

BASE DE DATOS

UNLaM

Materia de la carrera Ingeniería en Informática
Resúmenes, apuntes y modelos de exámenes del primer nivel de base de datos

Gonzalez, Nazarena Araceli
araceli009uo@gmail.com



BASE DE DATOS	5
UNIDAD N°1: DIAGRAMA DE ENTIDAD RELACION (DER)	5
• Introducción	5
Base de datos relacionales	5
• DER (Diagrama de Entidad Relación).....	6
ENTIDAD	6
Entidad débil	6
Jerarquías	6
▪ Jerarquía de Generalización	6
▪ Jerarquía de Subconjuntos.....	7
ATRIBUTO	7
Simple ó Compuesto	7
Monovaluado ó Multivaluado	7
No derivado ó Derivado/Calculable	8
No Complejo ó Complejo	8
Nulo/Opcional ó No Nulo/Obligatorio	8
Clave ó No Clave	8
▪ Superclave	8
▪ Clave candidata	8
▪ Clave Primaria;	9
RELACION.....	9
Restricción de Cardinalidad	9
Restricción de Participación	10
Grado de una relación	10
Relaciones Redundantes/Transitivas	11
Múltiples relaciones entre las mismas Entidades	11
• Dominios	11
• Etapas en el diseño de una base de datos	11
Análisis de Requerimientos	11
Modelo Conceptual	11
Modelo Lógico	12
Modelo Relacional	12
UNIDAD N°2: MODELO RELACIONAL.....	13
Primary Key (PK).....	13
Foreign Key (FK)	13
Relación.....	13
Restricciones	13
➤ Restricción de Dominio	13
➤ Restricción de Integridad de Entidades	13
➤ Restricción de Integridad Referencial	13
Pasaje del Modelo DER al MR	13
Reglas de conversión.....	13
ENTIDAD	13
ENTIDAD	13
➤ Entidad Débil	13
➤ Jerarquías.....	14
ATRIBUTOS.....	14
➤ N-->N	14
➤ 1-->N ó N-->1.....	14



➤ 1-->1	14
RELACION.....	15
➤ Relaciones Binarias.....	15
N-->1.....	15
1-->N.....	15
1-->1.....	15
➤ Relaciones Unarias	15
➤ Relaciones Ternarias	16
N-->N-->N	16
N-->N-->1	16
N-->1-->1.....	16
1-->1-->1	16
UNIDAD N°3: ALGEBRA RELACIONAL	18
Clasificación	18
Operaciones de AR:	18
➤ Unarias	18
➤ Binarias.....	18
➤ Primitivas.....	18
➤ De Base de Datos.....	18
SELECCIÓN	18
PROYECCIÓN	19
Relaciones Intermedias y Renombre de atributos.....	20
UNION.....	20
INTERSECCIÓN	21
Intersección con Operadores Básicos	22
DIFERENCIA.....	22
PRODUCTO CARTESIANO	22
JUNTAS.....	23
➤ JUNTA THETA.....	23
Junta Theta con Operadores Básicos	24
➤ JUNTA NATURAL	24
Junta Natural con Operadores Básicos	24
Operaciones Básicas y Derivadas.....	24
➤ Básicas	24
➤ Derivadas	24
EJERCICIOS DE PARCIAL	25
Hallar el menor / mayor	25
Contar.....	26
UNIDAD N°4: NORMALIZACION	28
• Introducción a la Normalización.....	28
DEPENDENCIA FUNCIONAL	29
DEPENDENCIA FUNCIONAL TRIVIAL	30
CUESTIONES A TENER EN CUENTA:	30
CLAVE Y SUPERCLAVE.....	30
AXIOMAS DE ARMSTRONG	31
Reflexividad	31
Aumento	31
Transitividad	31
REGLAS ADICIONALES/DERIVADAS	31



Descomposición.....	31
Unión	31
Pseudotransitividad	31
CLAUSURA DE UN CONJUNTO DE DEPENDENCIAS (F+)	32
CLAUSURA DE UN CONJUNTO DE ATRIBUTOS (X+).....	32
EQUIVALENCIA ENTRE DOS CONJUNTOS DE DEPENDENCIAS FUNCIONALES.....	33
UNIDAD N°5: SQL.....	34
● Introducción al SQL	34
Clasificación	34
DDL (Data Definition Language)	34
DML (Data Manipulation Language)	34
DCL (Data Control Language)	34
Etapas del diseño	35
DDL	35
CREATE TABLE	35
Tipos de Datos	36
DROP TABLE.....	36
ALTER TABLE.....	36
DML.....	37
Select	37
Condiciones.....	37
Like.....	38
IN	38
Alias.....	38
Any. Some. All	39
In. Exists	39
Union. Union all.....	40
Intersect	40
Except.....	40
Join	40
➤ Inner Join.....	40
➤ Left Join	41
➤ Right Join	41
➤ Cross Join.....	42
➤ Full Join.....	42
Update.....	43
Delete	43
Insert	44
TRUNCATE	44
Funciones de agregación	45
➤ SUM()	45
➤ MAX()	45
➤ MIN()	45
➤ AVG()	45
➤ COUNT()	45
COUNT(*).....	45
COUNT(nombre columna)	46
➤ GROUP BY.....	46
➤ HAVING	47
➤ ORDER BY	47



Vistas	48
UNIDAD N°6: SQL AVANZADO.....	50
Stored Procedure	50
Stored Procedure: Crear	50
Stored Procedure: Borrar	50
Stored Procedure: Cambiar	51
Stored Procedure: Ejecutar.....	51
Stored Procedure: Variables.....	51
Stored Procedure: While	51
Stored Procedure: if.....	52
Ejemplos de Stored procedure	52
Function	53
Crear Function.....	53
Invocar Function.....	53
Borrar Function.....	54
Cambiar Function.....	54
Trigger	54
Trigger: Crear	54
Trigger: Borrar	54
Trigger: Cambiar	55
Trigger: Habilitar / Deshabilitar	55



BASE DE DATOS

UNIDAD N°1: DIAGRAMA DE ENTIDAD RELACION (DER)

- Introducción

Un sistema de bases de datos es básicamente un sistema computarizado para llevar registros.

Aspecto de integración de datos: queremos decir que podemos imaginar a la base de datos como una unificación de varios archivos que de otro modo serían distintos, con una redundancia entre ellos eliminada al menos parcialmente.

Aspecto de compartimentación de datos: queremos decir que las piezas individuales de datos en la base pueden compartidas entre diferentes usuarios y que cada uno de ellos puede tener acceso a la misma pieza de datos, probablemente con fines diferentes.

Sistema de administración de base de datos (DBMS): es el software que maneja todo acceso a la base de datos. Eliminar archivos (o tablas), recuperar y almacenar datos desde y en dichos archivos, etcétera.

Un sistema de un solo usuario es aquel en el que sólo un usuario puede tener acceso a la base de datos en un momento dado

Un sistema multiusuario es aquel en el cual múltiples usuarios pueden tener acceso simultáneo a la base de datos.

Una *entidad* es cualquier objeto acerca del cual queremos registrar información.

Las entidades (incluidos los vínculos) poseen *propiedades* que corresponden a la información que deseamos registrar sobre ellas.

Beneficios del enfoque de base de datos

- Los datos pueden compartirse
- Es posible reducir la redundancia¹
- Es posible brindar un manejo de transacciones²
- Es posible mantener la integridad
- Es posible hacer cumplir la seguridad
- Es posible equilibrar los requerimientos en conflicto

Base de datos relacionales

Operaciones restringir, proyectar y juntar aplicadas a la base de datos

¹ Redundancia: Sobra o demasiada abundancia de cualquier cosa o en cualquier línea.

² Transacciones: Una transacción es una unidad de trabajo lógica, que por lo regular comprende varias operaciones de la base de datos. El ejemplo común es el de transferir una cantidad de efectivo de una cuenta A a otra cuenta B. Es claro que aquí se necesitan dos actualizaciones, una para retirar el efectivo de la cuenta A y la otra para depositarlo en la cuenta B. Si el usuario declara que las dos actualizaciones son parte de la misma transacción, entonces el sistema puede en efecto garantizar que se hagan ya sea ambas o ninguna de ellas,



- La operación restringir (también conocida como seleccionar) extrae las filas especificadas de una tabla.
- La operación proyectar extrae las columnas especificadas de una tabla.
- La operación juntar reúne dos tablas con base en valores comunes de una columna común.

Todas las operaciones **se realizan un conjunto a la vez**, no una fila a la vez. Entonces para poder filtrar los datos tenemos que pensar que nuestra consulta SQL se va a aplicar A TODOS LOS DATOS AL MISMO TIEMPO.

Esto significa que los completas en vez de sólo filas individuales, y que las tablas contienen conjuntos de filas. (Por supuesto, una tabla que contiene un conjunto de una sola fila es legal, como lo es también una tabla vacía, es decir, una tabla que no contiene fila alguna.)

Tener en cuenta esto cuando realizamos las consultas en BDD, porque al trabajar con conjuntos de datos **NO** podemos poner las mismas condiciones que en programación.

- [DER \(Diagrama de Entidad Relación\)](#)

Es un modelo conceptual que permite diseñar una base de datos. Está basado en una percepción del mundo real, utilizando objetos llamados entidades y relaciones.

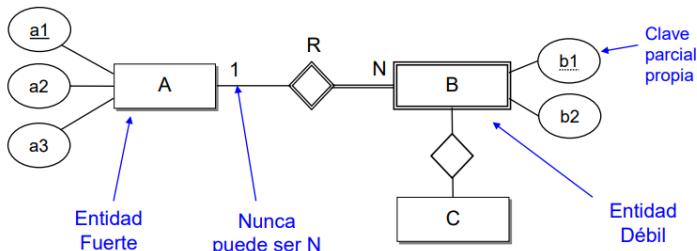
[ENTIDAD](#)

Es una cosa u objeto del mundo real que se distingue por sus cualidades propias. Puede ser física o abstracta

Nombre de la entidad

Entidad débil

Es una entidad cuya clave está conformada por atributos externos (provenientes de otras entidades), combinados o no con atributos propios.



La **participación** de la Entidad Débil en la relación con su Entidad Fuerte es siempre **TOTAL**.

