SQL

Parte I

Presentación

Los motores de bases de datos relacionales, utilizan SQL (Standard Query Language) como lenguaje para poder indicarle las sentencias a ejecutar.
El lenguaje SQL está estandarizado por normas ISO. Cada uno de los modelos de SGBD incorporan un determinado standard.

ISO: Versiones SQL

- *SQL-86*
- *SQL-89*
- *SQL-92*
- *SQL:1999*
- *SQL:2003*
- *SQL:2006*
- *SQL:2008*
- *SQL:2011*
- *SQL:2016*

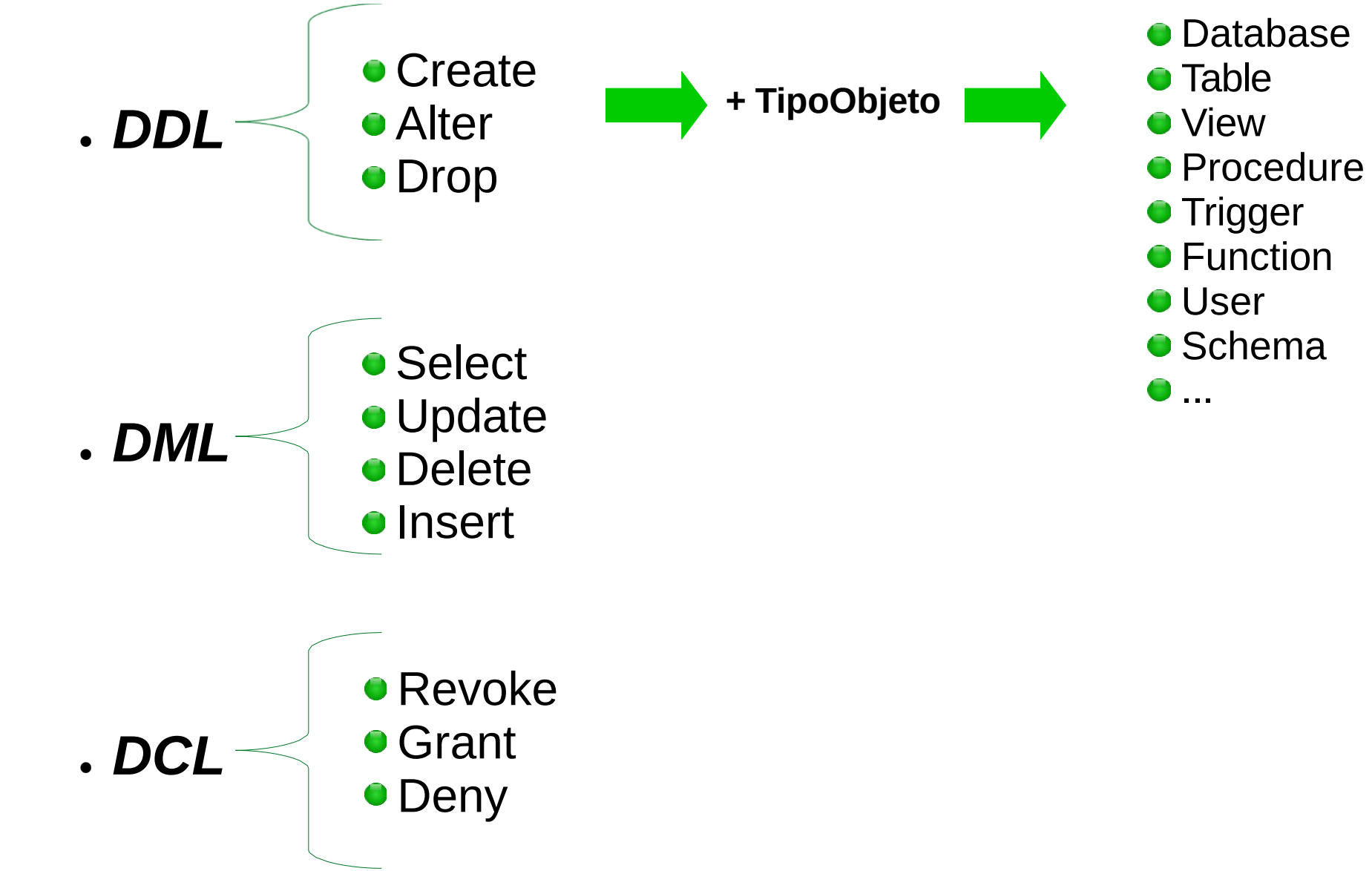
Clasificación

- ***DDL (Data Definition Language)****: Son todas aquellas instrucciones que permiten la definición de los distintos objetos de la base de datos.*

