

---

## 5.- Álgebra Relacional y el estándar SQL

### 5.1.- Álgebra Relacional

#### 5.1.1 Selección

#### 5.1.2 Proyección

#### 5.1.3 Operaciones con Conjuntos

#### 5.1.4 Producto Cartesiano

#### 5.1.5 Reunión

#### 5.1.6 Ejemplos de consultas algebraicas

### 5.2 SQL

#### 5.2.1 Introducción

#### 5.2.2 Ambiente de SQL

#### 5.2.3 Definiendo una BD en SQL

#### 5.2.4 Inserción, Actualización y Eliminación de Datos

#### 5.2.5 Procesamiento en tablas

---

## 5.1 Álgebra relacional

- Es una colección de operaciones que son usadas para manipular relaciones completas.
- El resultado de cada operación es una nueva relación, que a su vez también puede ser manipulada.

Está descrito de la siguiente forma:

$$\langle \text{operador} \rangle_{\langle \text{parámetros} \rangle} \langle \text{operandos} \rangle \rightarrow \langle \text{resultado} \rangle$$


# Instancias de relación

## EMP

ENo	ENomb	Titulo
E1	J. Doe	Ing. Eléctrico
E2	M. Smith	Analista Sistemas
E3	A. Lee	Ing. Mecánico
E4	J. Miller	Programador
E5	B. Casey	Analista Sistemas
E6	L. Chu	Ing. Eléctrico
E7	R. Davis	Ing. Mecánico
E8	J. Jones	Analista Sistemas

## PROY

JNo	JNomb	Presup	Lugar
J1	Instrumentacion	150000	Montreal
J2	Desarrollador BD	135000	New York
J3	CAD/CAM	250000	New York
J4	Mantenimiento	310000	Paris
J5	CAD/CAM	500000	Boston

## TRAB

ENo	JNo	Resp	Dur
E1	J1	Admin.	12
E2	J1	Analista	24
E2	J2	Analista	6
E3	J3	Consultor	10
E3	J4	Ingeniero	48
E4	J2	Programador	18
E5	J2	Admin.	24
E6	J4	Admin.	48
E7	J3	Ingeniero	36
E7	J5	Ingeniero	23
E8	J3	Admin.	40

## PAGO

Titulo	Salario
Ing. Eléctrico	40000
Analista Sistema	34000
Ing. Mecánico	27000
Programador	24000

# Operaciones del álgebra relacional

- Aunque muchas de las nociones del álgebra relacional coinciden con el álgebra de conjuntos, es preciso hacer notar que las relaciones usan el modelo de bolsas (bag) más que el modelo de conjuntos (set), por lo que en ocasiones hay diferencias en los resultados.

Las operaciones que pueden efectuarse entre relaciones son:

**Selección** ( $\sigma$ ).- Obtiene el conjunto de tuplas que cumplen (evalúan a verdadero) con la condición de búsqueda expresada como lógica de primer orden. Corresponde a la cláusula WHERE dentro de la sentencia SELECT del SQL.

**Proyección** ( $\pi$ ).- Elige un subconjunto de los atributos de una tabla, y en la cual algunos de estos atributos pueden ser renombrados e incluso calculados a partir de ciertas funciones agregadas. Corresponde a los parámetros de la sentencia SELECT del SQL.

**Unión** ( $\cup$ ), **intersección** ( $\cap$ ) y **diferencia** ( $-$ ).- Estos operadores de conjunto necesitan que los atributos entre las relaciones sean compatibles. La unión realiza la conjunción entre todos los elementos (tuplas) de las tablas, la intersección obtiene solo los elementos que pertenezcan a ambas tablas, y la diferencia obtiene los elementos de una tabla que no están presentes en la otra. Estas operaciones corresponden a los operadores UNION, INTERSECT y EXCEPT del SQL, respectivamente.