[Jerarquías](#)

Hay dos tipos de Jerarquías:

- [Jerarquía de Generalización](#)

Sin Solapamiento: Los subconjuntos son disjuntos

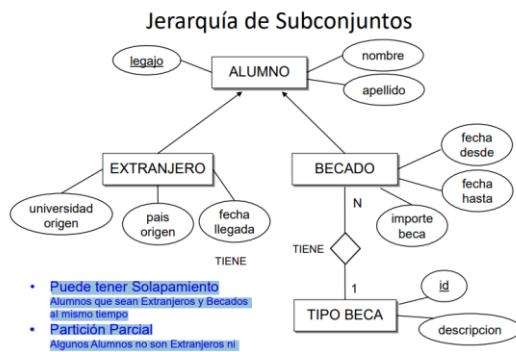
Partición Total: Todos los vehículos son de algún tipo



■ Jerarquía de Subconjuntos

Puede tener Solapamiento: Alumnos que sean Extranjeros y Becados al mismo tiempo •

Partición Parcial: Algunos Alumnos no son Extranjeros ni Becados



ATRIBUTO

Describen las cualidades de los objetos del diagrama.

Atributo

Simple ó Compuesto

Simple: Son atributos atómicos, no divisibles.

Apellido

Compuestos: Son atributos que pueden dividirse en otros con significado propio. El valor compuesto es la concatenación de todos sus componentes.



Monovaluado ó Multivaluado

Monovaluado: Son atributos que poseen un único valor.

Legajo

Multivaluado: Son atributos que poseen más de un valor posible.



No derivado ó Derivado/Calculable

No Derivado: Son atributos que su valor no puede ser calculado.



Derivado: Son atributos que su valor puede obtenerse de algún cálculo de un atributo y/o de algún cálculo.



No Complejo ó Complejo

No Complejo: Son atributos que admite un único valor y son atómicos.



Complejo: Son atributos que pueden adquirir más de un valor y que su valor se descompone en otros con significado propio. Es una combinación de multivaluado con compuesto.



Nulo/Opcional ó No Nulo/Obligatorio

El valor Nulo (null) es utilizado cuando no se conoce el dato

Clave ó No Clave

- **Superclave:** es un conjunto de atributos que nos permite identificar a una entidad dentro del conjunto de entidades de forma única. Si separo los dos atributos que conforman la superclave, y uno de ellos es clave entonces sé que es una superclave. Ej:

Superclave = AB

Atributo normal = B

Clave = A

Entonces "AB" es una superclave porque al separar los atributos que la componen sigue teniendo una clave.

- **Clave candidata**
-Es una superclave
No debe existir un subconjunto que también sea único



- **Clave Primaria:** Es la clave candidata elegida por el diseñador de la base de datos para una entidad. Ej:

Clave = AB

Atributo normal = B

Atributo normal = A

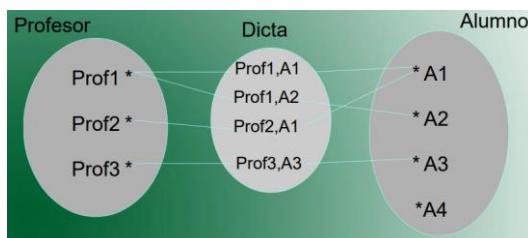
Entoncessss “AB” es una clave porque al separar los atributos que la componen no existe uno de ellos que sea una clave. Si lo voy subdividiendo no voy a encontrar algo que sea clave.

RELACION

Es una asociación entre diferentes entidades.



Las relaciones establecen las asociaciones entre las entidades.



Restricción de Cardinalidad: Es la cantidad de instancias que pueden vincularse en una relación.

- **Muchos a Muchos ($N \rightarrow N$)**



- **Muchos a Uno ($N \rightarrow 1$)**

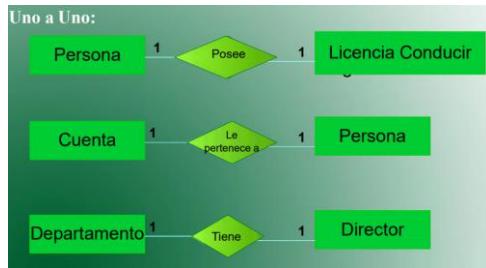


- **Uno a Muchos ($1 \rightarrow N$)**





- Uno a Uno ($1 \rightarrow 1$)



Restricción de Participación: Indica si es o no obligatorio que exista la relación para una entidad.

- Total

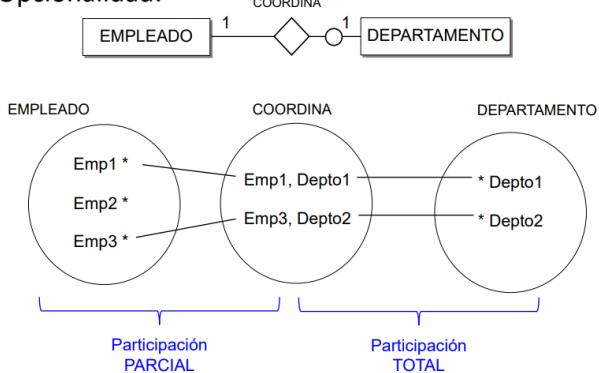


- Parcial



Se lee:
Un empleado **PUEDE** coordinar un departamento
Un departamento **DEBE** ser coordinado por un empleado

Opcionalidad:



Grado de una relación

Es la cantidad de entidades que participan de una relación.

- Unarias



- Binarias



- Ternarias



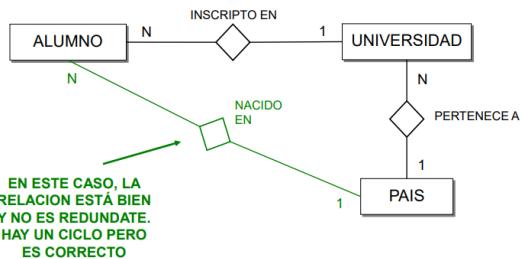
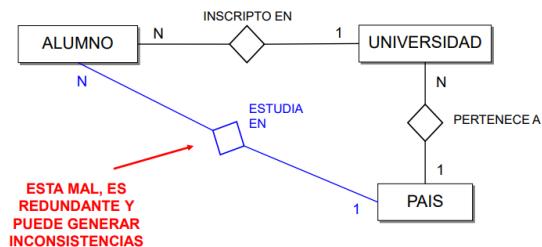
La relación ternaria se puede reemplazar por una entidad triplemente débil o doblemente débil, según la cardinalidad de la ternaria.

SIEMPRE QUE SE PUEDA RESOLVER CON RELACIONES BINARIAS, HAY QUE HACERLO DE ESA FORMA. LA TERNARIA ES EL ULTIMO RECURSO.



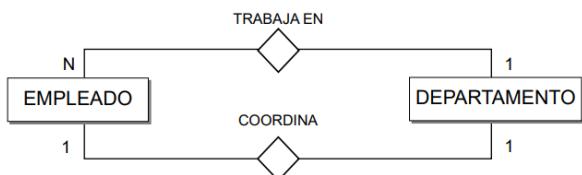
Relaciones Redundantes/Transitivas

Los ciclos en un DER están permitidos ya que no siempre indican redundancia, pero merecen que se les preste una especial atención para verificar que sean correctos. Por ejemplo:



Múltiples relaciones entre las mismas Entidades

Entre dos Entidades pueden existir varias relaciones que representen distinta información y eso es correcto. Ejemplos:



- Dominios

El dominio de un atributo es el conjunto de valores permitidos que este puede adquirir.

- Etapas en el diseño de una base de datos

Análisis de Requerimientos: en donde se relevan las necesidades de los usuarios y se interpreta qué debería realizar la base de datos.

Modelo Conceptual: es un modelo que permite la identificación de los actores de la base de datos y sus relaciones. Generalmente utilizamos el diagrama DER para su representación.

Modelo Lógico: es un modelo que permite convertir el modelo conceptual en un esquema que tenga relación con el DBMS.

Modelo Relacional. Modelo Físico: es el diseño físico de la base de datos, en cuanto a discos, índices y esquema de datos en lenguaje que la base de datos pueda interpretar.



- Es un modelo lógico que permite la representación del DER
- Un Modelo Relacional es un conjunto de relaciones
- En este modelo, su principal objeto se lo denomina relación
- Las relaciones poseen atributos, cuyos valores son atómicos
- Las filas de las relaciones se denominan tuplas y cada una de ellas se puede identificar únicamente.

El Modelo Relacional se basa en relaciones:

Primary Key (PK)

Una clave primaria es una columna (o combinación de columnas) especial de una tabla de base de datos relacional designada para identificar de forma exclusiva cada registro de la tabla.

Foreign Key (FK)

Es una clave de base de datos que se utiliza para vincular dos tablas de la base de datos. Es un campo (o colección de campos o columnas) en una tabla, que se refiere a la PRIMARY KEY en otra tabla.

Relación

Una relación es un conjunto de tuplas donde todas las filas tienen los mismos atributos, pueden identificarse únicamente y no existe un orden entre las filas.

Restricciones

- *Restricción de Dominio*
Indica que el valor del atributo A, debe ser un valor del Dominio(A).
- *Restricción de Integridad de Entidades*
La clave primaria identifica únicamente a cada fila y no puede ser nula
- *Restricción de Integridad Referencial*
Si una tupla t1 tiene relación en R1 (a1) con R2 (b1), t1 (a1) sólo puede contener valores existentes en R2 (b1).

Pasaje del Modelo DER al MR

Reglas de conversión

ENTIDAD

ENTIDAD

Toda Entidad del DER se convierte en una relación

- *Entidad Débil*
Las entidades débiles se modelan como una relación, pero la principal diferencia es que no puede identificarse por sí misma, por lo que requiere heredar la clave de la entidad fuerte.



➤ **Jerarquías**

Las subentidades de las jerarquías se modelan como otra relación; pero como no tienen clave, heredan la clave de su entidad padre.