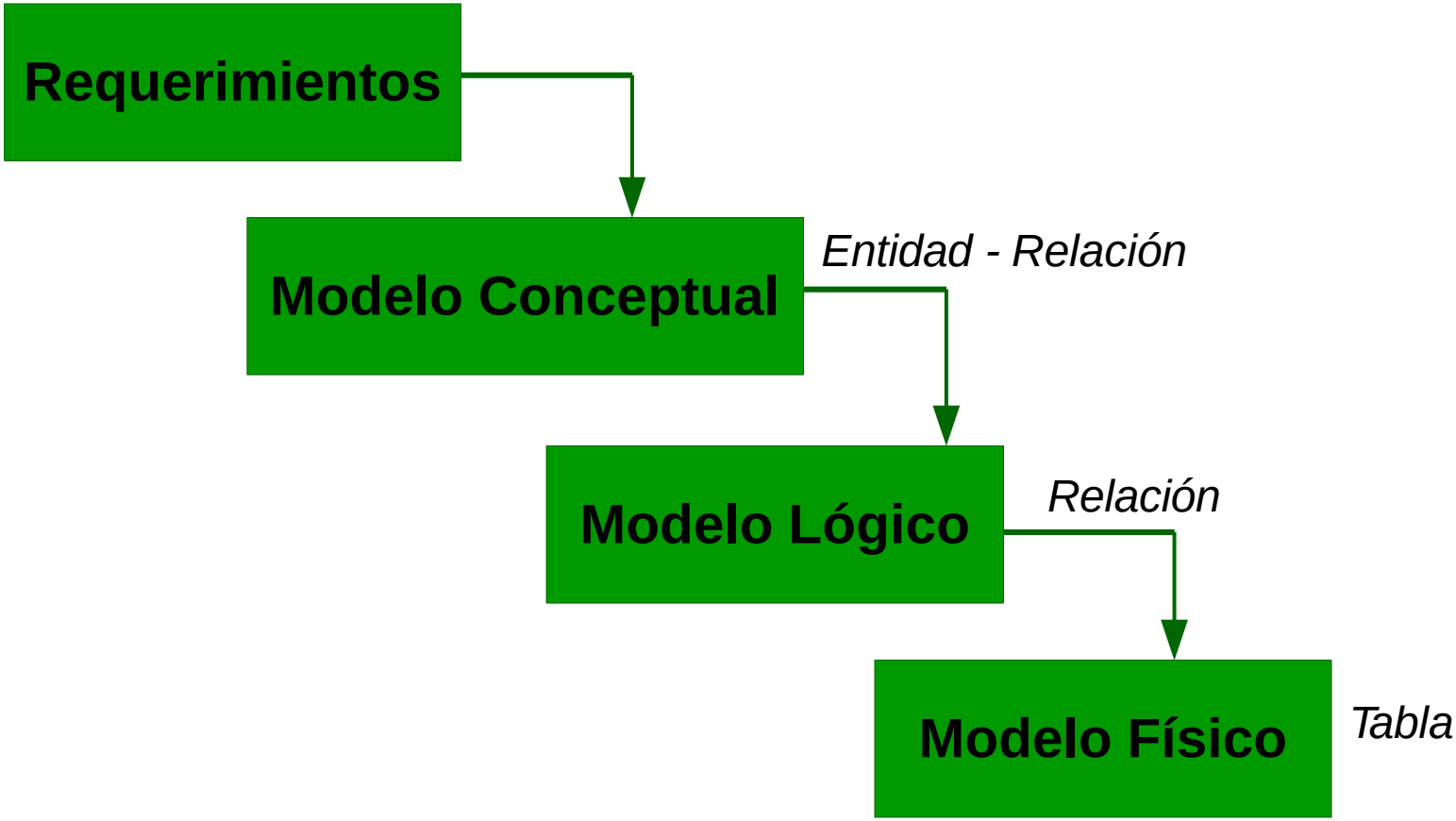
- ***DML (Data Manipulation Language)****: Son aquellas instrucciones que permite la manipulación de los datos almacenados en la base de datos.*

