TEMA 3. MODELO RELACIONAL

1. **TERMINOLOGÍA RELACIONAL**
2. **ESTRUCTURA DEL MODELO RELACIONAL**

   2.1 DEFINICIÓN DE BASE DE DATOS RELACIONAL

   2.2 NORMALIZACIÓN
3. **REGLAS DE INTEGRIDAD RELACIONAL**
4. **LENGUAJES RELACIONALES**

   4.1 INTRODUCCIÓN

   4.2 ÁLGEBRA RELACIONAL

   4.3 CÁLCULO RELACIONAL

1. TERMINOLOGÍA RELACIONAL

El **modelo relacional** se ocupa de la **estructura**, de la **integridad** y de la **manipulación de los datos**, cada una de estas partes tiene sus propios términos especiales. A continuación se exponen los **términos relacionados con la estructura de datos**.

- ✓ Una **Relación** se corresponde con lo que hasta ahora hemos llamado tabla en general.
- ✓ Una **Tupla** corresponde a una fila de esa tabla y un **Atributo** a una columna (campo). El número de filas de una relación se denomina **Cardinalidad** y al número de atributos se le llama **Grado**.
- ✓ La **Clave Primaria** es un **identificador único** para la tabla; es decir, **una columna** o **combinación de columnas** con la siguiente propiedad: **nunca** existen dos filas de la tabla con el **mismo valor** en esa columna o combinación de columnas.
- ✓ Un **Dominio** es una **colección de valores**, de los cuales uno o más **atributos** (columnas) **obtienen** sus **valores reales**. Por ejemplo, el dominio del código de proveedor es el conjunto de todos los códigos de proveedor legales, y el conjunto de valores que aparece en ese atributo en cualquier momento es algún subconjunto de ese conjunto.

Una **relación R** sobre un conjunto de dominios D1,D2,....,Dn (no necesariamente todos distintos), se **compone** de dos partes, una **cabecera** y un **cuerpo**.

Ejemplo:

PROVEEDORES

S#	NOMBRE	CIUDAD
S1	Salazar	Londres
S2	Jaimes	París
S3	Bernal	París
S4	Corona	Londres
S5	Aldana	Atenas

Esta tabla es la **relación** de PROVEEDORES.

La **Clave primaria** es S#

Las **tuplas** son cada una de estas filas

S1	Salazar	Londres
S2	Jaimes	París
S3	Bernal	París
S4	Corona	Londres
S5	Aldana	Atenas

La **cardinalidad** es 5

El **grado** es 3

La **cabecera** es

S#	NOMBRE	CIUDAD

Junto con sus correspondientes Dominios:

El Dominio de S# es S y un número natural.

El Dominio de Nombre el conjunto de caracteres que formen nombres posibles.

El Dominio Ciudad es el conjunto de ciudades de los proveedores.

Y el **cuerpo**

S1	Salazar	Londres
S2	Jaimes	París
S3	Bernal	París
S4	Corona	Londres
S5	Aldana	Atenas

Se podrían establecer las siguientes equivalencias.

Término relacional formal Equivalentes informales

Relación ----------Tabla

Tupla ----------Fila o registro

Cardinalidad----------Número de filas

Atributo ----------Columna o campo

Grado ----------Número de columnas

Clave primaria ----------Identificador único

Dominio----------Conjunto de valores permitidos

CONDICIONES QUE DEBE CUMPLIR UNA RELACIÓN

Las **condiciones** que debe cumplir **una relación o tabla** se especifica en los siguientes puntos:

1) Puede contener un **solo tipo de registro**, cada uno tiene un número fijo de campos con su correspondiente nombre. **La base de datos estará formada por muchas tablas**, de modo que **cada tipo de registro** será una **tabla diferente**. (No puede haber dos tablas iguales)
2) Dentro de una **relación no existen tuplas repetidas**. Esta propiedad se desprende de del hecho de que el **cuerpo** de una **relación es un conjunto matemático**.
3) El **orden de las tuplas** en una tabla **no está determinado**. Por el mismo motivo que el punto anterior.
4) Dentro de una **relación no existen atributos repetidos**. Esta propiedad se desprende del hecho de que la **cabecera** de una relación se define también como un **conjunto matemático**.
5) El **orden de los atributos** en una tabla **no está determinado**. Por el mismo motivo que el punto anterior.

2. ESTRUCTURA DEL MODELO RELACIONAL

2.1. DEFINICIÓN DE BASE DE DATOS RELACIONAL

Una base de datos relacional es una base de datos percibida por el usuario como una colección de ***relaciones normalizadas*** *de diversos* ***grados*** *que* ***varía con el tiempo****.*

Además, hemos de tener en cuenta que los **sistemas relacionales operan** conceptualmente sobre **tablas de datos completas**, mediante la especificación **de operaciones sobre ellas**.

2.2. NORMALIZACIÓN

Supongamos que tenemos una base de datos compuesta por una relación (tablas) y que contiene información sobre Proveedores, Piezas y Cantidades.