# Operaciones del álgebra relacional

**Producto** ( $\times$ ).- También se conoce como producto cartesiano y es la combinación de cada una de las tuplas pertenecientes a dos tablas entre sí, incluyendo en la tabla resultante todos los atributos de ambas. Corresponde a la lista de tablas que viene después de la cláusula FROM del SQL.

**Reunión** ( $\bowtie$ ).- Es parecida al producto cartesiano, solo que se agrega una condición lógica para determinados atributos entre tablas. Existen varios tipos de reuniones: la reunión natural, en la que solo se muestran una vez todos los atributos comunes entre las tablas; la theta-reunión ( $\theta$ ), en la que la condición puede implicar cualquiera de los operadores aritméticos de comparación ( $<$ ,  $>$ ,  $\leq$ ,  $\geq$ ,  $=$  y  $<>$ ), y la equi-reunión, en la que la operación de comparación es la igualdad ( $=$ ).

## 5.1.1 Selección

- Produce un subconjunto de tuplas horizontal de la relación operando.

Forma general:

$$\sigma_F(R) = \{t \mid t \in R \wedge F(t) \text{ es Verdadero}\}$$

donde:  $R$  es una tabla,  $t$  es una variable tupla

$F$  es una fórmula lógica consistente de

- operandos que son constantes o atributos
- operadores aritméticos de comparación ( $<$ ,  $>$ ,  $=$ ,  $\neq$ ,  $\geq$ ,  $\leq$ )
- operadores lógicos ( $\wedge$ ,  $\vee$ ,  $\neg$ )

# Ejemplo

## EMP

ENo	ENomb	Titulo
E1	J. Doe	Ing. Eléctrico
E2	M. Smith	Analista Sistemas
E3	A. Lee	Ing. Mecánico
E4	J. Miller	Programador
E5	B. Casey	Analista Sistemas
E6	L. Chu	Ing. Eléctrico
E7	R. Davis	Ing. Mecánico
E8	J. Jones	Analista Sistemas

$\sigma_{\text{TITULO}=\text{'Ing. Eléctrico'}}(\text{EMP})$

ENo	ENomb	Titulo
E1	J. Doe	Ing. Eléctrico
E6	L. Chu	Ing. Eléctrico

## 5.1.2 Proyección

- Produce una división vertical de una tabla

Forma general

$$\Pi_{A_1, \dots, A_n} (R) = \{ t[A_1, \dots, A_n] \mid t \in R \}$$

donde:  $R$  es una relación,  $t$  es una variable tupla

$\{A_1, \dots, A_n\}$  es un subconjunto de los atributos de  $R$  sobre los cuales la proyección será ejecutada.

- La proyección puede generar tuplas duplicadas
- El SQL permiten esto y proporciona:
  - Proyección con eliminación de duplicados
  - Proyección sin eliminación de duplicados



# Ejemplo




## PROY

JNo	JNomb	Presup	Lugar
J1	Instrumentacion	150000	Montreal
J2	Desarrollador BD	135000	New York
J3	CAD/CAM	250000	New York
J4	Mantenimiento	310000	Paris
J5	CAD/CAM	500000	Boston

## $\Pi_{JNO, PRESUP}(\text{PROY})$

JNo	Presup
J1	150000
J2	135000
J3	250000
J4	310000
J5	500000

## 5.1.3 Operaciones de conjuntos

Union	Interseccion	Diferencia
		
Las personas que cuentan con un medio de transporte: Auto o Bicicleta	Las personas que cuentan Auto y Bicicleta	Las personas que cuentan Bicicleta pero no con Auto
Hugo, Paco, Luis	Paco	Hugo

# Unión

- Similar a la unión de conjuntos
- Compatibilidad
  - mismo grado (número de atributos)
  - atributos correspondientes definidos sobre el mismo dominio

Forma general

$$R \cup S = \{ t \mid t \in R \vee t \in S \}$$

donde  $R$ ,  $S$  son tablas,  $t$  es una variable tupla

El resultado contiene tuplas que están en  $R$  o en  $S$ .