- ***DCL (Data Control Language)****: Son las aquellas instrucciones que permite el control de acceso a los datos.*

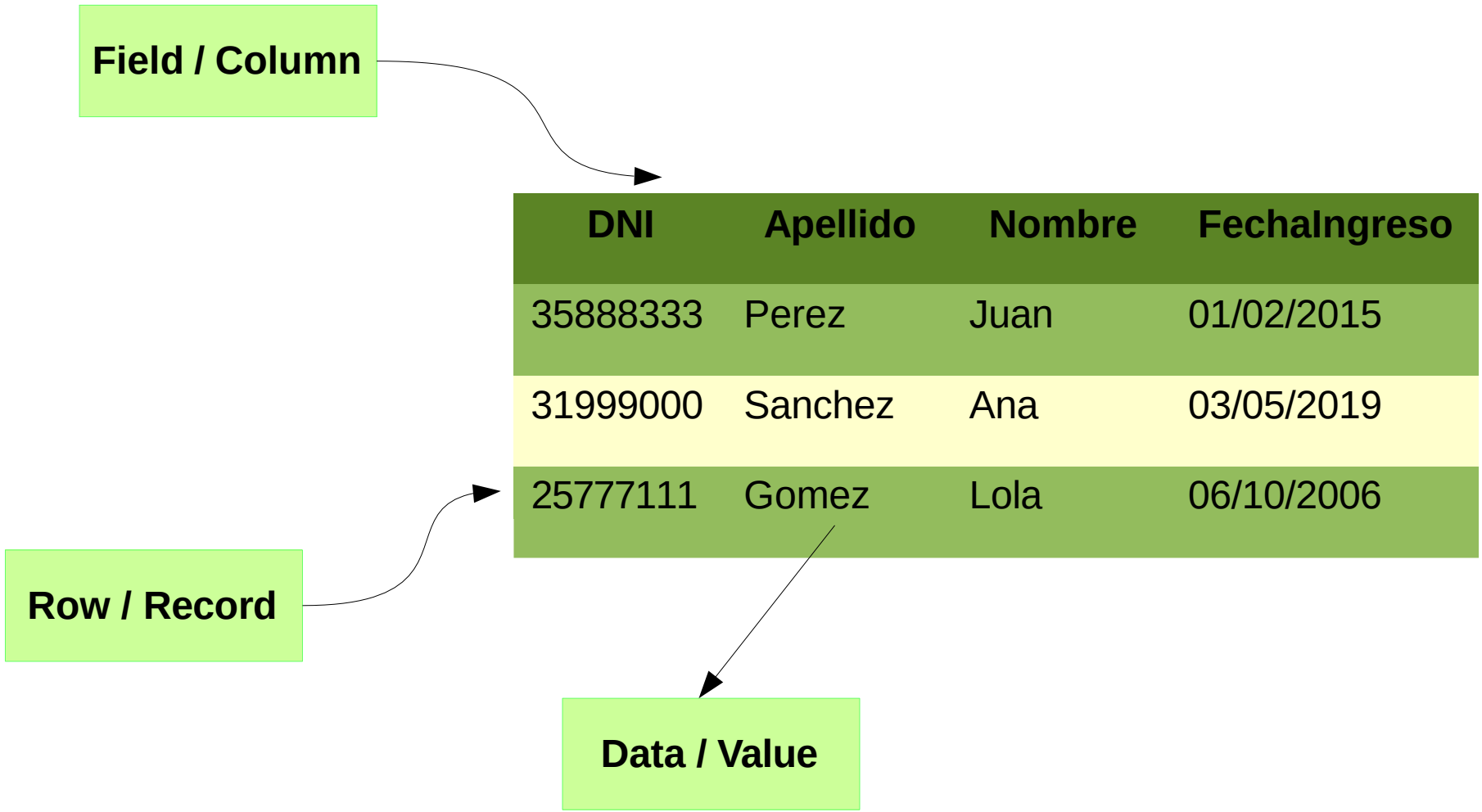
Clasificación



Etapas del diseño



DDL: Table


Field / Column
DNI
Apellido
Nombre
FechaIngreso
35888333
Perez
Juan
01/02/2015
31999000
Sanchez
Ana
03/05/2019
25777111
Gomez
Lola
06/10/2006
Row / Record
Data / Value

DDL: CREATE TABLE

- Sintaxis Simple:

```
CREATE TABLE NombreTabla
(NombreCampo1 tipodedatos1 [NULL|NOT NULL] [PRIMARY KEY],
 NombreCampo2 tipodedatos2 [NULL|NOT NULL] [PRIMARY KEY],
 …
 NombreCampoN tipodedatosN [NULL|NOT NULL] [PRIMARY KEY],
[CONSTRAINT NombreC PRIMARY KEY (NombreCampo1[,NombreCampoN])],
[CONSTRAINT NombreC FOREIGN KEY (NombreCampo1[,NombreCampoN])]
       REFERENCES NombreTablaC (NombreCampo1[,NombreCampoN])]
)
```

- Ejemplo Simple:

Alumno (Legajo, NyA, FechaIng, FechaNac, Mail)
Profesor (Legajo, NyA)
Curso (LegajoAlumno, LegajoProfesor)

```
CREATE TABLE Alumno
(Legajo int NOT NULL PRIMARY KEY,
 NyA varchar(100) NULL,