CodProveedor	Ciudad	Pais	CodPieza	Cantidad
S1	Londres	Inglaterra	P1	200
S1	Londres	Inglaterra	P2	1300
S1	Londres	Inglaterra	P4	345
S2	Madrid	España	P1	24
S2	Madrid	España	P6	988
S3	París	Francia	P2	200
S4	Londres	Inglaterra	P3	34

Es fácil darse cuenta de que esta base de datos **no está demasiado bien diseñada**, **debido al alto grado de redundancia que existe**, por ejemplo no es necesario especificar tantas veces que el vendedor S1 es de Londres, ya que con que hubiese una sola referencia a este hecho sería suficiente.

Esta redundancia a su vez **provoca** graves **problemas** de **manipulación** de los datos. Estos problemas pueden ser:

- ***Problemas de actualización***: Por ejemplo, si el proveedor S1 cambia de ciudad, sería necesario actualizar al nuevo valor todas las tuplas en las cuales aparezca dicho proveedor, con el riesgo de la equivocación del operador y por lo tanto de una errónea recuperación de los datos.
- ***Problemas de Inserción***: No sería posible insertar un nuevo proveedor hasta que no se le hiciera un pedido.
- ***Problemas de Borrado***: si borramos un proveedor al que solo le hemos hecho un pedido entonces también borramos el pedido.

Las Formas Normales (o la Normalización) son una serie de reglas que, de cumplirse aseguran que el esquema diseñado tendrá un buen comportamiento en cuanto a redundancia, **pérdida de información y representación de la misma**.

Podemos definir la ***Normalización*** *como una* ***técnica para agrupar la información de los datos en diferentes conjuntos de modo que se faciliten los procesos de manipulación****.*

Las tablas o relaciones de una base de datos relacional deben estar normalizadas, es decir todas las tablas de la base de datos deben estar en tercera forma normal (3FN).

Antes de definir las formas normales, es necesario conocer el concepto de Dependencia Funcional, Dependencia Funcional Plena Y Dependencia Transitiva.

DEPENDENCIA FUNCIONAL

Dados dos atributos X e Y de una relación R, se dice que **Y *depende funcionalmente* de X** si, y solo si, en cualquier instante, cada valor de X tiene asociado, en la relación R, un único valor de Y. Suele notarse:

$X.R \rightarrow Y.R$

O, para simplificar: $X \rightarrow Y$

Ejemplo: CodEmp → NomEmp

DEPENDENCIA FUNCIONAL PLENA (También se conoce como dependencia funcional completa)

Se dice que un atributo **Y tiene *dependencia funcional plena* de un atributo compuesto X (X_1,X_2)** si es funcionalmente dependiente de X y no lo es de ninguno de los subconjuntos de X_1 y X_2 de X.

$(X_1,X_2) \rightarrow Y$
$X_1 \not\rightarrow Y$
$X_2 \not\rightarrow Y$

Ejemplo: (NumFac, CodArt) → Cantidad

DEPENDENCIA TRANSITIVA

Un atributo depende transitivamente de otro si, y solo si, depende de él a través de otro atributo. Así, Z depende transitivamente de X, si:

$X \rightarrow Y$
$Y \rightarrow Z$, entonces $X \xrightarrow{DT} Z$

Ejemplo: CodCiudad → CodComunidadAutonoma
CodComunidadAutonoma → NomComAut
Entonces: NomComAut depende transitivamente de CodCiudad.

FORMAS NORMALES

PRIMERA FORMA NORMAL (1FN)

Una relación (tabla) está en primera forma Normal (1FN) si se cumple la propiedad de que sus **dominios no tienen elementos que a su vez sean conjuntos.**

Es decir, todos los dominios simples subyacentes contienen solo valores atómicos (el cruce de una fila con una columna contiene solo un valor).

EJEMPLO

S#	P#	Cant
S1	P1	300
	P2	300
	P3	400
S2	P1	300
	P2	400

Tabla sin Normalizar

S#	P#	Cant
S1	P1	300
S1	P2	300
S1	P3	400
S2	P1	300
S2	P2	400

Tabla en 1FN

Ejemplo: Continente – Habitat. Suele ocurrir cuando debe surgir una relación muchos a muchos y se modela poniéndola uno a muchos. También si en una relación 1:N se coloca la clave foránea en la entidad equivocada.

SEGUNDA FORMA NORMAL (2FN)

Una relación está en Segunda Forma Normal (2FN) si, además de estar en 1FN, **cualquiera de sus atributos no primarios tienen** una dependencia funcional plena con la clave primaria de la relación.

Tiene sentido estudiar si una relación está en 2FN solo si la clave primaria es compuesta. Por tanto, también decimos que está en Segunda Forma Normal, si además de estar en 1FN no tiene claves compuestas.

Ejemplo: Detalle_Pedido – Material. Suele ocurrir cuando la tabla tiene una clave primaria compuesta y en ella hay información de más de una tabla, porque hay atributos que tienen dependencia funcional solo de una parte de la clave primaria.