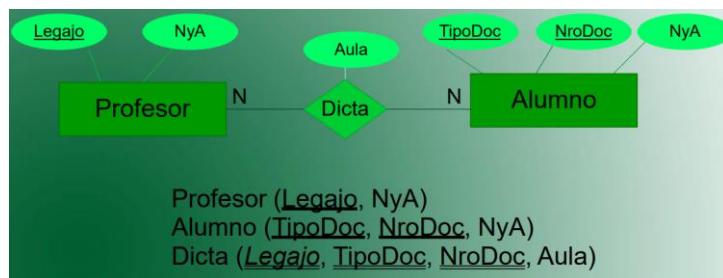


ATRIBUTOS

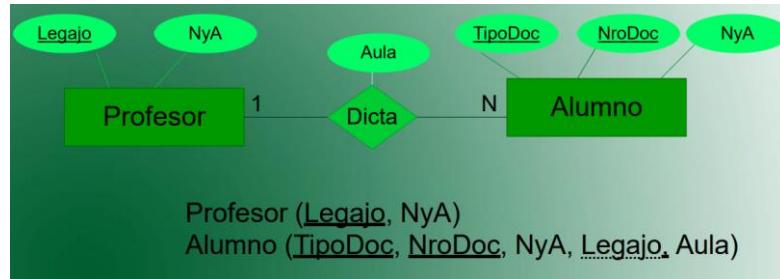
- Profesor Tel Los Atributos Multivaluados se transforman en otra Entidad.
- Los Atributos Compuestos se transforman agregando los atributos a la relación
- Los Atributos Calculados no se modelan en el MR

Si tenemos atributos en las relaciones, dependerá de la cardinalidad de la relación para saber en qué relación deberá generarse el atributo.

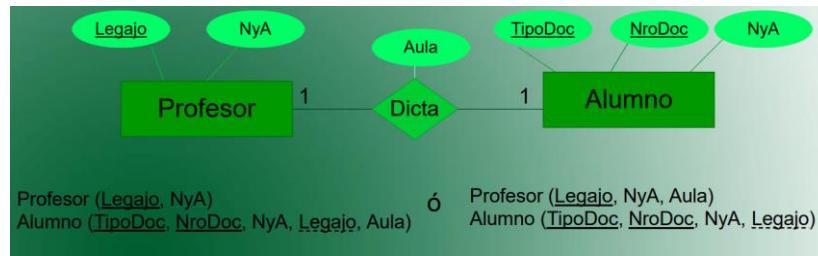
- **N-->N**: Se agregará el atributo en la nueva relación



- **1-->N ó N-->1**: Se agregarán los atributos en la relación con cardinalidad N.



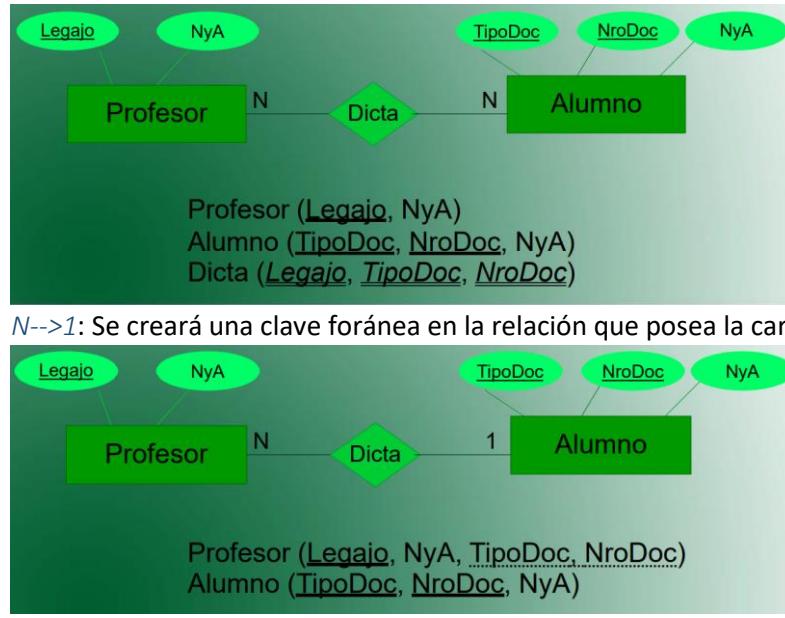
- **1-->1**: Se puede elegir en qué relación se agrega el atributo.



RELACION

- Relaciones Binarias: Se transformarán, según el tipo de cardinalidad que tenga establecida la relación.

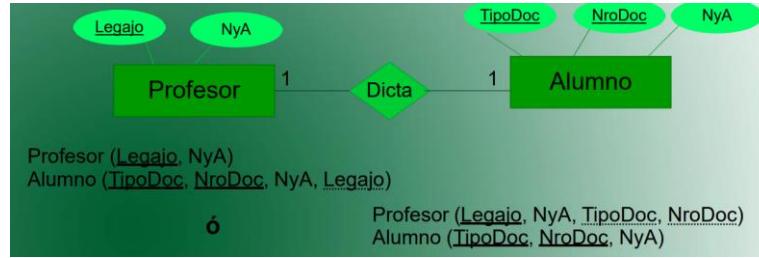
N-->N: Se creará una nueva relación



1-->N: Se creará una clave foránea en la relación que posea la cardinalidad N



1-->1: Se creará una clave foránea en cualquiera de las entidades participantes, asociadas con la otra relación

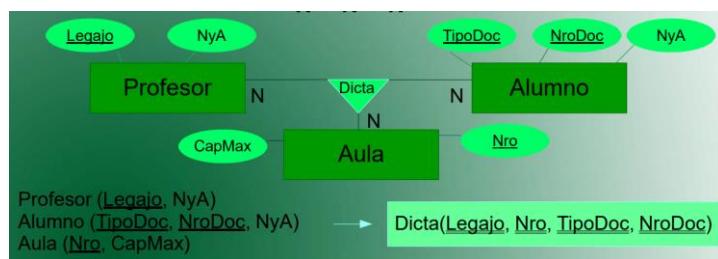


- Relaciones Unarias: Se considerará como si fueran dos relaciones binarias y se utilizará la misma estrategia vista en las relaciones binarias, sólo que existirá una única relación interviniente.

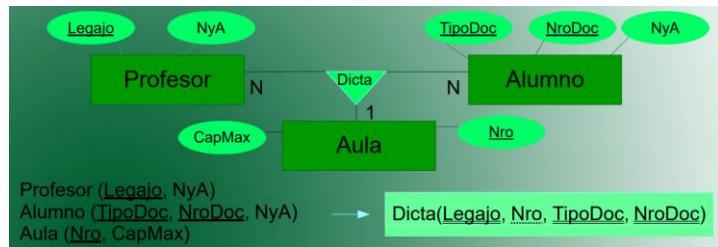


- **Relaciones Ternarias:** Las relaciones ternarias siempre generan una cuarta relación, en el modelo relacional. Lo que va a distinguir la conversión es el tipo de cardinalidad que posea la relación ternaria

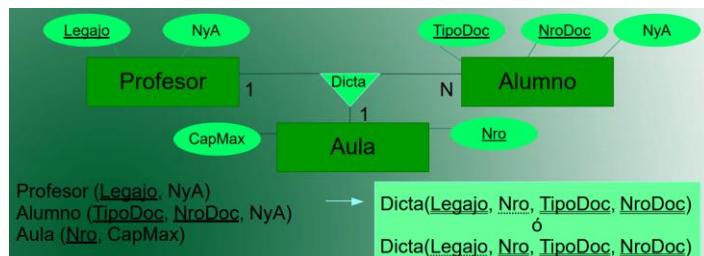
N-->N-->N



N-->N-->1: En la relación nueva generada, siempre tendrá una clave primaria compuesta, de al menos 2 atributos claves. Si es N, siempre será parte de la clave formada



N-->1-->1: La clave primaria de la nueva relación, debe estar compuesta de al menos 2 claves de las entidades relacionadas, en este caso, elegimos a la N y luego cualquiera de las relaciones con 1



1-->1-->1: Cuando la relación tiene cardinalidad de 1 en todas sus dependencias, podemos elegir cualquier clave compuesta de 2 atributos claves de las relaciones.



UNLaM

UNIDAD N°3: ALGEBRA RELACIONAL

Para lo que concierne al álgebra relacional, tomaremos a cada conjunto como una relación, compuesta por una cantidad determinada de atributos. Cada elemento del conjunto, será una tupla, fila o registro de la relación, que tendrá valores determinados en cada uno de los atributos.

Es un conjunto de operaciones que permiten manipular relaciones (futuras tablas).

Estas operaciones permiten obtener tuplas determinadas y combinar tuplas de distintas relaciones.

El resultado de aplicar estas operaciones es siempre otra relación.

Es un lenguaje de consultas puramente teórico y no se utiliza en la práctica (a diferencia del SQL).

A continuación veremos las siguientes operaciones:

- Selección
- Proyección
- Unión
- Intersección
- Diferencia
- Producto Cartesiano
- Junta Theta y Junta Natural

Clasificación

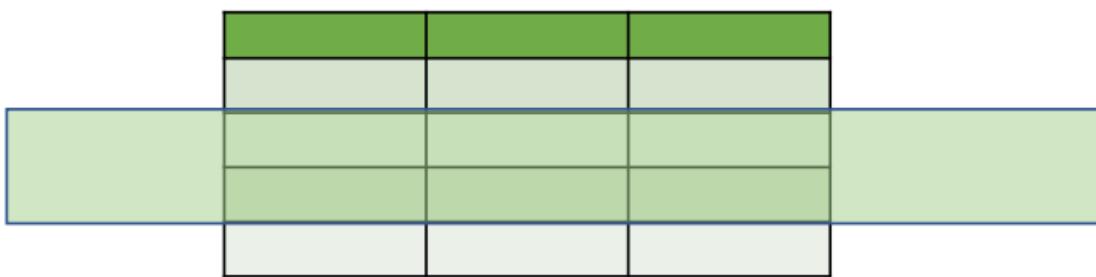
Operaciones de AR:

- **Unarias:** se aplican sobre una única relación.
- **Binarias:** se aplican sobre dos relaciones.
- **Primitivas:** se basan en la teoría de conjuntos. Por Ej: unión, intersección, diferencia y producto cartesiano
- **De Base de Datos:** selección, proyección y junta

SELECCIÓN

Esta operación permite seleccionar un subconjunto de tuplas de una relación que cumplen una determinada condición

Realiza un corte horizontal de la relación.