 FechaIng date,
 FechaNac date,
 Mail varchar(200))

 CREATE TABLE PROFESOR
 (Legajo int NOT NULL PRIMARY KEY,
  NyA varchar(100) NULL)
```

```
CREATE TABLE Curso
(LegajoAlumno int NOT NULL,
 LegajoProfesor int NOT NULL,
 CONSTRAINT PKCURSO PRIMARY KEY
        (LegajoAlumno, LegajoProfesor),
 CONSTRAINT FKALUMNO FOREIGN KEY
 (LegajoAlumno) REFERENCES Alumno(Legajo),
 CONSTRAINT FKPROFESOR FOREIGN KEY
 (LegajoProfesor) REFERENCES Profesor(Legajo)
```

DDL: CREATE TABLE

- **Tipos de Datos:**

Clasificación	Tipo de datos	Tamaños
Numérico Entero	bit Tinyint Smallint Int bigint	0-1 (1 byte) 0-255 (1 byte) -2^15 a 2^15 (2 bytes) -2^31 a 2^31 (4 bytes) -2^63 a 2^63 (8 bytes)
Numérico con Decimales	Numeric -Decimal Float – Real (n)	-2^38 a 2^38 (5-17bytes) 1-27 (7 digitos) (4 bytes) 23-53 (15 digitos) (8 bytes)
Texto	char (n) varchar(n) Text/varchar(MAX)	n=1 a 8000 2^31
Fecha/Hora	Date Time Smalldatetime Datetime Timestamp /rowversion	3 bytes 4 bytes 4 bytes 8 bytes 8-16 bytes
Otros	Binary XML	2^31 hasta 2 gb