TERCERA FORMA NORMAL (3FN)

Una relación está en Tercera Forma Normal (3FN) si está en 2FN y además **ninguno de sus atributos no primarios tiene dependencias transitivas respecto de la clave de la relación**.

Ejemplo: Polígono – Ciudad. Suele ocurrir cuando en la tabla (que puede tener clave primaria compuesta o no) hay información de más de una tabla porque hay atributos que tienen dependencia funcional de otro atributo no primario.

EJEMPLO: Normalizar hasta 3FN la siguente relación. Esta tabla pretende almacenar la cantidad de piezas que un vendedor ha vendido durante el tiempo que lleva en la empresa.

R

CodVendedor	**Ciudad**	**Pais**	**CodPieza**	**Cantidad**
S1	Londres	Inglaterra	P1	200
S1	Londres	Inglaterra	P2	1300
S1	Londres	Inglaterra	P4	345
S2	Madrid	España	P1	24
S2	Madrid	España	P6	988
S3	Paris	Francia	P2	200
S4	Barcelona	España	P3	34

La clave primaria es (CodVendedor, CodPieza).

1FN: La relación se encuentra en Primera Forma Normal porque no hay dominios que sean a su vez conjuntos.

2FN: Cada relación debe estar en 1FN y todos los atributos no primarios deben tener dependencia funcional plena con la clave primaria de la relación. En este caso la clave primaria es compuesta, por tanto, tiene sentido hablar de dependencia funcional plena. Estudiemos las dependencias funcionales.

(CodVendedor,CodPieza) $\not\rightarrow$ Ciudad Ciudad depende solo del Vendedor

(CodVendedor,CodPieza) $\not\rightarrow$ Pais Pais depende solo del Vendedor

(CodVendedor,CodPieza) $\rightarrow$ Cantidad

Por lo tanto, para que la relación quede en 2FN, la relación R desaparece, dando lugar a dos relaciones: R1 que contiene la clave primaria junto con los atributos que dependen

funcionalmente de manera plena de esta. Y R2 con una parte de la clave primaria y los atributos que dependen funcionalmente de ella.

R1:	***CodVendedor*** ***CodPieza*** Cantidad
R2:	***CodVendedor*** Ciudad Pais

3FN: Cada relación debe estar en 2FN y ningún atributo no primario debe tener dependencias transitivas respecto de la clave de la relación. En este caso R1 está en 3FN, pero R2 no, porque País depende transitivamente de CodVendedor.

CodVendedor→Ciudad y Ciudad→País

implica CodVendedor→País,

entonces País depende transitivamente de CodVendedor.

Por lo tanto para que la relación R2 quede en 3FN, la relación R2 desaparece, dando lugar a dos relaciones: R21 con la clave primaria y el atributo que depende funcionalmente de ella. Y R22 con el atributo que tenía dependencia transitiva respecto a la clave primaria y el atributo del que dependía funcionalmente.

R21	**CodVendedor** Ciudad
R22	**Ciudad** Pais

La base de datos en 3FN está compuesta por las relaciones: R1, R21, R22.

3. REGLAS DE INTEGRIDAD RELACIONAL

Cualquier base de datos refleja la realidad. Ahora bien algunas **configuraciones** de valores sencillamente no tienen sentido, en cuanto a que **no reflejan ningún posible estado del mundo real**. Por ejemplo, que en un **pedido** aparezca una **cantidad negativa**. Por lo tanto, es necesario **ampliar la definición de la base de datos**, a fin de **incluir** ciertas **reglas de integridad**, cuya **función** es informar al **SGBD** de **ciertas restricciones del mundo real**.

La lista de las reglas para la base de datos de proveedores, piezas y pedidos podría ser:

- ✓ Las cantidades son múltiplos de 100
- ✓ Las cantidades no pueden ser negativas
- ✓ Los números de los proveedores deben tener la forma Snnn
- ✓ Los números de las piezas deben tener la forma Pnnn

PROVEEDORES

S#	NOMBRE	CIUDAD
S1	Salazar	Londres
S2	Jaimes	París
S3	Bernal	París
S4	Corona	Londres
S5	Aldana	Atenas

PIEZAS

P#	NOMBREP	PRECIO
P1	TORNILLO	10
P2	TUERCA	20
P3	ARANDELA	15
P4	MARTILLO	17

PEDIDOS

S#	P#	CANTIDAD
S1	P1	200
S1	P2	300
S2	P4	500
S3	P1	700
S3	P2	1200

La mayor parte de las **reglas de integridad son *específicas***, en cuanto a que **se aplican a bases de datos concretas**. Los ejemplos anteriores son específicos en ese sentido. El modelo relacional, en cambio incluye **dos *reglas generales de integridad***. Generales en el sentido de que **se aplican** no solo a una base de datos especifica como la de proveedores, piezas y pedidos, sino a **todas las bases de datos**. Estas dos reglas generales de integridad hacen **referencia**, respectivamente, a las **claves primarias** y a las **claves ajenas**.