σ <condición> (<Nombre de la relación>)

Letra griega
Sigma

<Nombre de atributo>

<operador de
comparación>

<Valor constante /
Nombre de atributo>

>, >=, <, <=, =, <>

Múltiples condiciones se pueden combinar con AND (\wedge), con OR (v) y con NOT

- Da como resultado una relación con los mismos atributos que la relación original (el mismo Grado), pero con menor o igual cantidad de tuplas (menor o igual Cardinalidad)
 - Es una operación unaria, porque se aplica a una sola relación
 - Es conmutativa
 - En esta operación el conjunto resultante tendrá la misma cantidad de columnas que la relación original
- $$\sigma<\text{condicion1}> (\sigma<\text{condicion2}> (R)) = \sigma<\text{condicion2}> (\sigma<\text{condicion1}> (R))$$
- Cascada: Las selecciones en cascada (anidadas) se pueden expresar como una única condición combinada con AND
- $$\sigma<\text{condicion1}> (\sigma<\text{condicion2}> (R)) = \sigma<\text{condicion2}> \wedge <\text{condicion1}> (R)$$

Supongamos que tenemos la siguiente relación:

EMPLEADO (legajo, nya, ciudad, cod_dept, salario, legajo_supervisor)

Ejemplo1: Listar los empleados de la ciudad de San Justo

σ ciudad='San Justo' (Empleado)

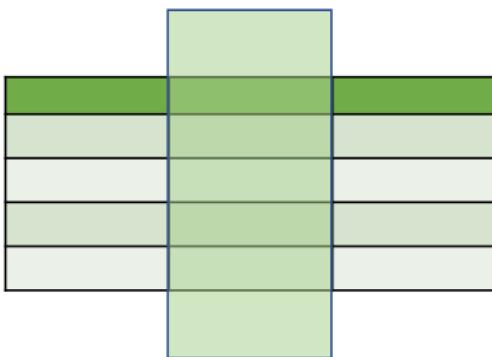
Ejemplo2: Listar los empleados que tengan salario mayor a 40000 y que además sean de Haedo o Morón

σ salario>40000 \wedge (ciudad='Haedo' v ciudad='Morón') (Empleado)

PROYECCIÓN

Esta operación selecciona ciertos atributos de una relación y descarta los otros.

Realiza un corte vertical de la relación



Π <lista de atributos> (<Nombre de la relación>)

The diagram consists of two blue arrows. The first arrow originates from the letter 'P' in the word 'Letra griega Pi' and points towards the word 'Nombres'. The second arrow originates from the letter 'i' in the word 'Letra griega Pi' and points towards the word 'separados'.

- Da como resultado una relación con un Grado igual a la cantidad de atributos de la lista y con una Cardinalidad igual o menor a la relación original, según si los valores resultantes se repiten o no (las tuplas repetidas se eliminan)
 - Es una operación unaria, porque se aplica a una sola relación
 - No es conmutativa

Supongamos que tenemos la siguiente relación:

EMPLEADO(legajo, nya, ciudad, cod depto, salario, legajo supervisor)

Ejemplo: Listar el Nombre y Apellido y el Salario de todos los empleados

π nya, salario (Empleado)

Relaciones Intermedias y Renombre de atributos

En vez de hacer operaciones de AR anidadas podemos utilizar las relaciones intermedias (es como guardar los datos que me trae una consulta/Query en un lugar/variable)

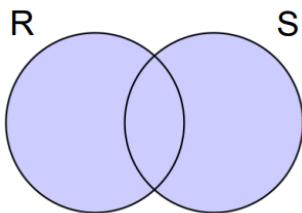
Emp40000 \leftarrow $\sigma_{\text{salario} > 40000}$ (Empleado)

π nya, salario (**Emp40000**)

Es posible Renombrar los atributos de las relaciones resultantes:

Emp (nombre completo, sueldo) $\leftarrow \pi_{\text{nya}, \text{salario}}(\text{Emp40000})$

UNION
BULLS



- Da como resultado una relación con las tuplas que están en R, en S o en ambas.
- Descarta las tuplas duplicadas (si una tupla está en ambas relaciones, solo aparece una vez en el resultado)
- El resultado queda con el nombre de los atributos de la primer relación
- Para poder hacer la Unión, las relaciones deben ser **compatibles**, es decir, deben tener **igual Grado** (misma cantidad de atributos) y además los atributos deben tener **igual Dominio** (igual tipo de atributo)
 $R(a_1, a_2, \dots, a_n) \text{ y } S(b_1, b_2, \dots, b_n)$
 $\text{Dom}(a_i) = \text{Dom}(b_i)$
- Es una operación binaria ya que se aplican a dos relaciones.
- Esta operación es conmutativa y asociativa

EMPLEADO (legajo, nya, ciudad, cod_dept, salario, legajo_supervisor)

Ejemplo: Listar los legajos de los empleados de Haedo junto con los legajos de los Supervisores de los empleados de Morón.

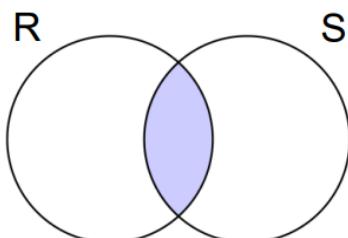
EmpHaedo $\leftarrow \pi_{\text{legajo}} (\sigma_{\text{ciudad}=\text{'Haedo'}} (\text{Empleado}))$

SupMoron $\leftarrow \pi_{\text{legajo_supervisor}} (\sigma_{\text{ciudad}=\text{'Moron'}} (\text{Empleado}))$

EmpHaedo U SupMoron

INTERSECCIÓN

$R \cap S$



- Da como resultado una relación con las tuplas que están tanto en R como en S (en ambas).
- Al igual que en la Unión, las relaciones deben ser compatibles (igual grado y dominio)
- Es una operación binaria ya que se aplican a dos relaciones.
- Esta operación es conmutativa y asociativa

EMPLEADO (legajo, nya, ciudad, cod_dept, salario, legajo_supervisor)

Ejemplo: Listar los códigos de departamento donde trabajan empleados de Haedo y Morón (de ambas ciudades).

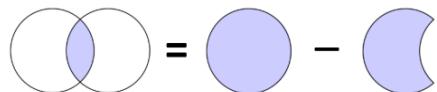
DepH $\leftarrow \pi_{cod_depto} (\sigma_{ciudad='Haedo'} (Empleado))$

DepM $\leftarrow \pi_{cod_depto} (\sigma_{ciudad='Morón'} (Empleado))$

DepH \cap DepM

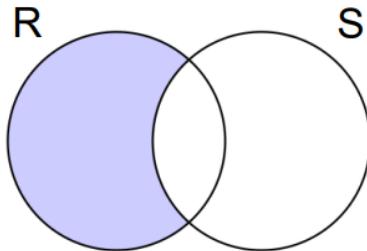
Intersección con Operadores Básicos

$$R \cap S = R - (R - S)$$



DIFERENCIA

$R - S$



- Da como resultado una relación con las tuplas que están en R pero no están en S.
- Al igual que en la Unión, las relaciones deben ser compatibles.
- La Diferencia no es comutativa ni asociativa.
- operaciones binarias ya que se aplican a dos relaciones.

EMPLEADO (legajo, nya, ciudad, cod_depto, salario, legajo_supervisor)

$\pi_{cod_depto} (Empleado) - DepNo$

PRODUCTO CARTESIANO

$R \times S$

R	A	B
1	100	
2	120	
3	100	

S	C	D	E
v	4	23	
f	5	18	

RXS	A	B	C	D	E
1	100	v	4	23	
1	100	f	5	18	
2	120	v	4	23	
2	120	f	5	18	
3	100	v	4	23	
3	100	f	5	18	

Combina las tuplas de ambas relaciones y como resultado se genera una relación con todas las combinaciones de tuplas (todos contra todos).

- El resultado tendrá TODOS los atributos, por más que tengan el mismo nombre (de ser necesario se le agrega una comilla o apóstrofo).
- Grado de RXS = Grado de R + Grado de S
- Cardinalidad de RXS = Cardinalidad de R * Cardinalidad de S
- No es necesario que las relaciones sean compatibles.
- Es una operación binaria.
- Esta operación es conmutativa

[JUNTAS](#)

La Junta permite combinar tuplas relacionadas de dos relaciones en una única tupla.

➤ JUNTA THETA

$$\begin{array}{c} R \mid X \mid S \\ \theta \end{array}$$

Da como resultado aquellas combinaciones que cumplen la condición de junta.

- Las tuplas cuyos valores sean nulos en los atributos de junta, no cumplirán la condición de junta y por lo tanto no aparecerán en el resultado
- Grado de la Junta Theta = Grado de R + Grado de S

Ejemplo:

Emp (id, nombre, cod_dep)

Dep (cod, descripcion)

Hacemos la Junta Theta de ambas relaciones a través del código de departamento:

Emp |X| Dep

cod_dep=cod

Ejemplo:

Emp	id	nombre	cod_dep	Dep	cod	descripcion
	1	Jorge	10		10	Ventas
	2	Pablo	11		11	Sistemas
	3	Juan			12	Compras
	4	Laura	10			Desconocido

Emp X Dep cod_dep=cod	id	nombre	cod_dep	cod	descripcion
	1	Jorge	10	10	Ventas
	2	Pablo	11	11	Sistemas
	4	Laura	10	10	Ventas



Junta Theta con Operadores Básicos

$$R |_{\theta} X | S = \sigma_{\theta}(R \times S)$$

Producto Cartesiano seguido de una Selección

➤ JUNTA NATURAL

$$R | X | S$$

Es una Junta automática, en la que se igualan aquellos atributos que tienen en común (atributos con igual nombre).