** solo se citan los más relevantes*

DDL: CREATE TABLE

- Sintaxis Completa:

```
CREATE TABLE
    { database_name.schema_name.table_name | schema_name.table_name | table_name }
    [ AS FileTable ]
    ( {   <column_definition>
        | <computed_column_definition>
        | <column_set_definition>
        | [ <table_constraint> ] [ ,... n ]
        | [ <table_index> ] }
          [ ,...n ]
          [ PERIOD FOR SYSTEM_TIME ( system_start_time_column_name
             , system_end_time_column_name ) ]
      )
    [ ON { partition_scheme_name ( partition_column_name )
           | filegroup
           | "default" } ]
    [ TEXTIMAGE_ON { filegroup | "default" } ]
    [ FILESTREAM_ON { partition_scheme_name
           | filegroup
           | "default" } ]
    [ WITH ( <table_option> [ ,...n ] ) ]
[ ; ]
```

```
<column_definition> ::=
column_name <data_type>
    [ FILESTREAM ]
    [ COLLATE collation_name ]
    [ SPARSE ]
    [ MASKED WITH ( FUNCTION = ' mask_function ')
    [ CONSTRAINT constraint_name [ DEFAULT consta
```

```
<data_type> ::=
[ type_schema_name . ] type_name
    [ ( precision [ , scale ] | max |
        [ { CONTENT | DOCUMENT } ] xml_schema_c
```

DDL: CREATE TABLE

- **Ejemplo**:

  Proyecto (ID, Nombre, fechainicio, HsTotales)
  Cargo (ID, Descripcion)
  Empleado (TipoDoc,NroDoc, NyA, Fingreso, Sueldo, IDCargo)
  Trabaja (IDProyecto, TipoDoc, NroDoc, fechaInicio)

```
CREATE TABLE PROYECTO
(ID int not null primary key,
 Nombre varchar(60),
 Fechainicio date,
 HsTotales smallint)

CREATE TABLE EMPLEADO
(TipoDoc tinyint not null,
  Nrodoc bigint not null,
  Nya varchar(150),
  Fingreso date,
  Sueldo numeric(12,2),
  IDCargo smallint not null,
  constraint PKEmpleado primary key (Nrodoc, tipodoc),
  constraint FK1Empleado foreign key (IDCargo)
              references Cargo (ID)
)
```

```
CREATE TABLE CARGO
(ID smallint not null primary key,
 Descripcion varchar(60))

CREATE TABLE TRABAJA
( IDProyecto int not null,
  TipoDoc tinyint not null,
  NroDoc bigint not null,
  FechaInicio date,
  constraint PKTrabaja
        primary key (idproyecto,tipodoc,nrodoc),
  constraint FK1Trabaja foreign key (idproyecto)
        references Proyecto (ID),
  constraint FK2Trabaja foreign key (nrodoc,tipodoc)
        references Empleado (nrodoc,tipodoc))
```

DDL: DROP TABLE

- **Sintaxis Simple:**

```
DROP TABLE NombreTabla
```

- **Ejemplo Simple:**

```
DROP TABLE Curso

DROP TABLE Alumno
```

DDL: DROP TABLE

- **Sintaxis Completa:**

```
DROP TABLE [ IF EXISTS ] { database_name.schema_name.table_name | schema_name.table_name |
[ ; ]
```

DDL: ALTER TABLE

- **Sintaxis Simple:**

```
ALTER TABLE Nombretabla
    ADD NombreCampo tipodatos [modificadores]

ALTER TABLE Nombretabla
    DROP COLUMN NombreCampo [,NombreCampoN]

ALTER TABLE Nombretabla
    ALTER COLUMN NombreCampo tipodatos [modificadores]

ALTER TABLE Nombretabla
    ADD CONSTRAINT nombreC [PRIMARY KEY|FOREIGN KEY] …

ALTER TABLE Nombretabla
    DROP CONSTRAINT nombreC
```

- **Ejemplo Simple:**

```
ALTER TABLE Empleado ADD Mail varchar(200)
ALTER TABLE Empleado DROP COLUMN Mail

ALTER TABLE Empleado ADD CONSTRAINT PKEmpleado PRIMARY KEY (NroDoc, TipoDoc)
ALTER TABLE Empleado DROP CONSTRAINT FKCargo
```