Por último, señalar que **todas las reglas de integridad**, tanto las especificas con respecto una base de datos, como las dos reglas generales del modelo relacional, **se aplican a las relaciones base**.

Una ***Relación Base*** corresponde al concepto de **tabla en SQL**; es decir una relación autónoma, con nombre. En otras palabras, las relaciones base son aquellas cuya importancia es tal que el diseñador de la base de datos ha decidido darle un nombre y

hacerlas parte directa de la base de datos en sí, a diferencia de otras relaciones cuya naturaleza es más efímera, como por ejemplo el resultado de una consulta.

Las reglas generales de integridad son:

- ✓ **Regla de Integridad de la Entidad**
- ✓ **Regla de Integridad Referencial**

A continuación analizaremos las claves primarias y la regla de integridad de la entidad, y las claves ajenas y la regla de integridad referencial.

CLAVES PRIMARIAS.

Se dice que K es una ***Clave Candidata*** o clave posible de una relación R, si es **un atributo** (dominio) o **combinación de atributos** (dominios) de R, tal que cada fila es identificada de formas unívoca por la componente K.

Las claves candidatas de una relación pueden **ser varias**.

Una ***Clave Primaria*** es cualquier **elección** de una clave candidata; al resto le llamamos ***Claves Alternativas***. En la práctica elegimos como clave primaria la que tiene verdadera importancia.

Las **claves primarias tienen tanta importancia**, porque constituyen el **mecanismo de direccionamiento a nivel de tuplas básico en un sistema relacional.**

REGLA DE INTEGRIDAD DE LAS ENTIDADES

Ningún componente de clave primaria de una relación base puede aceptar nulos.

CLAVE AJENA.

En **términos informales** se puede definir una Clave Ajena como un atributo (**quizás compuesto**) de una relación R2, cuyos valores deben tener concordancia con los de la clave primaria de alguna relación R1 (donde R1 y R2 no necesariamente son distintos).

La **relación que contiene la clave ajena** se conoce como ***Relación Referencial***, y la **relación que contiene a la clave primaria** correspondiente se denomina relación referida o ***Relación Objetivo***. Podemos representar esta situación con un ***Diagrama Referencial***.

Ejemplo.

S# y P# de la relación PEDIDO son claves ajenas.

S# representa una referencia a la tupla donde se encuentra el valor correspondiente de la clave primaria (tupla objetivo). PEDIDO es la relación referencial y PROVEEDORES es la relación objetivo.

P# representa una referencia a la tupla donde se encuentra el valor correspondiente de la clave primaria (tupla objetivo). PEDIDO es la relación referencial y PIEZAS es la relación objetivo.

El Diagrama Referencial es:

PROVEEDORES ⟵ PEDIDOS ⟶ PIEZAS

La **definición formal** del término **clave ajena** es la siguiente:

El atributo LF (quizás compuesto) de la relación base R2 es una clave ajena si y solo si satisface estas dos propiedades independientemente del tiempo:

- ✓ Cada valor de LF es nulo del todo o bien no nulo del todo; es decir, si LF es compuesto, todos los componentes son nulos o bien todos los componentes son no nulos, no una combinación.
- ✓ Existe una relación base R1 con clave primaria LP, tal que cada valor no nulo de LF es idéntico al valor de LP en alguna tupla de R1.

Surgen los siguientes puntos:

a) Una **clave ajena** dada y la **clave primaria correspondiente** deben definirse sobre el **mismo dominio**.

b) La **clave ajena no necesita ser** un componente de la **clave primaria** de la relación que la contiene.

Ejemplo:

DEPTO (NUMDEP, ..., PRESUP,...)

EMP (NUMEMP, ..., NUMDEP, ..., SALARIO, ...)

En esta base de datos, el atributo EMP.NUMDEP es una clave ajena en la relación EMP (y concuerda con la clave primaria DEPTO.NUMDEP de la relación DEPTO); sin embargo, no es componente de la clave primaria EMP.NUMDEP de esa relación EMP.

c) Las Relaciones **R1 y R2** en la definición de clave ajena **no son necesariamente distintas**. Es decir, una relación podría incluir una clave ajena cuyos valores (no nulos) deben concordar con los valores de la clave primaria de esa misma relación.

Ejemplo:

EMP (NUMEMP,...,SALARIO,...,NUMEMP_GER,...)

Aquí, el atributo NUMEMP_GER representa el número de empleado que es gerente del empleado identificado mediante NUMEMP. NUMEMP es la clave primaria, y NUMEMP_GER es una clave ajena que hace referencia a ella. Este tipo de relaciones se denominan ***autorreferenciales***.

d) Las **claves ajenas**, a diferencia de las claves primarias, pueden **aceptar nulos.**

REGLA DE INTEGRIDAD REFERENCIAL.

La base de datos no debe contener valores de clave ajena sin concordancia.

REGLAS PARA CLAVES AJENAS.

