.

2) Si *F* es de la forma *(G)*, *F* se evalúa al valor de certeza de *G*.

3) Si *F* es de una de las siguientes formas NOT *G*, *G* AND *H* ó *G* OR *H* donde *G* y *H* son fórmulas, entonces *F* se evalúa de acuerdo a las siguientes tablas de verdad:

2.5.2.1- La sentencia SELECT: La aproximación lógica.

G	H	F =G AND H	F = G OR H
falso	falso	falso	Falso
indefinido	falso	falso	indefinido
cierto	falso	falso	cierto
falso	indefinido	falso	indefinido
indefinido	indefinido	indefinido	indefinido
cierto	indefinido	indefinido	cierto
falso	cierto	falso	cierto
indefinido	cierto	indefinido	cierto
cierto	cierto	cierto	cierto

G	F = NOT G
falso	cierto
indefinido	indefinido
cierto	falso

2.5.2.1- La sentencia SELECT: La aproximación lógica.

4) Si *F* es de la forma:

EXISTS(select *

from R1 [AS] R1X, R2 [AS] R2X, ... , Rn [AS] RnX

[**where** *G(R1X, R2X, ..., RnX)*])

entonces F se evalúa a cierto si existen valores de las variables R1X, ..., RnX de las extensiones de R1, ..., Rn para los cuales G se evalúa a cierto, en caso contrario se evalúa a falso.

2.5.2.1- La sentencia SELECT: La aproximación lógica.

EJEMPLO:

RÍO(rcod:dom_rcod, nombre:dom_nom)
PROVINCIA(pcod:dom_pcod, nombre:dom_nom)
PASA_POR(pcod:dom_pcod, rcod:dom_rcod)

Consulta1: “Provincias por las que pasa el río de código r1”.

Lógica de 1er orden:

Variables tupla:

Cláusula SELECT:

2.5.2.1- La sentencia SELECT: La aproximación lógica.

La sintaxis del cuantificador existencial en el lenguaje SQL:

EXISTS(SELECT *
FROM R1 R1X, R2 R2X, ..., Rn RnX
WHERE F(R1X, R2X, ..., RnX))

equivale a la fórmula $\exists$R1X:R1($\exists$R2X:R2 ...($\exists$RnX:Rn (F(R1X, R2X, ..., RnX))...)

En SQL no existe el cuantificador universal, se utiliza el existencial en su lugar mediante la conversión: $\forall x\ F(x) \equiv \neg\exists x\ (\neg\ F(x))$

2.5.2.1- La sentencia SELECT: La aproximación lógica.

EJEMPLO:

RÍO(rcod:dom_rcod, nombre:dom_nom)
PROVINCIA(pcod:dom_pcod, nombre:dom_nom)
PASA_POR(pcod:dom_pcod, rcod:dom_rcod)

Consulta3: "Obtener los ríos que pasan por todas las provincias".

Variables tupla:

Cláusula SELECT:

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

UNIÓN

- Permite fusionar el contenido de dos relaciones (o resultados de consultas) en una única tabla.
- La correcta ejecución de la operación de unión requiere que las dos relaciones que se unen sean compatibles.

EJEMPLO:

Cocinero(nombre:varchar, edad: number, país:varchar)
Camarero(nombre:varchar, edad: number, país:varchar)

Obtén la lista de trabajadores mayores de edad del restaurante:

Select nombre **from** Cocinero **where** edad >= 18
UNION
Select nombre **from** Camarero **where** edad >= 18;

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

DIFERENCIA

- La palabra reservada en SQL para realizar diferencias entre relaciones es EXCEPT.
- La correcta ejecución de la operación de diferencia requiere que las dos relaciones sean compatibles.

EJEMPLO:

Cocinero(nombre:varchar, edad: number, país:varchar)
Camarero(nombre:varchar, edad: number, país:varchar)

Obtén la lista de trabajadores que trabajan únicamente como cocineros en el restaurante:

1. **Select * from** (Cocinero **except** Camarero)
2. Cocinero **except** Camarero

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

INTERSECCIÓN

- La palabra reservada en SQL para realizar diferencias entre relaciones es INTERSECT.
- La correcta ejecución de la operación de diferencia requiere que las dos relaciones que intersectan sean compatibles.

EJEMPLO:

Cocinero(nombre:varchar, edad: number, país:varchar)

Camarero(nombre:varchar, edad: number, país:varchar)

Obtén la lista de trabajadores que trabajan como cocineros y camareros en el restaurante:

1. **Select * from** (Cocinero **intersect** Camarero)
2. Cocinero **intersect** Camarero

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

PRODUCTO CARTESIANO

- La correcta ejecución de la operación de producto cartesiano requiere que las relaciones que intervienen tengan diferentes nombres.
- En SQL el producto cartesiano de relaciones se aplica añadiendo dichas relaciones, separadas por comas, dentro de la cláusula FROM.