DDL: ALTER TABLE

- **Sintaxis Completa:**

```
ALTER TABLE { database_name.schema_name.table_name | schema_name.table_name | table_name }
{
    ALTER COLUMN column_name
    {
        [ type_schema_name. ] type_name
            [ (
                {
                   precision [ , scale ]
                 | max
                 | xml_schema_collection
                }
            ) ]
        [ COLLATE collation_name ]
        [ NULL | NOT NULL ] [ SPARSE ]
      | { ADD | DROP }
          { ROWGUIDCOL | PERSISTED | NOT FOR REPLICATION | SPARSE | HIDDEN }
      | { ADD | DROP } MASKED [ WITH ( FUNCTION = ' mask_function ') ]
    }
    [ WITH ( ONLINE = ON | OFF ) ]
    | [ WITH { CHECK | NOCHECK } ]

    | ADD
    {
        <column_definition>
      | <computed_column_definition>
      | <table_constraint>
      | <column_set_definition>
    } [ ,...n ]
      | [ system_start_time_column_name datetime2 GENERATED ALWAYS AS ROW START
                [ HIDDEN ] [ NOT NULL ] [ CONSTRAINT constraint_name ]
            DEFAULT constant_expression [WITH VALUES] ,
                system_end_time_column_name datetime2 GENERATED ALWAYS AS ROW END
                   [ HIDDEN ] [ NOT NULL ][ CONSTRAINT constraint_name ]
            DEFAULT constant_expression [WITH VALUES] ,
        ]
       PERIOD FOR SYSTEM_TIME ( system_start_time_column_name, system_end_time_column_name )
    | DROP
     [ {
         [ CONSTRAINT ][ IF EXISTS ]
```

DDL: TABLE

- **Resumen de comandos:**

  - `CREATE TABLE` *`Nombretabla ...`*
  - `ALTER TABLE` *`Nombretabla ...`*
  - *`DROP TABLE Nombretabla ...`*

DML

- Select
- Update
- Delete
- Insert

DML: Select

- **Sintaxis Simple:**

```
SELECT  [top n] [distinct] [* | <lista de campos>]
FROM tabla1 [,tabla2, … tablan]
[WHERE <condicion> [AND|OR <condicion>]]
[ORDER BY campo1 [asc|desc] [,campo2 [asc|desc] ...]]
```

- **Ejemplo Simple:**

```
SELECT  * FROM empleado

SELECT  mail FROM empleado WHERE legajo=1

SELECT  mail,nya FROM empleado WHERE legajo>6 and nac='AR'

SELECT  nya FROM empleado WHERE legajo>=100 order by nya

SELECT  nya FROM empleado order by legajo desc

SELECT  nya FROM empleado order by nya desc, legajo asc

SELECT TOP 10 legajo FROM empleado

SELECT distinct nya FROM empleado
```

DML: Select. Condiciones

Las condiciones armadas en un where, podrían utilizar cualquiera de los siguientes Operadores:

Operador	Ejemplo
=	legajo = 1
<>	nombre <> 'Juan'
>	fechaNac > '2015-01-01'
<	lejajo < 100
>=	fechaing >='2020-01-01'
<=	sueldo <= 20000
Like	nya like 'Perez%' / nya like '%perez%' / nya like '_perez'
between	Legajo between 20 and 50
IN (<lista de valores>)	legajo IN (10,20,30) / nombre IN ('Juan','Ana','Lola')
Is / is not	mail is null / mail is not null

DML: Like - IN

- **Like**: Es un operador que permite utilizar comodines.

Por ejemplo:

```
SELECT * FROM EMPLEADO WHERE APELLIDO LIKE 'PEREZ%'

SELECT * FROM EMPLEADO WHERE APELLIDO LIKE 'PEREZ_'
```

- **IN**: Es un operador que permite realizar comparaciones de OR con uno o más valores de la lista.

Por ejemplo:

```
SELECT * FROM EMPLEADO WHERE APELLIDO IN ('Perez', 'Lopez')

SELECT * FROM EMPLEADO WHERE IDCargo IN (1,2,10,55)

SELECT * FROM EMPLEADO WHERE IDCargo NOT IN (1,2,10,55)
```

DML: Alias

Los nombres de los campos y las tablas se pueden llamar a través de un alias.
Para las tablas, esto permite que podamos referenciarla por el alias en lugar de usar el nombre de la tabla.
En el caso de los campos, también podemos colocarle un alias y en el caso que se encuentren en el Select, mostrará con ese valor la salida.

```
SELECT campo [as] aliascampo

FROM tabla [as] aliastabla
```