Cualquier estado de la base de datos que no satisfaga la regla de integridad referencial será incorrecto por definición, pero ¿cómo pueden evitarse tales estados incorrectos?

Una posibilidad, es que el **sistema rechazara** cualquier operación que en caso de ejecutarse, produciría un estado ilegal. Pero en muchos caso una **alternativa** preferible, sería que el **sistema aceptara la operación**, pero realizara (en caso necesario) ciertas **operaciones de compensación** con objeto de garantizar el estado legal como resultado final.

En consecuencia, para **cualquier base de datos, el diseñador de esta deberá poder especificar cuales operaciones han de rechazarse y cuales han de aceptarse, y en el caso de estas últimas, cuáles operaciones de compensación (si acaso) deberá realizar el sistema.**

Para cada clave ajena es necesario responder las siguientes tres preguntas:

1. **¿Puede aceptar nulos?.**

   La respuesta dependerá de la política vigente en la porción del mundo real que se esté modelando.

   En PEDIDOS no tiene sentido permitir que exista un pedido y no especifiquemos el proveedor. Sin embargo, en EMPLEADOS si podrá existir un empleado no asignado a un departamento.

2. **¿Qué deberá suceder si hay un intento de eliminar el objetivo de una referencia de clave ajena?**

   Aplicamos esta pregunta a nuestro ejemplo: ¿Qué debe suceder si intentamos eliminar un proveedor del cual existe al menos un pedido?

   Entonces existen tres posibilidades:

   a) RESTRINGIDA. (RESTRICTED – NONE – NO ACTION)

      La operación está restringida (limitada) al caso en el cual no existan envíos. (La operación se va a realizar si no existen valores de clave ajena).

   b) SE PROPAGA (CASCADE).

      La operación de eliminación se propaga en cascada, eliminando también los envíos correspondientes.

   c) ANULA (NULLFIELD – SET NULL)

      Se asignan valores nulos a la clave ajena en todos los envíos correspondientes y se elimina el proveedor. Este caso **no es aplicable si la clave ajena no acepta nulos**.

3. **¿Qué deberá suceder si hay un intento de modificar el objetivo de una referencia de clave ajena?**

Aplicamos esta pregunta a nuestro ejemplo: ¿Qué debe suceder si intentamos modificar un proveedor del cual existe al menos un pedido?

Entonces existen tres posibilidades:

a) RESTRINGIDA. (RESTRICTED – NONE – NO ACTION)

La operación de modificación está restringida (limitada) al caso en el cual no existan envíos. (La operación se va a realizar si no existen valores de clave ajena).

b) SE PROPAGA (CASCADE).

La operación de modificación se propaga en cascada, modificando también la clave ajena en los envíos correspondientes.

c) ANULA (NULLFIELD – SET NULL)

Se asignan valores nulos a la clave ajena en todos los envíos correspondientes y se modifica el proveedor. Este caso **no es aplicable si la clave ajena no acepta nulos**.

Concretando, **para cada clave ajena** del diseño, **el diseñador** de la base de datos deberá **especificar**:

1. El atributo o combinación de atributos que constituyen esa clave ajena.
2. La relación objetivo a la cual hace referencia esa clave ajena.
3. La respuesta a las tres cuestiones anteriores. Es decir, la respuesta a las reglas para claves ajena:
   - ✓ Regla de Nulos
   - ✓ Regla de Eliminación.
   - ✓ Regla de Modificación.

Para poder especificar estas tres reglas utilizaremos la siguiente propuesta de sintaxis para definir una relación base.

Ejemplo de uso de reglas de eliminación y modificación

```
CREATE TABLE Nombre_tabla
        (Atributo1....[NOT NULL], Atributo2 ....[NOT NULL],.....,  AtributoN ....[NOT NULL],
        PRIMARY KEY (Atributos que forma la clave primaria),
        FOREIGN KEY (Atributos que forman la clave ajena) REFERENCES Objetivo(Atributo que es clave  primaria)
        ON UPDATE Efecto ON DELETE Efecto
```

Donde "Efecto" puede ser:

- ✓ RESTRICTED - NONE – NO ACTION – RESTRICT (Restringido)
- ✓ CASCADE (Se propaga)
- ✓ NULLFIELD – SET NULL (Anula)

Ejemplo de creación de una tabla de en MySQL:

```
CREATE TABLE departamento(
CodDep int NOT NULL ,
NomDep varchar( 60 ) NOT NULL ,
PRIMARY KEY ( CodDep )
)


CREATE TABLE empleado(
CodEmp int NOT NULL ,
NomEmp varchar( 60 ) NOT NULL ,
CodDep int ,
PRIMARY KEY ( CodEmp ) ,
FOREIGN KEY ( CodDep ) REFERENCES departamento( CodDep ) ON DELETE SET NULL ON
UPDATE CASCADE
)
```