EJEMPLO:

Equipo1(nombre:varchar, edad: number, país:varchar)

Equipo2(nombre:varchar, edad: number, país:varchar)

Obtén todas las posibles combinaciones de jugadores del equipo 1 con jugadores del equipo 2:

Select * from Equipo1, Equipo2
Select * from Equipo 1 **CROSS JOIN** Equipo2

Obtén pares de jugadores del Equipo1 del mismo país:

Select * from Equipo1 e1, Equipo1 e2 **where** e1.país = e2.país and e1.nombre < e2.nombre

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

PROYECCIÓN

- Para proyectar basta con escribir el nombre de los atributos que se desean visualizar dentro de la cláusula SELECT separados por comas.
- Los atributos proyectados se pueden renombrar utilizando la cláusula AS.

EJEMPLO:

Cocinero(nombre:varchar, edad: number, país:varchar)

Obtén el nombre de los cocineros del restaurante:

Select nombre **from** cocinero

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

CONCATENACIÓN

- Corresponden a variantes del operador concatenación del Álgebra Relacional.
- Hay dos tipos básicos de concatenación en SQL: Interna y Externa.
- Concatenación interna:

  referencia_tabla [natural] [inner] join *referencia_tabla*
  [on *expresión_condicional* | using (*comalista_columna*)]

- Concatenación externa:

  referencia_tabla [natural]
  {left [outer] | right [outer] | full [outer]} JOIN *referencia_tabla*
  [on *expresión_condicional* | using (*comalista_columna*)]

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

CONCATENACIÓN *(Cont.)*

EJEMPLOS: Concatenación interna.

referencia_tabla [natural] [inner] join *referencia_tabla*
[on *expresión_condicional* | using (*comalista_columna*)]

PERSONA(nif: dom_nif, nombre: dom_nom, edad: dom_edad)

VIVIENDA(cod_viv: dom_cod, prop: dom_nif, dir: dom_dir, num_hab: dom_num)

- Obtener un listado en el que aparezca cada vivienda asociada con su propietario:

1. PERSONA inner join VIVIENDA on PERSONA.nif = VIVIENDA.prop
2. PERSONA natural inner join VIVIENDA AS V(cv, nif, dir, nh)
3. SELECT * FROM PERSONA, VIVIENDA WHERE nif = prop

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

CONCATENACIÓN *(Cont.)*

EJEMPLOS: Concatenación externa.

referencia_tabla [natural]
{left [outer] | right [outer] | full [outer]} JOIN *referencia_tabla*
[on *expresión_condicional* | using (*comalista_columna*)]

PERSONA(nif: dom_nif, nombre: dom_nom, edad: dom_edad)

VIVIENDA(cod_viv: dom_cod, nif: dom_nif, dir: dom_dir, num_hab: dom_num)

- Obtener un listado en el que aparezca cada vivienda asociada con su propietario:

1. PERSONA natural left join VIVIENDA ⟹ Aparecen todos los propietarios
2. PERSONA natural right join VIVIENDA ⟹ Aparecen todas las viviendas
3. PERSONA natural full join VIVIENDA ⟹ Aparecen todas las viviendas y todos los propietarios

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

CONCATENACIÓN *(Cont.)*

- Concatenación unión

... FROM T1 UNION JOIN T2 ≡ ... FROM
select t1.*, null, null, ..., null from t1
union all
select null, null, ..., null, t2.* from t2

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

SELECCIÓN

- La expresión del álgebra relacional:

R **DONDE** F(Ai, Aj, Ak, ...)

es equivalente a la expresión en SQL:

SELECT * FROM R **WHERE** F(R.Ai, R.Aj, R.Ak, ...)

- En el caso de que se incluyan varias relaciones en la cláusula FROM del SELECT:

SELECT * FROM R1, R2, ..., Rn **WHERE** F(R1.Ai, ..., Rn.Zk)

su equivalente en álgebra relacional sería:

R1 x R2 x ... x Rn **DONDE** F (R1.Ai, ..., Rn.Zk)

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

Operador	Álgebra Relacional	SQL
Selección	R DONDE F	SELECT ... FROM R WHERE F
Proyección	R [A_i, A_j ..., A_k]	SELECT A_i, A_j ..., A_k FROM R
Producto Cartesiano	R_1 x R_2, ... x R_n	SELECT ... FROM R_1, R_2, ..., R_n o SELECT...FROM R_1 CROSS JOIN R_2, ..., CROSS JOIN R_n
Concatenación	R_1 ⋈ R_2	SELECT... FROM R_1 NATURAL JOIN R_2
Unión	R_1 ∪ R_2	SELECT * FROM R_1 UNION SELECT * FROM R_2
Diferencia	R_1 - R_2	SELECT * FROM R_1 EXCEPT SELECT * FROM R_2
Intersección	R_1 ∩ R_2	SELECT * FROM R_1 INTERSECT SELECT * FROM R_2

2.5.2.1- La sentencia SELECT: La aproximación algebraica.