- **Ejemplo Simple:**

```
SELECT e.nombreyapellido as nya, e.legajo as leg
FROM empleado e
Where e.legajo between 10 and 20


SELECT e.* FROM EMPLEADO e
```

DML: Any. Some. All

Estas sentencias permiten comparar un campo, utilizando cualquier operador, con un conjunto de valores devueltos a través de una consulta dinámica.
Any y ***Some*** actúan de igual modo y sólo existen ambos por compatibilidad.

```
campo <operador> ANY (<sentencia select>)

campo <operador> SOME (<sentencia select>)

campo <operador> ALL (<sentencia select>)
```

- **Ejemplo Simple:**

```
SELECT * FROM empleado WHERE legajo = ANY (select legajo from hist)

SELECT * FROM empleado WHERE legajo = SOME (select legajo from hist)

SELECT * FROM empleado WHERE legajo >= ANY (select legajo from hist)

SELECT * FROM empleado WHERE legajo >= ALL (select legajo from hist)
```

DML: In. Exists.

Estos operadores permiten también verificar si un valor escalar o campo se encuentra dentro de un conjunto de valores. Estos valores podrán estar compuestos por el resultado de una consulta o una lista estática, pero en este caso sólo funcionará igualdad.

```
campo IN (<sentencia select>|<lista de valores>)

campo NOT IN (<sentencia select>|<lista de valores>)

exists (<sentencia select>)

not exists (<sentencia select>)
```

- **Ejemplo Simple:**

```
SELECT * FROM empleado WHERE legajo IN (10,20,30)

SELECT * FROM empleado WHERE legajo IN (select legajo from hist)

SELECT * FROM empleado WHERE legajo NOT IN (40,50,60)

SELECT * FROM empleado WHERE exists (select 1 from hist
                                     where empleado.legajo=hist.legajo)
```

DML: Union. Union all

Estos operadores permiten unir el resutlado de dos o más consultas distintas.
Las condiciones que deben tener estas consultas, es que deben ser de igual grado (misma cantidad de campos) e igual dominio de los campos de igual orden.
La diferencia entre Union y Union all, es que Union permite anular los duplicados, realizando un distinct luego de la union.

```
SELECT campo1, campo2 FROM tabla1

[UNION ALL|UNION]

SELECT campo1, campo2 FROM tabla2
```

- **Ejemplo Simple:**

```
SELECT legajo,fechaingreso from empleado1

UNION

SELECT leg, fing FROM empleado2
```

DML: Intersect

Este operador devuelve todas las filas en común de las tablas involucradas en la operación. También deben ser compatibles, es decir, igual grado e igual dominio.

```
SELECT campo1, campo2 FROM tabla1

INTERSECT

SELECT campo1, campo2 FROM tabla2
```

- **Ejemplo Simple:**

```
SELECT legajo,fechaingreso from empleado1

INTERSECT

SELECT leg, fing FROM empleado2
```

DML: Except

Este operador devuelve todas las filas que no están en el resto de las tablas de la operación. Simula ser una resta de las operaciones algebráicas.
También deben ser compatibles, es decir, igual grado e igual dominio.

```
SELECT campo1, campo2 FROM tabla1

EXCEPT

SELECT campo1, campo2 FROM tabla2
```

- **Ejemplo Simple:**

```
SELECT legajo,fechaingreso from empleado1

EXCEPT

SELECT leg, fing FROM empleado2
```

DML: Join

Las juntas permiten unir tuplas de distintas relaciones para generar una nueva tupla.

- **Tipos de Joins:**
  - Inner Join
  - Left Join
  - Right Join
  - Cross Join
  - Full Join

- **Sintaxis:**

```
TABLA1 T1 [INNER|LEFT|RIGHT|CROSS|FULL] JOIN TABLA2 T2
ON T1.CAMPO1=T2.CAMPO2
```

DML: Inner Join

Este tipo de junta, solo devuelve una tupla cuando encuentra exactamente una coincidencia en la otra relación.