- Elimina los atributos repetidos y solo deja uno de cada uno de ellos.
- La condición es siempre de igualdad (no hay $>$, $<$, etc.)
- Si las relaciones no tienen atributos en común, se comporta como un Producto Cartesiano.
- Si hay atributos en común pero no tienen valores iguales, entonces devuelve una relación vacía

Ejemplo:

R	A	B		S	B	C	D
	1	100			50	4	6
	2	120			100	5	8

Ejemplo:

R	A	B		S	B	C	D
	1	100			50	4	6
	2	120			100	5	8

R X S	A	B	C	D
	1	100	5	8
	1	100	9	3

Junta Natural con Operadores Básicos

$$R | X | S = \pi_{\text{<atributos distintos>}} (\sigma_{\text{<condición con atributos iguales>}}(R \times S))$$

Producto Cartesiano seguido de una Selección, seguido de una Proyección

Operaciones Básicas y Derivadas

- **Básicas:** es un conjunto completo porque en base a estas operaciones se pueden obtener las otras. Selección, Proyección, Unión, Diferencia y Producto Cartesiano.
- **Derivadas:** Intersección, Junta Theta y Junta Natural

EJERCICIOS DE PARCIAL

Estos ejercicios los toman en los parciales

Hallar el menor / mayor

EJERCICIO 3

Almacén (Nro, Responsable)
 Artículo (CodArt, descripción, Precio)
 Material (CodMat, Descripción)
 Proveedor (CodProv, Nombre, Domicilio, Ciudad)
 Tiene (Nro, CodArt)
 CompuestoPor (CodArt, CodMat)
 ProvistoPor (CodMat, CodProv)

o. Hallar el o los códigos de los artículos de mayor precio.

Relación
“Artículo”

Codigo	Precio
a	10
b	20
c	30
d	40

- 1) Buscaremos obtener los artículos que tiene un precio “NO” máximo o dicho de otra manera... Un Precio menor a otro precio existente

Articulo X Articulo

Codigo	Precio	Codigo'	Precio'
a	10	a	10
a	10	b	20
a	10	c	30
a	10	d	40
b	20	a	10
b	20	b	20
b	20	c	30
b	20	d	40
c	30	a	10
c	30	b	20
c	30	c	30
c	30	d	40
d	40	a	10
d	40	b	20
d	40	c	30
d	40	d	40

Codigo	Precio	Codigo'	Precio'
a	10	a	10
a	10	b	20
a	10	c	30
a	10	d	40
b	20	a	10
b	20	b	20
b	20	c	30
b	20	d	40
c	30	a	10
c	30	b	20
c	30	c	30
c	30	d	40
d	40	a	10
d	40	b	20
d	40	c	30
d	40	d	40

- 2) $\sigma((2)<(4))$ (Articulo X Articulo)



Codigo	Precio	Codigo'	Precio'
a	10	b	20
a	10	c	30
a	10	d	40
b	20	c	30
b	20	d	40
c	30	d	40

3) $\pi(\text{CodArt}) (\sigma((2)<(4)) (\text{Articulo} \times \text{Articulo}))$

Codigo
a
b
c

Artículos cuyo precio es menor a algún otro

4) $\pi(\text{CodArt})(\text{Articulo}) - (\pi(\text{CodArt}) (\sigma((2)<(4)) (\text{Articulo} \times \text{Articulo})))$

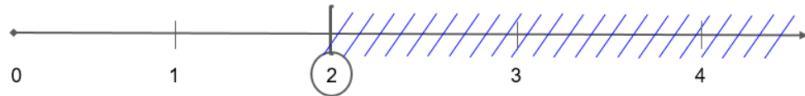
Codigo
d

Artículos cuyo precio NUNCA es menor a algún otro

Contar

Hallar el o los materiales provistos por 2 o más Proveedores

ProvistoPor (**CodMat, CodProv**)



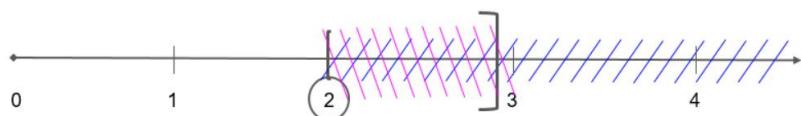
- $\pi(\text{CodMat}) (\sigma(1=3 \wedge 2>4) (\text{ProvistoPor} \times \text{ProvistoPor}))$

Hallar el o los materiales provistos por más de 2 Proveedores.



- $\text{RTA} \leftarrow \pi(\text{CodArt}) (\sigma(1=3 \wedge 1=5 \wedge 2>4 \wedge 2>6 \wedge 4>6) ((\text{ProvistoPor} \times \text{ProvistoPor}) \times \text{ProvistoPor}))$

Hallar el o los materiales provistos por exactamente 2 Proveedores.



- $\text{ProvX2} \leftarrow \pi(\text{CodArt}) (\sigma(1=3 \wedge 2 <> 4) (\text{ProvistoPor} X \text{ProvistoPor}))$
 $\text{ProvX3} \leftarrow \pi(\text{CodArt}) (\sigma(1=3 \wedge 1=5 \wedge 2 <> 4 \wedge 2 <> 6 \wedge 4 <> 6) (\text{ProvistoPor} X \text{ProvistoPor} X \text{ProvistoPor}))$
Rta $\leftarrow \text{ProvX2} - \text{ProvX3}$



UNIDAD N°4: NORMALIZACION

- [Introducción a la Normalización](#)

La Normalización es un proceso mediante el cual se puede depurar un diseño de base de datos.

Permite eliminar ciertos defectos y características indeseables de los esquemas.

Para entender bien cuáles son esos defectos que pueden tener los esquemas de datos, vamos a ver a continuación algunos ejemplos de problemas que se pueden presentar.

Problemas de un esquema **NO** Normalizado

Existen muchos problemas o inconvenientes, nosotros solo vamos a ver algunos de ellos:

- REDUNDANCIA
- ANOMALIAS DE ACTUALIZACIÓN
- ANOMALIAS DE INSERCIÓN
- ANOMALIAS DE ELIMINACIÓN
- PERDIDA DE INFORMACIÓN

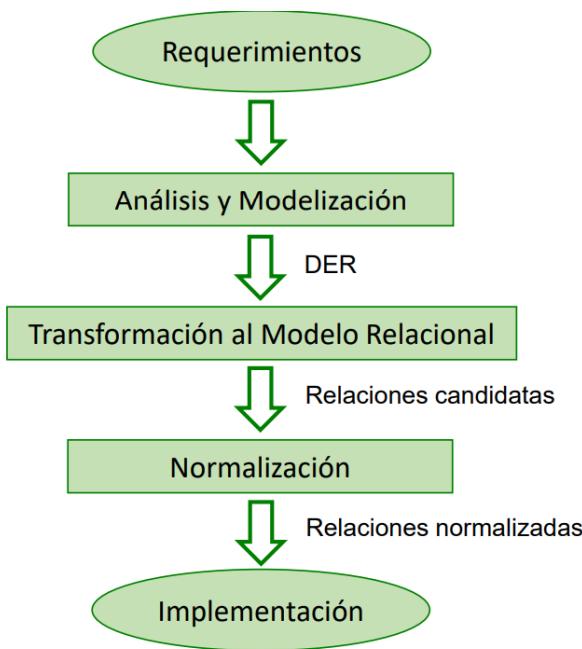
REDUNDANCIA: Esto ocurre porque estamos mezclando información de hechos independientes

ANOMALIAS DE ACTUALIZACIÓN: Si hay que modificar el nombre de un alumno, tendremos que actualizar varias filas (en todas las que figure ese alumno). Mientras que si hubiese estado bien normalizado, el nombre debería estar solo en una fila y solo sería necesario modificar un único valor.

ANOMALIAS DE INSERCIÓN: Si tenemos que dar de alta un nuevo alumno, que aún no cursó ni rindió ninguna materia, vamos a tener que dejar con valor NULO los otros atributos. Ídem si tenemos que dar de alta una materia sin alumnos.

ANOMALIAS DE ELIMINACIÓN: Si tenemos que eliminar un Alumno y no queda otra fila con esa Materia, se pierde. O bien, si se decide eliminar a todos los exámenes desaprobados (que tienen nota menor a 4) se eliminaran muchas filas y tal vez se pierdan alumnos que solo tienen exámenes desaprobados.

PÉRDIDA DE INFORMACIÓN: Este problema aparece en el caso de una descomposición errónea de un esquema en varios subesquemas. Más adelante veremos detalladamente en que consiste la Pérdida de Información.



DEPENDENCIA FUNCIONAL

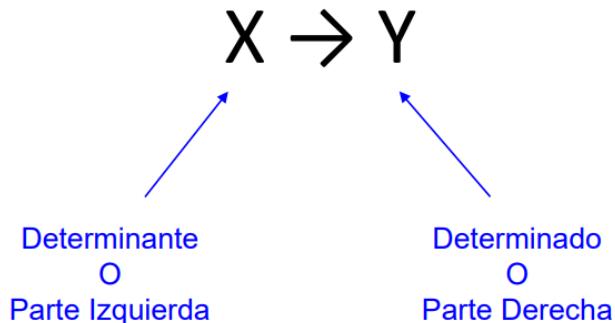
Una dependencia funcional es una restricción entre dos conjuntos de atributos de una relación. Sea R un esquema de relación, sea r una instancia cualquiera de R y sean X e Y dos subconjuntos de R . La dependencia funcional $X \rightarrow Y$ (léase X determina a Y) se cumple si y sólo si para cualesquier dos tuplas t_1 y t_2 de r , tales que:

$$t_1[X] = t_2[X], \text{ entonces necesariamente } t_1[Y] = t_2[Y]$$

Es decir, si dos tuplas cualesquier tienen igual valor en X , deben tener igual valor en Y , para que se cumpla la dependencia funcional.

Esto significa que los valores componentes de Y de una tupla de r dependen de los valores del componente X , o están determinados por ellos; o bien, que los valores del componente X de una tupla determinan de manera única los valores del componente Y .

X determina a Y



Ejemplo:

EXAMEN (DNI_Alumno, Nombre, Apellido, Cod_Materia,
Nombre_Materia, Fecha_Examen, Nota)

Se cumplen las siguientes dependencias funcionales:

$$F = \{ \text{DNI_Alumno} \rightarrow \text{Nombre, Apellido} ; \\ \text{Cod_Materia} \rightarrow \text{Nombre_Materia} ; \\ \text{DNI_Alumno, Cod_Materia, Fecha_Examen} \rightarrow \text{Nota} \}$$

DEPENDENCIA FUNCIONAL TRIVIAL

Decimos que una dependencia funcional es Trivial cuando es obvia.