4. LENGUAJES RELACIONALES

4.1. INTRODUCCIÓN

- El **Álgebra Relacional** y el **Cálculo Relacional** son **dos alternativas** para establecer una **base formal** de la parte **manipulativa del Modelo Relacional**.
- El **objetivo** de estos lenguajes **no es la obtención de datos** sino **ayudarnos a escribir expresiones.**
- **SQL** es en realidad un **híbrido entre el álgebra y el cálculo**. Por ejemplo, ofrece el cuantificador existencial "EXIST" (de cálculo) y el operador UNIÓN (de álgebra). Aunque **la intención original** era construir **algo distinto del álgebra y del cálculo**.
- **El álgebra y el cálculo relacional son totalmente equivalentes**, de forma que para cualquier expresión del álgebra existe una expresión del cálculo y viceversa.
- En cálculo **definimos la tabla resultado**, mientras que el álgebra **proporciona el procedimiento para resolver el problema**.
- El **cálculo** se parece más al **lenguaje natural** mientras que el **álgebra** se parece más a un **lenguaje de programación**.

4.2. ÁLGEBRA RELACIONAL

El álgebra relacional **consiste** en un **conjunto de operadores de alto nivel que operan sobre relaciones**. Cada uno de los operadores toma una o dos relaciones como entrada y produce una nueva relación como salida.

Existen **ocho operadores** que podemos dividir para su estudio en dos grupos: las **tradicionales de conjuntos** y las **relacionales básicas**.

OPERACIONES TRADICIONALES DE CONJUNTOS.

Para que se puedan realizar las operaciones de **UNIÓN**, **INTERSECCIÓN** Y **DIFERENCIA** las tablas con las que trabaja cada una, deben ser compatibles; es decir, **se debe de cumplir**:

a. Las **relaciones** deben tener el **mismo grado**.
b. Los **atributos** deben estar **definidos sobre los mismos dominios**.

UNIÓN

La Unión de dos tablas R ∪ S, es el conjunto de todas las filas pertenecientes a R o a S o a ambas, evidentemente **eliminando duplicaciones**.

Sintaxis:

R UNION S

INTERSECCIÓN

La Intersección de dos tablas R ∩ S, es el conjunto de todas las filas que aparecen en ambas tablas. La tabla resultante tiene todas las filas que pertenecen (son iguales) a la tabla R y a la S.

Sintaxis:

R INTERSECCION S

DIFERENCIA

La Diferencia de dos tablas R - S, es el conjunto de todas las filas que pertenezcan a R y no a S. El orden de los factores es importante.

Sintaxis:

R MINUS S

EJEMPLOS:

Tabla A

S#	NOMBRE	CIUDAD
S1	Salazar	Londres
S4	Corona	Londres

Tabla B

S#	NOMBRE	CIUDAD
S1	Salazar	Londres
S2	Jaimes	París

a. A UNION B

S#	NOMBRE	CIUDAD
S1	Salazar	Londres
S4	Corona	Londres
S2	Jaimes	Paris

b. A INTERSECCIÓN B

S#	NOMBRE	CIUDAD
S1	Salazar	Londres

c. A MINUS B

S#	NOMBRE	CIUDAD
S4	Corona	Londres

PRODUCTO CARTESIANO

Dadas dos tablas R y S que contienen filas r y s, entonces R * S es el conjunto de todas las filas posibles f obtenidas concatenando una fila r y una s. La concatenación de una fila r = (r1, r2, ..., rp) y una fila s = (s1, s2, ..., sq) es una fila:

f = (r1, r2, ..., rp, s1, s2, ..., sq).

Otra forma de expresar el producto cartesiano es como el conjunto de todas las posibles filas formadas tomando una fila de R y otra se S y concatenándolas.

Sintaxis:

R TIMES S

EJEMPLO:

R

TALLA
1
2

S

COLOR
Blanco
Negro

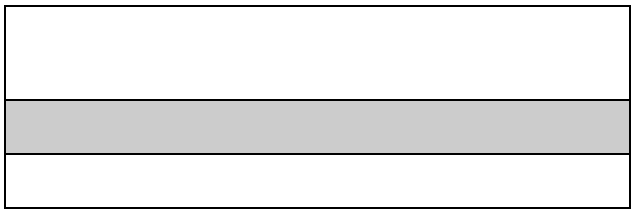
R*S

TALLA	COLOR
1	Blanco
1	Negro
2	Blanco
2	Negro

OPERACIONES RELACIONALES.

SELECCIÓN.

Selecciona ciertas filas de una tabla. La tabla original se procesa y como resultado **se obtiene una nueva tabla** mediante el uso de algún criterio de selección.



PROYECCIÓN

Crea una nueva tabla a partir de una dada que contiene algunas columnas de la original, ignorando el resto.

CONCATENACIÓN

Consiste en **establecer una correspondencia** entre los **valores que contiene** un **par de columnas pertenecientes una a cada tabla**. Las columnas representan generalmente al mismo atributo, aunque es suficiente con que tengan el **mismo dominio de valores**. Las dos tablas que participan en la concatenación pueden tener distinto número de columnas y filas.