# Diferencia de conjunto

- Forma general

$$R - S = \{ t \mid t \in R \wedge t \notin S \}$$

- El resultado contiene todas las tuplas que están en  $R$ , pero no en  $S$ .

$$R - S \neq S - R$$

- $R, S$  deben ser compatibles en unión

# Intersección

- Es una operación que resulta en otra relación que contiene los elementos comunes a las relaciones iniciales

Definida como:

$$\begin{aligned} R \cap S &= \{ t \mid t \in R \wedge t \in S \} \\ &= R - (R - S) \end{aligned}$$

## 5.1.4 Producto cartesiano

- Dadas las tablas:

$R$  de grado  $n$ , cardinalidad  $x$

$S$  de grado  $m$ , cardinalidad  $y$

el producto cartesiano se define como:

$$(R \times S) = \{ t[A_1, \dots, A_n, B_1, \dots, B_m] \\ | t[A_1, \dots, A_n] \in R \wedge t[B_1, \dots, B_m] \in S \}$$

- El resultado de  $R \times S$  es una tabla de grado  $(m + n)$  y consiste de todas la  $(x \cdot y)$ -tuplas, en donde cada tupla es la concatenación de una tupla de  $R$  con cada una de las tuplas de  $S$ .

# Ejemplo

## EMP

ENo	ENomb	Titulo
E1	J. Doe	Ing. Eléctrico
E2	M. Smith	Analista Sistemas
E3	A. Lee	Ing. Mecánico
E4	J. Miller	Programador
E5	B. Casey	Analista Sistemas
E6	L. Chu	Ing. Eléctrico
E7	R. Davis	Ing. Mecánico
E8	J. Jones	Analista Sistemas

## PAGO

Titulo	Salario
Ing. Eléctrico	40000
Analista Sistema	34000
Ing. Mecánico	27000
Programador	24000

## EMP × PAGO

ENo	ENomb	EMP.Titulo	PAGO.Titulo	Salario
E1	J. Doe	Ing. Eléctrico	Ing. Eléctrico	40000
E1	J. Doe	Ing. Eléctrico	Analista Sistemas	34000
E1	J. Doe	Ing. Eléctrico	Ing. Mecanico	27000
E1	J. Doe	Ing. Eléctrico	Programador	24000
E2	M. Smith	Analista Sistemas	Ing. Eléctrico	40000
E2	M. Smith	Analista Sistemas	Analista Sistemas	34000
E2	M. Smith	Analista Sistemas	Ing. Mecanico	27000
E2	M. Smith	Analista Sistemas	Programador	24000
E3	A. Lee	Ing. Mecanico	Ing. Eléctrico	40000
...	...	...	...	...
E8	J. Jones	Analista Sistemas.	Ing. Eléctrico	40000
E8	J. Jones	Analista Sistemas	Analista Sistemas	34000
E8	J. Jones	Analista Sistemas.	Ing. Mecanico	27000
E8	J. Jones	Analista Sistemas	Programador	24000

## 5.1.5 Reunión

- Forma general

$$R \bowtie_{F(R.A_i, S.B_j)} S = \{ t[A_1, \dots, A_n, B_1, \dots, B_m] \mid t[A_1, \dots, A_n] \in R \wedge t[B_1, \dots, B_m] \in S \wedge F(R.A_i, S.B_j) \text{ es Verdadero} \}$$

Donde  $R$ ,  $S$  son tablas,  $t$  es una variable tupla

$F(R.A_i, S.B_j)$  es una formula definida como una selección

- Una derivación del producto cartesiano

$$R \bowtie_F S = \sigma_F(R \times S)$$




# Ejemplo

**EMP**

ENo	ENomb	Titulo
E1	J. Doe	Ing. Eléctrico
E2	M. Smith	Analista Sistemas
E3	A. Lee	Ing. Mecánico
E4	J. Miller	Programador
E5	B. Casey	Analista Sistemas
E6	L. Chu	Ing. Eléctrico
E7	R. Davis	Ing. Mecánico
E8	J. Jones	Analista Sistemas