Por ejemplo $X \rightarrow X$

Todo conjunto de atributos se determina a sí mismo o a un conjunto de atributos menor (que este contenido en el primero).

$XY \rightarrow X$

CUESTIONES A TENER EN CUENTA:

Las dependencias funcionales vinculan dos conjuntos de atributos.

Según la teoría de conjuntos:

- El orden de los elementos de un conjunto no importa.

$XY \rightarrow Z$ es exactamente igual que $YX \rightarrow Z$

- Los conjuntos no pueden tener elementos repetidos.

No podemos tener $XX \rightarrow Y$

De las dos X automáticamente se suprime una y queda $X \rightarrow Y$

CLAVE Y SUPERCLAVE

Dado un esquema de relación $R(A_1, A_2, \dots, A_n)$ y un conjunto de dependencias funcionales F asociado, se dice que el conjunto de atributos X perteneciente a R es una **CLAVE** de R si se cumplen las siguientes dos condiciones:

1) X determina a todos los atributos de R , $X \rightarrow A_1, A_2, \dots, A_n$

2) No existe ningún Z (subconjunto de X) que determine a todos los atributos de R (condición de minimalidad).

Por otra parte, decimos que el conjunto de atributos Y es **SUPERCLAVE** de R si cumple la condición 1.

Se cumple que toda clave es superclave, pero no a la inversa.



CLAVE Y SUPERCLAVE

Ejemplo:

EMPLEADO (Legajo, Nombre, DNI)

Clave Primaria
Primary Key (PK)

Claves Candidatas = { Legajo ;
DNI }

El Diseñador de la Base de
Datos tiene que elegir a una de
las claves candidatas como
Clave Primaria

Superclaves = { Legajo, Nombre ;
DNI, Nombre ;
Legajo, DNI ;
Legajo, Nombre, DNI ;
Legajo ;
DNI }

AXIOMAS DE ARMSTRONG

Reflexividad

Si $Y \subseteq X \Rightarrow$ se cumple $X \rightarrow Y$

Aumento

Dada $X \rightarrow Y$ se puede inferir $XZ \rightarrow YZ$

Transitividad

Dadas $X \rightarrow Y$ y $Y \rightarrow Z$ se puede inferir $X \rightarrow Z$

REGLAS ADICIONALES/DERIVADAS

Descomposición

Dada $X \rightarrow YZ$ se pueden inferir $X \rightarrow Y$ y $X \rightarrow Z$

Unión

Dadas $X \rightarrow Y$ y $X \rightarrow Z$ se puede inferir $X \rightarrow YZ$

Pseudotransitividad

Dadas $X \rightarrow Y$ y $WY \rightarrow Z$ se puede inferir $WX \rightarrow Z$

Las Reglas Adicionales se pueden demostrar a partir de los

Axiomas de Armstrong.

Demostración de la Pseudotransitividad:

- 1) $X \rightarrow Y$ Dada
- 2) $WX \rightarrow WY$ Aumento con W en 1
- 3) $WY \rightarrow Z$ Dada
- 4) $WX \rightarrow Z$ Transitividad entre 3 y 2



CLAUSURA DE UN CONJUNTO DE DEPENDENCIAS (F+)

Dado un conjunto de dependencias llamado F, denominamos clausura de F (F+) al conjunto de todas las dependencias posibles que se pueden inferir de F.

Ejemplo: R(A, B, C) F={ A→B, BC→A }

$$\begin{aligned} F^+ = & \{ \text{Todas las dependencias Triviales} \\ & + \\ & \text{Las dependencias de } F \\ & + \\ & \text{Dependencias inferidas} \end{aligned}$$

Ejemplo: R(A, B, C) F={ A→B, BC→A }

$$\begin{aligned} F^+ = & \{ \text{A} \rightarrow A, & \text{Azul: Triviales} \\ & B \rightarrow B, & \text{Rojo: Provenientes de } F \\ & C \rightarrow C, & \text{Verde: Inferidas} \\ & AB \rightarrow AB, AB \rightarrow A, AB \rightarrow B, \\ & AC \rightarrow AC, AC \rightarrow A, AC \rightarrow C, \\ & BC \rightarrow BC, BC \rightarrow B, BC \rightarrow C, \\ & ABC \rightarrow ABC, ABC \rightarrow AB, ABC \rightarrow AC, ABC \rightarrow BC, ABC \rightarrow A, ABC \rightarrow B, ABC \rightarrow C, \\ & A \rightarrow B, A \rightarrow AB, AC \rightarrow BC, AC \rightarrow ABC, AC \rightarrow AB, AC \rightarrow B, \\ & BC \rightarrow A, BC \rightarrow AB, BC \rightarrow AC, BC \rightarrow ABC \end{aligned}$$

}

CLAUSURA DE UN CONJUNTO DE ATRIBUTOS (X+)

Sea X un conjunto de atributos del esquema de relación R, y F un conjunto de dependencias funcionales que se cumplen en R, llamamos Clausura de X en F (X+F) al conjunto de **todos los atributos determinados funcionalmente por x**

Ejemplo:

R(A, B, C) F = { A→B, BC→A }

$$\begin{aligned} A^+_F &= AB \\ B^+_F &= B \\ C^+_F &= C \\ AB^+_F &= AB \\ AC^+_F &= ABC \\ BC^+_F &= ABC \\ ABC^+_F &= ABC \end{aligned}$$

Con esto podemos calcular F+ fácilmente, (haciendo todas las combinaciones)



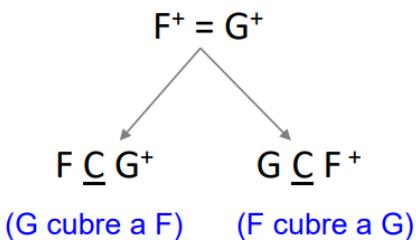
EQUIVALENCIA ENTRE DOS CONJUNTOS DE DEPENDENCIAS FUNCIONALES

Dos conjuntos de dependencias funcionales F y G son equivalentes cuando sus clausuras son iguales:

$$F \equiv G \iff F^+ = G^+$$

Calcular F^+ y G^+ puede ser muy largo y tedioso

Para probar que dos conjuntos F y G son equivalentes, sin necesidad de calcular F^+ ni G^+ , se debería probar que toda dependencia de F puede inferirse en G ($F \subseteq G^+$), y recíprocamente, toda dependencia de G se puede inferir en F ($G \subseteq F^+$).



Podemos determinar si F cubre a G calculando $X+F$ para cada dependencia $X \rightarrow Y$ de G , y comprobando que $Y \subseteq X+F$

UNIDAD N°5: SQL

- Introducción al SQL

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 incorpora un determinado standard.

Clasificación

DDL (Data Definition Language)

Son todas aquellas instrucciones que permiten la definición de los distintos objetos de la base de datos.

Permiten modificar la estructura de las tablas.

NO se pueden deshacer (NO admiten Roll Back)

- TRUNCATE
- DROP
- ALTER
- CREATE

DML (Data Manipulation Language)

Son aquellas instrucciones que permite la manipulación de los datos almacenados en la base de datos.

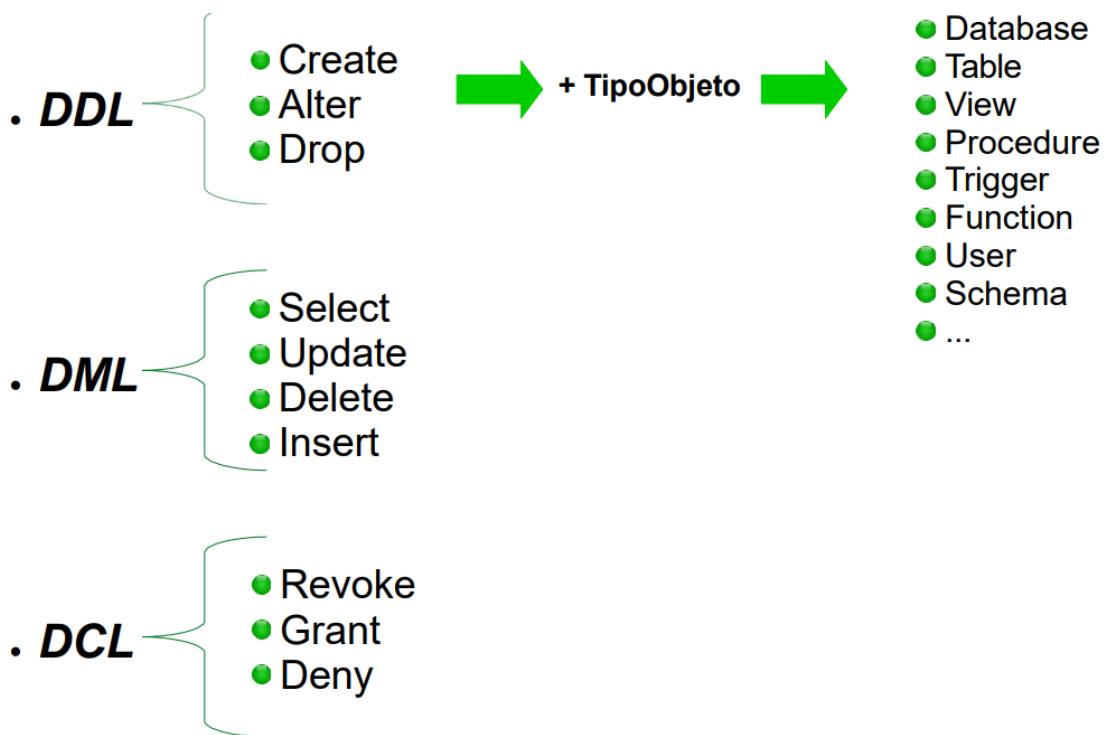
Permiten modificar los datos de las tablas.