Ejemplo:

R

A	B
A1	B1
A2	B1
A3	B2

S

B	C
B1	C1
B2	C2
B3	C3

R concatenación S sobre B

A	B	C
A1	B1	C1
A2	B1	C1
A3	B2	C2

DIVISIÓN

Sea R una tabla binaria cuyos atributos son A y B. Sea S una tabla con un solo atributo C. Supongamos que los atributos B y C tienen el mismo dominio. Entonces podemos dividir R entre S sobre B y C, produciéndose una tabla cociente con dominio en A.

El cociente T contiene filas en una sola columna con dominio en A. Un valor para T aparecerá solo cuando R contenga filas (a , b) para todos los valores de B en S.

Ejemplo:

R S T : Dividir R entre S sobre COLOR

TALLA	COLOR
1	Blanco
1	Negro
1	Rojo
2	Blanco
2	Negro
3	Blanco

COLOR
Blanco
Negro

TALLA
1
2

Las tres operaciones básicas relacionales tienen, en álgebra, la siguiente notación:

SELECT tabla WHERE condición GIVING tabla_resultado --------Operación Relacional: SELECCIÓN

PROJECT tabla OVER atributo/s GIVING tabla_resultado --------Operación Relacional: PROYECCIÓN

JOIN tabla1 AND tabla2 OVER atributo_común GIVING tabla_resultado---Operación Relacional: CONCATENACIÓN

DIV tabla_binaria IN tabla_unaria OVER atributo_común GIVING tabla_resultado---Operación Relacional: DIVISIÓN

Ejemplos:

Consideremos las siguientes tablas:

T

T#	Talla	Color
T1	1	Blanco
T2	1	Negro
T3	1	Rojo
T4	2	Blanco
T5	2	Negro
T6	3	Blanco

P

P#	Nombre	Ciudad
P1	Juan	Madrid
P2	Ana	Barcelona
P3	José	Málaga

TP

P#	T#
P1	T1
P1	T2
P1	T6
P2	T1
P2	T4
P3	T2

1. Seleccionar los proveedores de Málaga:

   SELECT P WHERE Ciudad = 'Málaga' GIVING RESULTADO

2. Seleccionar los códigos de proveedores que suministran los productos de color blanco.

   SELECT T WHERE Color =' Blanco' GIVING R1

   En R1 obtenemos todas las filas de T donde el color es blanco.

   R1

T#	**Talla**	**Color**
T1	1	Blanco
T4	2	Blanco
T6	3	Blanco

   JOIN R1 AND TP OVER T# GIVING R2

   En R2 concatenamos las tablas R1 y TP sobre el atributo común T#, obteniendo todos los proveedores que suministran productos de color blanco.

   R2

T#	**Talla**	**Color**	**P#**
T1	1	Blanco	P1
T1	1	Blanco	P2
T4	2	Blanco	P2
T6	3	Blanco	P1

   PROJECT R2 OVER P# GIVING RESULTADO

   Nos quedamos con el código del proveedor que es lo que nos interesa.

   RESULTADO

P#
P1
P2

3. Obtener las ciudades de los proveedores que suministran los productos cuyo código es T2.

   SELECT TP WHERE T#='T2' GIVING R1

   JOIN R1 AND P OVER P# GIVING R2

   PROJECT R2 OVER CIUDAD GIVING RESULTADO

4. Encontrar la ciudad de los proveedores que suministran productos de color negro.

   SELECT T WHERE COLOR='NEGRO' GIVING R1

   JOIN R1 AND TP OVER T# GIVING R2

   JOIN R2 AND P OVER P# GIVING R3

   PROJECT R3 OVER CIUDAD GIVING RESULTADO

5. Obtener la ciudad y el nombre de los proveedores que suministran todos los productos.

   PROJECT T OVER T# GIVING R1

   DIV TP IN R1 OVER T# GIVING R2

   JOIN R2 AND P OVER P# GIVING R3

   PROJECT R3 OVER NOMBRE, CIUDAD GIVING RESULTADO

Para **insertar** nuevos elementos en las tablas se utiliza la unión de conjuntos, y para **borrar** la diferencia.

Así para añadir una nueva fila a la tabla T se haría:

T UNION ((‘T7’,’1’,’Rojo’)) GIVING T

Y para eliminar filas:

T MINUS ((‘T3’,’1’,’Rojo’)) GIVING T

Así mismo podemos usar el símbolo “?” para representar cualquier valor. De esta manera para borrar todas las tallas ‘2’ haremos:

T MINUS ((?,’2’,?)) GIVING T

4.3. CÁLCULO RELACIONAL

- El ***cálculo relacional*** es otro método que permite la **manipulación de los datos** y realizar consultas sobre relaciones.
- QUEL (INGRES) y QBE (IBM) se basan en cálculo relacional.

ESTRUCTURA DE LAS OPERACIONES REALIZADAS EN LOS LENGUAJES BASADOS EN CÁLCULO RELACIONAL

<OBJETIVO> WHERE <PREDICADOS>

- OBJETIVO: Especifica las relaciones y atributos requeridos.
- PREDICADOS: Contiene las propiedades o condiciones que deben satisfacerse.
- WHERE: puede sustituirse por ":".