Empleado

Legajo	NyA
L1	Juan
L2	Ana
L3	Lola
L4	Pedro
L5	Martín

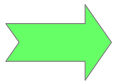
Trabaja

Legajo	IDP
L1	PR1
L2	PR2
L3	PR1
L3	PR2

Proyecto

IDP	Desc
PR1	Migración
PR2	Analisis
PR3	Patch

```
SELECT e.legajo, e.NyA
from empleado e
    inner join trabaja t on e.legajo=t.legajo
```



Legajo	NyA
L1	Juan
L2	Ana
L3	Lola
L3	Lola

DML: Left Join

Este tipo de junta, solo devuelve una tupla cuando encuentra en la primer tabla (left table) del Join, independientemente que no exista en la segunda tabla.

Empleado

Legajo	NyA
L1	Juan
L2	Ana
L3	Lola
L4	Pedro
L5	Martín

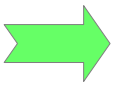
Trabaja

Legajo	IDP
L1	PR1
L2	PR2
L3	PR1
L3	PR2

Proyecto

IDP	Desc
PR1	Migración
PR2	Analisis
PR3	Patch

```
SELECT e.legajo, t.idp
from empleado e
    left join trabaja t on e.legajo=t.legajo
```



Legajo	IDP
L1	PR1
L2	PR2
L3	PR1
L3	PR2
L4	null
L5	null

DML: Right Join

Este tipo de junta, solo devuelve una tupla cuando encuentra en la segunda tabla (right table) del Join, independientemente que no exista en la primer tabla.

Empleado

Legajo	NyA
L1	Juan
L2	Ana
L3	Lola
L4	Pedro
L5	Martín

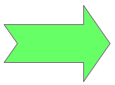
Trabaja

Legajo	IDP
L1	PR1
L2	PR2
L3	PR1
L3	PR2

Proyecto

IDP	Desc
PR1	Migración
PR2	Analisis
PR3	Patch

```
SELECT e.legajo, t.idp
from trabaja t
    right join empleado e on e.legajo=t.legajo
```



Legajo	IDP
L1	PR1
L2	PR2
L3	PR1
L3	PR2
L4	null
L5	null

DML: Full Join

Este tipo de junta, solo devuelve una tupla cuando encuentra en la segunda tabla (right table) del Join o en la primer tabla (left table). Aquellos datos que no pueda completar porque no exista coincidencia en la tupla, se visualizará con null.

Empleado

Legajo	NyA
L1	Juan
L2	Ana
L3	Lola
L4	Pedro
L5	Martín

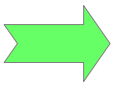
Trabaja

Legajo	IDP
L1	PR1
L2	PR2
L3	PR1
L3	PR2

Proyecto

IDP	Desc
PR1	Migración
PR2	Analisis
PR3	Patch

```
SELECT e.nya, p.desc
from trabaja t
    full join empleado e on e.legajo=t.legajo
    full join proyecto p on t.IDP=p.IDP
```



NyA	Desc
Juan	Migracion
Ana	Analisis
Lola	Migracion
Lola	Analisis
Pedro	null
Martin	null
null	Patch

DML: Cross Join

Realiza un producto cartesiado con ambas tablas. En este caso el cross join no tiene condición de junta, ya que junta todas las tuplas.

Empleado

Legajo	NyA
L1	Juan
L2	Ana
L3	Lola
L4	Pedro
L5	Martín

Trabaja

Legajo	IDP
L1	PR1
L2	PR2
L3	PR1
L3	PR2

Proyecto

IDP	Desc
PR1	Migración
PR2	Analisis
PR3	Patch

```
SELECT e.legajo, p.idp
from empleado e
    cross join proyecto p
Where e.legajo in ('L1','L2')
```

Legajo	IDP
L1	PR1
L1	PR2
L1	PR3
L2	PR1
L2	PR2
L2	PR3

Revisión SQL I

- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- SELECT ... FROM
- SELECT … FROM … WHERE ...
- ORDER BY
- DISTINCT
- TOP
- ALIAS
- ANY. SOME. ALL
- IN. EXISTS
- UNION. UNION ALL
- INTERSECT
- EXCEPT
- JOIN. INNER. LEFT. RIGHT. FULL. CROSS

Consultas

Gracias.