Se pueden deshacer (admiten Roll Back)

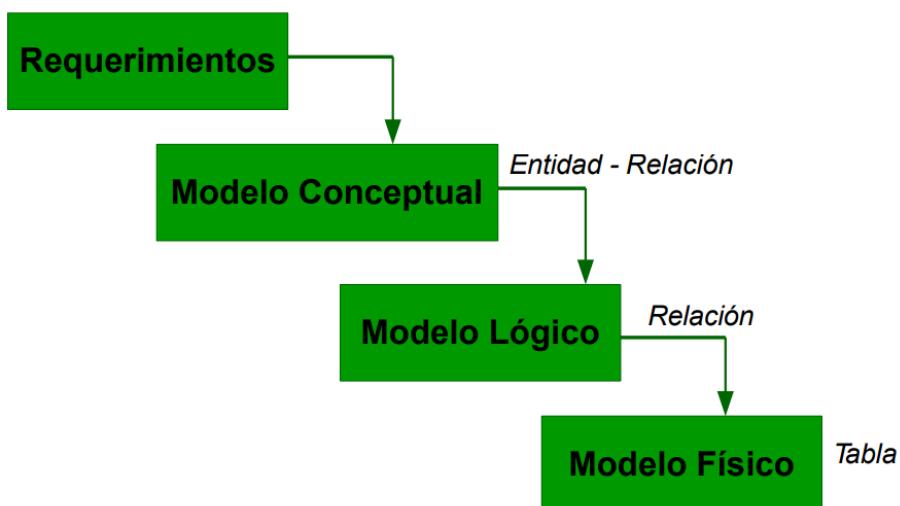
- INSERT
- UPDATE
- DELETE

DCL (Data Control Language)

Son las aquellas instrucciones que permite el control de acceso a los datos.



Etapas del diseño



DDL

CREATE TABLE

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])]

)

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)
```

Tipos de Datos

Clasificación	Tipo de datos	Tamaños	
Numérico Entero	bit	0-1	(1 byte)
	Tinyint	0-255	(1 byte)
	Smallint	-2^15 a 2^15	(2 bytes)
	Int	-2^31 a 2^31	(4 bytes)
	bigint	-2^63 a 2^63	(8 bytes)
Numérico con Decimales	Numeric -Decimal	-2^38 a 2^38 (5-17bytes)	
	Float – Real (n)	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	3 bytes	
	Time	4 bytes	
	Smalldatetime	4 bytes	
	Datetime	8 bytes	
	Timestamp	8-16 bytes	
	/rowversion		
Otros	Binary XML	2^31 hasta 2 gb	

[DROP TABLE](#)

[DROP TABLE NombreTabla](#)

[ALTER TABLE](#)

[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  
ALTER TABLE Empleado ADD Mail varchar(200)  
ALTER TABLE Empleado DROP COLUMN Mail  
ALTER TABLE Empleado ADD CONSTRAINT PKEmppleado PRIMARY KEY (NroDoc, TipoDoc)  
ALTER TABLE Empleado DROP CONSTRAINT FKCargo
```

DML

Select

```
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
```

Condiciones



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

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)
```

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
```



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

```
SELECT e.* FROM EMPLEADO e
```

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>)

Any / Some: uno de los valores de la tabla tiene que cumplir con esa condición para que dé como verdadero, utiliza un OR.

All: todos los valores de la tabla tienen que cumplir con esa condición para que dé como verdadero, utiliza un AND.

```
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)
```

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>)

```

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)

```

[Union. Union all](#)

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 legajo,fechaingreso from empleado1
UNION
SELECT leg, fing FROM empleado2

```

[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 legajo,fechaingreso from empleado1
INTERSECT
SELECT leg, fing FROM empleado2

```

[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 legajo,fechaingreso from empleado1
EXCEPT
SELECT leg, fing FROM empleado2

```

[Join](#)

➤ [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

➤ 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

➤ 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 primera 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

➤ 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

➤ Full Join

Este tipo de junta, solo devuelve una tupla cuando encuentra en la segunda tabla (right table) del Join o en la primera 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
```

Resulting Table:

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

Update

Permite modificar los valores de una o varias columnas de una o varias filas de una tabla.

UPDATE nombre_tabla

SET columna1 = <nuevo valor>,

columna2 = <nuevo valor>,

....

columnaN = <nuevo valor>

[WHERE condición]

Ejemplo:

UPDATE Empleado

SET telefono = '555-1234', salario = 50000

WHERE legajo = 23

Delete

Permite eliminar una o varias filas en una tabla.

DELETE [FROM] nombre_tabla

[WHERE condición]

Ejemplo:

DELETE Empleado

WHERE cod_depto IN (5,4)

Puede dar error si se intenta
eliminar una fila referenciada por
una Foreign Key de otra tabla.

Insert

Permite insertar una o varias filas en una tabla.

Se puede utilizar de dos formas:

Forma 1 (inserta de a una fila):

INSERT [INTO] nombre_tabla [(lista de columnas)]

VALUES (lista de valores)

Si se omite, se supone que son todas las columnas de la tabla

Ejemplo:

INSERT INTO Empleado (legajo, nombre, apellido)
VALUES (10,'Jorge', 'Varela')

Forma 2 (inserta multiples filas):

Inserta en una tabla el resultado de una consulta.

INSERT [INTO] nombre_tabla [(lista de columnas)]

SELECT lista de columnas

FROM nombre_tabla

Ejemplo:

INSERT INTO Empleado_backup
SELECT *
FROM Empleado

TRUNCATE

Permite eliminar TODAS las filas de una tabla.

TRUNCATE TABLE nombre_table

Ejemplo:

TRUNCATE TABLE Empleado

Es similar al DELETE, ya que ambas sentencias eliminan filas, pero tiene 3 diferencias:

1. El TRUNCATE elimina TODAS las filas de la tabla, no puede eliminar solo algunas
2. No tiene posibilidad de deshacerse, ya que no permite Roll Back. Igualmente esto está cambiando y algunas versiones nuevas de algunos motores están comenzando a permitirlo.
3. Libera el espacio físico que ocupan las filas. El Delete hace un borrado lógico y no libera el espacio. Si vemos cuanto espacio ocupa una tabla, luego hacemos un DELETE y eliminamos la mitad de sus filas y por ultimo volvemos a ver cuanto espacio ocupa la



tabla, veremos que sigue ocupando lo mismo. Ya que no libera el espacio de las filas eliminadas. El TRUNCATE si lo hace.

Funciones de agregación

Se aplican sobre un conjunto de filas y devuelven un único valor para todas ellas. Hay muchas, las que más se utilizan son:

- **SUM()**

Calcula la suma de valores de una columna. Ejemplo:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_dept)

DEPARTAMENTO (cod_dept, descripcion)

Suma de salarios de todos los empleados

```
SELECT SUM(salario)
```

```
FROM Empleado
```

- **MAX()**

Devuelve el mayor valor de una columna

- **MIN()**

Devuelve el menor valor de una columna

Ejemplo:

```
EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_dept)  
DEPARTAMENTO (cod_dept, descripcion)
```

Salario Mínimo y Máximo de los empleados de categoría 3

```
SELECT MIN(salario), MAX(salario)
```

```
FROM Empleado
```

```
WHERE categoria = 3
```

- **AVG()**

- **COUNT()**

COUNT()*

Cuenta la cantidad de filas



Ejemplo:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_dept)
DEPARTAMENTO (cod_dept, descripcion)

Cantidad de Departamentos

```
SELECT COUNT(*)  
FROM Departamento
```

Ejemplo2:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_dept)
DEPARTAMENTO (cod_dept, descripcion)

Cantidad total de Empleados y cuantos de ellos tienen teléfono

```
SELECT COUNT(*) cant_empleados,  
       COUNT(tel) cant_con_tel  
  FROM Empleado
```

COUNT(nombre columna)

Cuenta la cantidad de filas que no tienen valores nulos en esa columna.

➤ GROUP BY

Permite dividir el resultado de una consulta en grupos, según el valor de uno o más atributos.

Ejemplo:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_dept)
DEPARTAMENTO (cod_dept, descripcion)

Contar la cantidad de empleados de cada categoría

```
SELECT categoria, COUNT(*)  
  FROM Empleado  
GROUP BY categoria
```

GROUP BY

```
SELECT categoria, COUNT(*)  
  FROM Empleado  
GROUP BY categoria
```

Las columnas normales (no agrupadas)
siempre DEBEN estar en el GROUP BY



GROUP BY

```
SELECT COUNT(*)  
FROM Empleado  
GROUP BY categoria
```

Pero en el GROUP BY es posible colocar columnas que no estén en el SELECT

➤ HAVING

Permite especificar condiciones que se aplicarán luego del GROUP BY.

Es como el WHERE pero para colocar condiciones luego de haber agrupado.

Las condiciones del WHERE se aplican antes de hacer el agrupamiento.

Ejemplo:

Listar los Departamentos que tengan 5 o más empleados de categoría 2.

```
SELECT cod_depto, count(*) cant  
FROM Empleado  
WHERE categoria = 2  
GROUP BY cod_depto  
HAVING count(*) >=5
```

Esta condición se resuelve **antes** del GROUP BY

Esta condición se resuelve **después** del GROUP BY

NO se puede usar el Alias de la columna
HAVING cant >= 5

➤ ORDER BY

Permite ordenar el resultado de una consulta por una o más columnas.

Se debe colocar al final de la consulta.

```
SELECT ....  
FROM ....  
.....
```

Por defecto ordena en forma Ascendente

ORDER BY columna1, columna2 **DESC**,, columnaN **ASC**

Se puede ordenar por una columna que no se encuentre en el SELECT (siempre y cuando la consulta no esté agrupada)

R (a, b, c)

```
SELECT a, b  
FROM R  
ORDER BY c
```

Se puede usar el alias de una columna en el ORDER BY

R (a, b, c)

```
SELECT a, b as e  
FROM R  
ORDER BY e
```

Se puede indicar el número de columna en el ORDER BY

R (a, b, c)

```
SELECT a, c  
FROM R  
ORDER BY 2, 1
```

Se puede ordenar por una columna derivada (proveniente de un cálculo)

R (a, b, c)

```
SELECT a, b-c  
FROM R  
ORDER BY b-c
```

Vistas

Las vistas son Consultas SQL almacenadas en la base de datos con un nombre.

A diferencia de las Tablas, las vistas no contienen información, sino que simplemente devuelven el resultado de la consulta la cual se ejecuta cada vez que la vista es invocada.