**TRAB**

ENo	JNo	Resp	Dur
E1	J1	Admin.	12
E2	J1	Analista	24
E2	J2	Analista	6
E3	J3	Consultor	10
E3	J4	Ingeniero	48
E4	J2	Programador	18
E5	J2	Admin.	24
E6	J4	Admin.	48
E7	J3	Ingeniero	36
E7	J5	Ingeniero	23
E8	J3	Admin.	40

**EMP**  **EMP.En0 > TRAB.En0** **TRAB**

EMP. ENo	ENomb	Titulo	TRAB .ENo	JNo	Resp	Dur
E2	M. Smith	Analista Sistemas	E1	J1	Admin.	12
E3	A. Lee	Ing. Mecánico	E1	J1	Admin.	12
E3	A. Lee	Ing. Mecánico	E2	J1	Analista	24
E3	A. Lee	Ing. Mecánico	E2	J2	Analista	6
E4	J. Miller	Programador	E1	J1	Admin.	12
E4	J. Miller	Programador	E2	J1	Analista	24
E4	J. Miller	Programador	E2	J2	Analista	6
E4	J. Miller	Programador	E3	J3	Consultor	10
E4	J. Miller	Programador	E3	J4	Ingeniero	48
E5	B. Casey	Analista Sistemas	E1	J1	Admin.	12
E5	B. Casey	Analista Sistemas	E2	J1	Analista	24
E5	B. Casey	Analista Sistemas	E2	J2	Analista	6
E5	B. Casey	Analista Sistemas	E3	J3	Consultor	10
E5	B. Casey	Analista Sistemas	E3	J4	Ingeniero	48
E5	B. Casey	Analista Sistemas	E4	J2	Programador	18
...	...	...	...	...	...	...

# Tipos de reuniones

## $\theta$ -reunión

La fórmula  $F$  usa el operador  $\theta$

## Equi-reunión

La fórmula  $F$  contiene el operador de igualdad

$$R \bowtie_{R.A = S.B} S$$

## Reunión natural

Equi-reunión de dos tablas  $R$  y  $S$  sobre un atributo (o atributos) comunes a  $R$  y  $S$  y proyectando sólo una copia de estos atributos

$$R \star S = \Pi_{R \cup S} \sigma_F(R \times S)$$

# Ejemplo (reunión natural)

## EMP

ENo	ENomb	Titulo
E1	J. Doe	Ing. Eléctrico
E2	M. Smith	Analista Sistemas
E3	A. Lee	Ing. Mecánico
E4	J. Miller	Programador
E5	B. Casey	Analista Sistemas
E6	L. Chu	Ing. Eléctrico
E7	R. Davis	Ing. Mecánico
E8	J. Jones	Analista Sistemas

## EMP \* PAGO

ENo	ENomb	Titulo	Salario
E1	J. Doe	Ing. Eléctrico	40000
E2	M. Smith	Analista Sistemas	34000
E3	A. Lee	Ing. Mecánico	27000
E4	J. Miller	Programador	24000
E5	B. Casey	Analista Sistemas	34000
E6	L. Chu	Ing. Eléctrico	40000
E7	R. Davis	Ing. Mecánico	27000
E8	J. Jones	Analista Sistemas	34000

## PAGO

Titulo	Salario
Ing. Eléctrico	40000
Analista Sistema	34000
Ing. Mecánico	27000
Programador	24000

**La reunión es sobre el atributo común *Titulo***

# Tipos de reunión

- Reuniones externas ( $R \bowtie_o S$ )

Se asegura que las tuplas de una o ambas tablas que no satisfacen la condición de reunión aún aparezcan en el resultado final con el conjunto de los valores de los otros atributos establecidos a nulo (NULL)

- Reunión externa izquierda ( $R \bowtie_{o_L} S$ ) - sólo son tomadas en cuenta las tuplas sin correspondencia de la relación R en el resultado final.
- Reunión externa derecha ( $R \bowtie_{o_R} S$ ) - sólo son tomadas en cuenta las tuplas sin correspondencia de la relación S en el resultado final.