EJEMPLO:

RÍO(rcod:dom_rcod, nombre:dom_nom)

PROVINCIA(pcod:dom_pcod, nombre:dom_nom)

PASA_POR(pcod:dom_pcod, rcod:dom_rcod)

Consulta2: "Provincias por las que no pasa ningún río".

Álgebra Relacional: ⋈

SQL:

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

INSERT

- Sirve para insertar una o varias tuplas en una relación.

- Su sintaxis es:

  insert into tabla [(*comalista_columna*)]

  { default values | values (*comalista_átomos*) | *expresión_tabla*}

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

insert into tabla [(comalista_columna)]

{ default values | values (comalista_átomos) | expresión_tabla}

- Si no se incluye la lista de columnas se deberán insertar filas completas de *tabla*.

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

Cocinero(nombre:varchar, edad: number, país:varchar)

Nombre	Edad	País
. . .	. . .	. . .

INSERT INTO Cocinero

VALUES (“Carmelo Cotón”, 27, “Francia”);

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

Cocinero(nombre:varchar, edad: number, país:varchar)

Nombre	Edad	País
. . .	. . .	. . .
Carmelo Cotón	27	Francia
. . .	. . .	. . .

INSERT INTO Cocinero

VALUES (“Carmelo Cotón”, 27, “Francia”);

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

Cocinero(nombre:varchar, edad: number, país:varchar)

Nombre	Edad	País
. . .	. . .	. . .

INSERT INTO Cocinero(Edad, Nombre)

VALUES (27, “Carmelo Cotón”);

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

Cocinero(nombre:varchar, edad: number, país:varchar)

Nombre	Edad	País
. . .	. . .	. . .
Carmelo Cotón	27	?
. . .	. . .	. . .

INSERT INTO Cocinero(Edad, Nombre)

VALUES (27, “Carmelo Cotón”);

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

insert into tabla [(comalista_columna)]

{ default values | values (comalista_átomos) | expresión_tabla}

- Si no se incluye la lista de columnas se deberán insertar filas completas de *tabla*.
- Si se incluye la opción default values se insertará una única fila en la tabla con los valores por defecto apropiados en cada columna (según la definición de *tabla*).
- En la opción values(*comalista_átomos*) los átomos vienen dados por expresiones escalares.
- En la opción *expresión_tabla*, se insertarán las filas resultantes de la ejecución de la expresión (SELECT).

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

Cocinero(nombre:varchar, edad: number, país:varchar)

Nombre	Edad	País
. . .	. . .	. . .

Persona(nombre:varchar, edad: number)

Nombre	Edad
Paco	22
Antonio	19
Soledad	26

INSER INTO Cocinero(Nombre, Edad)

SELECT Nombre, Edad

FROM Persona

WHERE Edad > 20;

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

Cocinero(nombre:varchar, edad: number, país:varchar)

Nombre	Edad	País
. . .	. . .	. . .
Paco	22	?
Soledad	26	?
. . .	. . .	. . .

INSER INTO Cocinero(Nombre, Edad)

SELECT Nombre, Edad

FROM Persona

WHERE Edad > 20;

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

UPDATE

- Sirve para modificar los valores de los atributos de una o más tuplas seleccionadas.
- Su sintaxis es:

update *tabla*

set *comalista_asignaciones*

[where *expresión_condicional*]

donde una *asignación* es de la forma:

columna = {default | null | *expresión_escalar*}

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

Si se incluye la cláusula **where** sólo se aplicará a las filas que hagan cierta la condición.

EJEMPLO: Decrementar en 1 unidad la edad de los cocineros franceses.

UPDATE Cocinero **SET** Edad = Edad - 1

WHERE País = “Francia” ;

2.5.2.2- SQL como lenguaje de manipulación de datos: la actualización.

DELETE

- Elimina una o varias tuplas de una relación.
- Su sintaxis es:

  Delete from *tabla* [where *expresión_condicional*]

- Si se incluye la cláusula where se eliminarán aquéllas que hagan cierta la condición.

EJEMPLO: Eliminar la información de los cocineros menores de 18 años.

DELETE FROM Cocinero **WHERE** Edad < 18;