Las vistas se utilizan principalmente para:

1. Almacenar consultas que utilizamos con frecuencia
2. Para restringir permisos sobre ciertos datos (por ejemplo: tabla de empleados sin la columna salario)



Creación de una vista

Ejemplo:

```
EMPLEADO (legajo, nom, ape, fecha_ingreso, salario, cod_depto)
DEPARTAMENTO (cod_depto, descripcion, legajo Gerente)
```

```
CREATE VIEW Depto AS
SELECT descripción, count(*) as cant_empleados, g.ape as gerente
FROM Empleado e, Departamento d, Empleado g
WHERE e.cod_depto=d.cod_depto
AND d.legajo_gerente=g.legajo
GROUP BY descripción, g.ape
```

Luego podemos consultar la vista como si fuese una tabla:

```
SELECT *
FROM Depto
WHERE cant_empleados > 10
```

Eliminación de una vista

```
DROP VIEW nombre_vista
```

Ejemplo:

```
DROP VIEW Depto
```

Si hay otras vistas que la usan, deja eliminarla igual, pero luego las otras vistas darán error cuando se quieran utilizar.

Actualización de los datos de una vista

Si un usuario quiere modificar, insertar o eliminar un dato de una tabla, lo mejor es que lo haga sobre la misma tabla.

Sin embargo, en algunos casos es posible hacerlo sobre la vista y el cambio se aplica automáticamente sobre la tabla a la que apuntan.

No todas las vistas son actualizables, es decir, permiten que se modifiquen los datos. Eso depende de cada motor de base de datos y versión del mismo.

En términos generales, podemos decir que si la vista es sobre una sola tabla y contiene su clave, casi seguro que permitirá que se modifiquen los datos.

Por el contrario, si la vista es una consulta agrupada, entonces casi seguro que no permitirá modificar sus datos.



UNIDAD N°6: SQL AVANZADO

Stored Procedure

Es un conjunto de sentencias SQL que pueden ejecutarse, con tan solo invocar su nombre. Los Stored procedures son similares a los procedimientos que se generan en otros lenguajes de programación. En ellos se podrá:

- Incluir 1 ó n sentencias SQL, ya sea de DML o DDL.
- Aceptar parámetros de entrada y podrá emitir parámetros de salida.
- Se podrá llamar a otro procedure.
- Podrá devolver el estado de ejecución del procedure en su nombre.
- Utilizar estrategias de programación, tales como variables, ciclos, condicionales.

Stored Procedure: Crear

Ejemplo 1:

```
CREATE PROCEDURE p_borrarclientes
AS
    DELETE FROM CLIENTE
```

```
CREATE PROCEDURE p_borrarclientesID (@IDDESDE int, @IDHASTA int)
AS
BEGIN
    DELETE FROM CLIENTE where idcliente between @IDDESDE and @IDHASTA
    INSERT INTO LOG VALUES (GETDATE(), 'BORRA CLIENTES')
END
```

Ejemplo 2:

```
CREATE PROCEDURE p_borrarclientesID (@IDDESDE int, @IDHASTA int)
AS
BEGIN
    DELETE FROM VENTA where idcliente between @IDDESDE and @IDHASTA
    DELETE FROM CLIENTE where idcliente between @IDDESDE and @IDHASTA
END
```

```
CREATE PROCEDURE p_listarclientesID (@IDDESDE int, @IDHASTA int)
AS
BEGIN
    SELECT idcliente, nombre + ' ' + apellido, fechaalta, cuit
    FROM CLIENTE
    where idcliente between @IDDESDE and @IDHASTA
    Order by idcliente
END
```

Stored Procedure: Borrar

Ejemplo 1:



```
DROP PROCEDURE p_borrarclientes  
  
DROP PROCEDURE IF EXISTS p_borrarclientesID
```

Stored Procedure: Cambiar

```
ALTER PROCEDURE p_borrarclientes  
AS  
    DELETE FROM CLIENTE  
  
ALTER PROCEDURE p_borrarclientesID (@IDDESDE int, @IDHASTA int)  
AS  
BEGIN  
    DELETE FROM CLIENTE where idcliente between @IDDESDE and @IDHASTA  
END
```

Stored Procedure: Ejecutar

```
EXEC PROCEDURE p_borrarclientes
```

```
EXECUTE PROCEDURE p_borrarclientesID @pcliid1,@pcliid2
```

Stored Procedure: Variables

Declaración de variables:

```
DECLARE @vapellido varchar(100)  
DECLARE @id int, @vnombre varchar(100)
```

Asignación de valores a las variables:

```
SET @id = 1  
SET @vapellido='Perez'
```

Stored Procedure: While

```
DECLARE @cant int  
SET @cant = (select count(*) from producto)  
  
While @cant>0  
Begin  
    UPDATE Producto set precio=precio*1.10 where id=@cant  
    Set @cant=@cant-1  
  
End
```

Stored Procedure: if

```

DECLARE @sueldo numeric(10,2)
SET @Sueldo = (Select sueldo from empleado where legajo=@legajo)

If @Sueldo > 20000
    UPDATE empleado SET sueldo=sueldo*1.20 where legajo=@legajo
Else
    UPDATE empleado SET sueldo=sueldo*1.30 where legajo=@legajo

```

Ejemplos de Stored procedure

```

CREATE PROCEDURE p_nuevocliente (@razonsocial varchar(100), @cuit bigint,
@domicilio varchar(200), @mail varchar(200))
AS
BEGIN

    Declare @id int
    Set @id = (Select max(idcliente) from cliente)

    If @razonsocial is null or @razonsocial='' or @cuit=0
        Select 'No se podrá insertar el cliente. Verificar datos'
    Else
    Begin
        If len(@mail)=0
            Set @mail='NO ASIGNADO'

        INSERT INTO Cliente (id,razonsocial,cuit,domicilio,mail)
        VALUES(@id+1, @razonsocial, @cuit, @domilicio, @mail)

    end

END

```

```

CREATE PROCEDURE p_nuevocliente (@razonsocial varchar(100), @cuit bigint,
@domicilio varchar(200), @mail varchar(200), @newid INT output)
AS
BEGIN
    Set @newid = (Select max(idcliente) from cliente)+1

    INSERT INTO Cliente (id,razonsocial,cuit,domicilio,mail)
    VALUES(@newid, @razonsocial, @cuit, @domilicio, @mail)
END

```

```

declare @rz varchar(200) = 'Perez S.A.'
declare @cuit bigint = 30258761234
declare @dom varchar(200) = 'Salta 123'
declare @new int

EXEC p_nuevocliente @rz,@cuit,@dom,@new OUTPUT

print @new

```



Function

Es un conjunto de sentencias SQL que pueden ejecutarse, con tan solo invocar su nombre y **en su nombre** devolverá una respuesta. Las funciones son similares a las funciones que se generan en otros lenguajes de programación. En ellas se podrá:

- Incluir 1 ó n sentencias SQL DML.
- Aceptar parámetros de entrada y podrá emitir parámetros de salida.
- Se podrá invocar a otra función.
- Utilizar estrategias de programación, tales como variables, ciclos, condicionales.
- Devolver un valor en su nombre. El valor que devuelve será un valor escalar o también podrá devolver un valor de tipo table

Crear Function

```
CREATE FUNCTION f_ProximoCliente ()  
RETURNS int  
AS  
BEGIN  
    declare @ult int  
    Set @ult=(select coalesce(max(idcliente),0) from Cliente)  
  
    return @ult + 1  
END  
  
CREATE FUNCTION f_Clientes ()  
RETURNS table  
AS  
    return (select idcliente,razonsocial from Cliente)
```

Invocar Function.

```
CREATE FUNCTION f_Clientes ()  
RETURNS table  
AS  
    return (select idcliente,razonsocial from Cliente)
```

Select * from f_clientes()

```
CREATE FUNCTION f_ProximoCliente ()  
RETURNS int  
AS  
BEGIN  
    declare @ult int  
    Set @ult=(select coalesce(max(idcliente),1) from Cliente)  
  
    return @ult + 1  
END
```

Select f_proximocliente()

```
Insert into cliente values (f_proximocliente(),'Juan')
```

Borrar Function.

```
DROP FUNCTION f_Clientes
```

```
DROP FUNCTION IF EXISTS f_Clientes
```

Cambiar Function.

```
ALTER FUNCTION f_Clientes ()
RETURNS table
AS
    return (select idcliente as idx, razonsocial from Cliente)
...
INVOCAR:   SELECT * FROM F_CLIENTES() WHERE idx=10
```

Trigger

Es un conjunto de sentencias SQL que sólo se disparan cuando se produce un evento. Los eventos de DML pueden ser de Update, Delete o Insert. En ellos se podrá:

- Incluir 1 ó n sentencias SQL, ya sea de DML o DDL.
- Utilizar estrategias de programación, tales como variables, ciclos, condicionales.
- Utilizar el mismo trigger para distintos eventos de DML.
- Utilizar sólo para una única tabla o vista.
- Utilizar las tablas temporales deleted, inserted para verificar lo que está tratando de cambiarse en el trigger

Trigger: Crear

```
CREATE TRIGGER tg_borrarclientes ON cliente INSTEAD OF DELETE
AS
    If not exists(Select 1 from factura f
                  Where f.nrocliente in (select nro from deleted))
        delete from cliente
        where nro in(select nro from deleted)

CREATE TRIGGER tg_clientes ON cliente AFTER INSERT,UPDATE
AS
    UPDATE cliente
    Set fechamodificacion=getdate()
    Where nro in (Select nro from inserted)
```

Trigger: Borrar

```
DROP TRIGGER tg_clientes
```

```
DROP TRIGGER IF EXISTS tg_clientes
```

Trigger: Cambiar

```
ALTER TRIGGER tg_borrarclientes ON cliente INSTEAD OF DELETE
AS
BEGIN
    If not exists(Select 1 from factura f
                  Where f.nrocliente in (select nro from deleted))
        delete from cliente
        where nro in(select nro from deleted)
END

ALTER TRIGGER tg_clientes ON cliente AFTER INSERT,UPDATE
AS
BEGIN
    UPDATE cliente
    Set fechamodificacion=getdate()
    Where nro in (Select nro from inserted)
END
```

Trigger: Habilitar / Deshabilitar

```
ENABLE TRIGGER tg_clientes on Cliente

DISABLE TRIGGER ALL on Cliente
```