# Outer Join

## PROY

JNo	JNomb	Presup	Lugar
J1	Instrumentacion	150000	Montreal
J2	Desarrollador BD	135000	New York
J3	CAD/CAM	250000	New York
J4	Mantenimiento	310000	Paris
J5	CAD/CAM	500000	Boston

## TRAB

ENo	JNo	Resp	Dur
E1	J1	Admin.	12
E2	J1	Analista	24
E2	J2	Analista	6
E3	J3	Consultor	10
E3	J4	Ingeniero	48
E4	J2	Programador	18

## PROY<sub>L</sub> ⋈<sub>o</sub> TRAB

ENo	JNo	JNomb	Presup	Lugar	Resp	Dur
E1	J1	Instrumentacion	150000	Montreal	Admin.	12
E2	J1	Instrumentacion	150000	Montreal	Analista	24
E2	J2	Desarrollador BD	135000	New York	Analista	6
E3	J3	CAD/CAM	250000	New York	Consultor	10
E3	J4	Mantenimiento	310000	Paris	Ingeniero	48
E4	J2	Desarrollador BD	135000	New York	Programador	18
null	J5	CAD/CAM	500000	Boston	null	null

# Otras operaciones derivadas de la reunión

Algunas consultas se han convertido en comunes para los usuarios que se han extendido las definiciones de la reunión para poder aplicarlas. Sus definiciones son:

- 1.- La **semi-reunión** ( $R \bowtie S$ ) de dos tablas  $R$  y  $S$  es la bolsa de tuplas  $t$  en  $R$  tal que existe al menos una tupla en  $S$  que coincide con  $t$  en todos los atributos que  $R$  y  $S$  tienen en común.
- 2.- La **antisemi-reunión** ( $R \underline{\bowtie} S$ ) es la bolsa de tuplas  $t$  en  $R$  que no coinciden con ninguna tupla de  $S$  en los atributos comunes a  $R$  y  $S$ .

# División

Dadas las relaciones

$R$  de grado  $m$  ( $R = \{A_1, \dots, A_m\}$ )

$S$  de grado  $n$  ( $S = \{B_1, \dots, B_n\}$ )

La división de  $R$  entre  $S$  (dado,  $\{B_1, \dots, B_n\} \subseteq \{A_1, \dots, A_m\}$ )

$$R \div S = \{t[\{A_1, \dots, A_m\} - \{B_1, \dots, B_n\}] \mid \forall u \in S \\ \exists v \in R (v[S] = u \wedge v[R - S] = t)\}$$

$$= \Pi_{R - S} (R) - \Pi_{R - S} ((\Pi_{R - S} (R) \times S) - R)$$

$R \div S$  resulta en una relación de grado  $(m - n)$  y consiste de todas las  $(m - n)$ -tuplas  $t$  tal que para todas las  $m$ -tuplas  $u$  en  $S$ , la tupla  $tu$  está en  $R$ .

# Ejemplo de división

## EMP\_2

<b>ENo</b>	<b>JNo</b>	<b>JName</b>	<b>Presup</b>
E1	J1	Instrumentacion	15000
E2	J1	Instrumentacion	15000
E2	J2	Desarrollador BD	13500
E3	J1	Instrumentacion	15000
E3	J2	Desarrollador BD	13500
E3	J3	CAD/CAM	25000
E3	J4	Mantenimiento	31000
E4	J2	Instrumentacion	15000
E5	J2	Instrumentacion	15000
E6	J4	Mantenimiento	31000
E7	J3	CAD/CAM	25000
E8	J3	CAD/CAM	25000

## EMP\_2 ÷ PROY

<b>ENo</b>
E3

## PROY

<b>JNo</b>	<b>JNomb</b>	<b>Presup</b>
J1	Instrumentacion	150000
J2	Desarrollador BD	135000
J3	CAD/CAM	250000
J4	Mantenimiento	310000



# Otros operadores relacionales

Además de estas operaciones sobre relaciones, existen algunos otros operadores muy empleados para la optimización de las consultas:

**Renombrado ( $\rho$ )** .- Permite referir a un objeto (columna, tabla u otro) con otro nombre en las expresiones de consulta. Se especifica con la cláusula AS en la lista de columnas del SELECT, o simplemente con el nuevo nombre a continuación del nombre de la tabla en la sentencia FROM.