***Ejemplo*:**

Obtener el código y nombre de los proveedores de Madrid.

Álgebra:

SELECT P WHERE Ciudad = 'Madrid' GIVING R1

PROYECT R1 OVER P#, Nombre GIVING RESULTADO

Cálculo:

(P.P#, P.Nombre) WHERE (P.Ciudad = 'Madrid')

Cálculo relacional de tuplas:

Los lenguajes de cálculo relacional de tuplas emplean el concepto de VARIABLE DE TUPLA que actúan **como receptáculo de una/s fila/s de una relación**.

Si definimos el rango de la variable X como la tabla P, entonces se escribe:

RANGE X IS P

X toma los valores de las filas de P.

Conceptualmente tenemos dos tablas X y P, pero realmente tenemos una, P.

El ejemplo anterior se expresa:

Range X is P

(X.P#, X.Nombre) : (X.Ciudad = 'Madrid')

Codd presentó un **lenguaje basado en cálculo de predicados** al que llamó "**Sublenguaje de datos Alpha**". Este no se comercializó y posteriormente dio lugar al QUEL, después al SQUEL y por último al SQL.

QUEL → SQUEL →SQL

Veamos el ejemplo anterior en **QUEL** que es un lenguaje basado en **cálculo relacional** y en **SQL** que es un **híbrido** entre **cálculo y álgebra** relacional.

QUEL

Range of X is P

Retrieve (X.P#, X.Nombre) WHERE X.Ciudad = 'Madrid'

(Se puede omitir la definición de la tupla).

SQL

SELECT P#, Nombre

FROM P

WHERE Ciudad = 'Madrid'

CUANTIFICADORES

Los cuantificadores **se utilizan** como **predicados**. Vamos a estudiar mediante dos ejemplos, el **cuantificador existencial EXIST** y el **cuantificador universal FORALL**.

CUANTIFICADOR EXISTENCIAL EXIST

Range X is P

EXIST X (X.Nombre = 'Jose')

Se lee: Existe una tupla en X cuyo valor en nombre es José.

CUANTIFICADOR UNIVERSAL FORALL

Range Y is T

FORALL Y (Y.Color = 'Rojo')

Se lee: Para todas las tuplas de Y el color es Rojo.

OPERACIONES DE CONJUNTOS

Vamos a mostrar aquí los operadores equivalentes de las operaciones de conjuntos del álgebra relacional.

- F1 $\wedge$ F2 : Es la intersección ($\cap$) del álgebra relacional.
- F1 $\vee$ F2 : Es la operación unión ($\cup$) del álgebra relacional.
- $\neg$ F1 : Equivale a la negación.
- F1 $\wedge \neg$F2: En álgebra relacional equivale a la diferencia, pero en cálculo no se dispone de un operador para expresar esta operación. Se lee: las tuplas que están en F1 y no en F2 (Se realiza utilizando una combinación de las anteriores operaciones).

Nota:

- Si en el objetivo hay una sola relación, en esa relación se pregunta fila a fila si se cumple el predicado.
- Si en el objetivo aparece más de una relación, en el producto cartesiano de estas se pregunta fila a fila si se cumple el predicado.
- Utilizaremos el/los predicado/s EXIST o/y FORALL, si para comprobar el predicado necesitamos de alguna/s relación/es diferente/s a la/s que aparece/n en el objetivo.

Ejemplos:

Range XT is T

Range XP is P

Range XTP is TP

1. Seleccionar el nombre de los proveedores de Málaga.

(XP.NOMBRE) : (XP.CIUDAD=’MÁLAGA’)

2. Seleccionar los códigos de proveedores que suministran los productos de color blanco.

(XTP.P#) : EXIST XT (XT.T#=XTP.T# AND XT.COLOR=’BLANCO’)

3. Obtener las ciudades de los proveedores que suministran los productos cuyo código es T2.

(XP.CIUDAD) : EXIST XTP (XTP.P# = XP.P# AND XTP.T#=’T2’)

4. Encontrar la ciudad de los proveedores que suministran productos de color rojo.

(XP.CIUDAD) : EXIST XTP (XTP.P#=XP.P# AND EXIST XT (XT.T#=XTP.T# AND XT.COLOR=’ROJO’))

5. Obtener la ciudad y el nombre de los proveedores que suministran todos los tipos de productos.

(XP.CIUDAD,XP.NOMBRE):FORALL XT (EXIST XTP(XTP.P#=XP.P# AND XTP.T#=XT.T#)

(XP.CIUDAD,XP.NOMBRE): EXIST XTP(XTP.P#=XP.P# AND FORALL XT (XTP.T#=XT.T#)

6. Obtener el código del proveedor junto con el código del producto que suministra y su correspondiente color.

(XTP.P#, XTP.T#,XT.COLOR): (XTP.T#=XT.T#)