**Eliminación de duplicados ( $\delta$ )**.- Esta operación convierte una bolsa en un conjunto; esto es, no existen tuplas que tengan el mismo valor para cada uno de sus atributos. Se denota por la cláusula DISTINCT en la sentencia SELECT.

**Ordenamiento ( $\tau$ )**.- Aplicado a un atributo de una relación, se puede ordenar los valores contenidos por dicho atributo de forma ascendente o descendente, sin modificar los valores contenidos en los demás atributos. Se especifica mediante la cláusula ORDER BY después del WHERE.

# Otros operadores relacionales

**Agrupamiento y agregación ( $\gamma$ ).**- Como lo indica, establece un agrupamiento en un determinado atributo, con el fin de poder implementar las operaciones de agregación, como son `sum()`, `avg()`, entre otras. Se especifica por la cláusula `GROUP BY` después del `WHERE`. De aquí se desprenden algunas diferencias:

- a) **Operadores de agregación.**- Existen básicamente cinco operadores de agregación: promedio (AVG), suma (SUM), conteo (COUNT), mínimo (MIN) y máximo (MAX), que evalúan todos los valores de las tuplas del atributo al que son aplicadas.
- b) **Agrupamiento.**- La relación resultado de una selección es agrupada en términos de los atributos mencionados en la cláusula `GROUP BY`. Las agregaciones son aplicadas en base a dicho agrupamiento.
- c) **Pertenencia.**- Una cláusula `HAVING` en una selección debe seguir a una declaración de agrupamiento, y proporciona una condición que se debe cumplir para los atributos involucrados en el agrupamiento o en la agregación.

Los subíndices que corresponden al  $\gamma$  pueden ser un atributo (agrupamiento), un operador de agregación (agregación) o una condición de pertenencia (having).

## 5.1.6 Ejemplos de Consultas algebraicas

**Emp**(Eno, ENomb, Titulo, Ciudad)

**Proy**(Pno, PNomb, Presup, Ciudad)

**Pago**(Titulo, Salario)

**Trab**(Eno, Pno, Resp, Dur)

Listar todos los nombres de empleados

$P_{\text{ENomb}}(\text{Emp})$

Listar los nombres de todos los proyectos junto con sus presupuestos

$P_{\text{PNomb, Presup}}(\text{Proy})$

# Consultas de ejemplo

Encontrar los registros de todos los empleados quienes trabajan en Toronto

$$\sigma_{\text{Ciudad}='Toronto'}(\text{Emp})$$

Encontrar todas las ciudades donde un empleado trabaje o un proyecto exista

$$\Pi_{\text{Ciudad}}(\text{Emp}) \cup \Pi_{\text{Ciudad}}(\text{Proy})$$

Encontrar todas la ciudades que tengan un proyecto pero sin empleados trabajando en el

$$\Pi_{\text{Ciudad}}(\text{Proy}) - \Pi_{\text{Ciudad}}(\text{Emp})$$

# Consultas de ejemplo

Encontrar el nombre de todos los empleados quienes trabajen en una ciudad que no tenga proyectos.

$$\Pi_{\text{ENomb}} (\text{Emp} \bowtie (\Pi_{\text{Ciudad}} (\text{Emp}) - \Pi_{\text{Ciudad}} (\text{Proy})))$$

$\text{Emp.Ciudad} = \text{X.Ciudad}$

Encontrar todas las ciudades que tengan empleados y proyectos

$$\Pi_{\text{Ciudad}} (\text{Emp}) \cap \Pi_{\text{Ciudad}} (\text{Proy})$$

Encontrar todos los empleados que trabajen en cada proyecto

$$\Pi_{\text{Eno}, \text{Jno}} (\text{Trab}) \div \Pi_{\text{Jno}} (\text{Proy})$$

# Consultas de ejemplo

Encontrar el nombre y presupuestos de todos los proyectos que empleen programadores.

$\Pi_{\text{PNomb.Presup}} (\text{Proy} \bowtie \text{Trab} \bowtie \sigma_{\text{Titulo}='Programador'} (\text{Emp}))$   
 $\text{Proy.Pno} = \text{Trab.Pno} \quad \text{Trab.Eno} = \text{Emp.Eno}$

Encontrar todos los empleados y proyectos que estén en la misma localidad

$\Pi_{\text{ENomb, PNomb}} (\text{Emp} \bowtie \text{Proy})$   
 $\text{Proy.Ciudad} = \text{Emp.Ciudad}$

# Leyes de equivalencia

- Existen dentro del álgebra relacional una serie de leyes de igualdad, que nos permiten obtener una equivalencia entre operaciones.
- Conmutativas y asociativas.*- Estas reglas definen que el orden en que se realicen las operaciones sobre las relaciones no afecta el resultado, así como la posibilidad de agrupamiento de operadores sobre relaciones. Las leyes que se emplean son:

## Conmutativas

$$R \times S = S \times R$$

$$R \bowtie S = S \bowtie R$$

$$R \cup S = S \cup R$$

$$R \cap S = S \cap R$$

$$R \underset{C}{\bowtie} S = S \underset{C}{\bowtie} R$$

## Asociativas

$$(R \times S) \times T = R \times (S \times T)$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$(R \cup S) \cup T = S \cup (R \cup T)$$

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

# Leyes que involucran selección (1)

La operación de selección trata de reducir el tamaño de una relación, por lo que se usan las siguientes reglas para “mover” las selecciones en la expresión sin modificar las restantes operaciones involucradas.

Cuando una condición involucra varios conectores lógicos (OR y AND), es mejor estrategia hacer una descomposición de sus partes constitutivas, usando las siguientes reglas:

$$\sigma_{c_1 \wedge c_2} (R) = \sigma_{c_1} (\sigma_{c_2} (R))$$

$$\sigma_{c_1 \vee c_2} (R) = (\sigma_{c_1} (R)) \cup (\sigma_{c_2} (R))$$

$$\sigma_{c_1} (\sigma_{c_2} (R)) = \sigma_{c_2} (\sigma_{c_1} (R))$$



# Leyes que involucran selección (2)

La siguiente familia de leyes involucran la selección permitiendo “empujarlas” a través de las operaciones binarias producto, unión, intersección, diferencia y reunión. Entonces:

$$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$$

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

$$\sigma_C(R \times S) = \sigma_C(R) \times S \text{ sólo si } C \text{ tiene atributos de } S \quad \sigma_C(R \times S) = R \times \sigma_C(S)$$

$$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S \quad \circ \quad \sigma_C(R \bowtie S) = R \bowtie \sigma_C(S)$$

$$\sigma_C(R \cap S) = \sigma_C(R) \cap S \quad \circ \quad \sigma_C(R \cap S) = R \cap \sigma_C(S)$$

si C tiene todos los atributos de R y S, entonces:

$$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie \sigma_C(S)$$

Las leyes en “casos externos” son:

una selección en una relación vacía es vacío.

Si C es una condición de siempre verdadero, entonces  $\sigma_C(R) = R$

Si R es vacío, entonces  $R \cup S = S$

# Otras equivalencias

## Conmutación de selección con proyección

Si la condición  $C$  involucra los atributos  $A_1, A_2, \dots, A_n$  en la lista de proyección, entonces:

$$\pi_{A_1, A_2, \dots, A_n} (\sigma_C (R)) = \sigma_C (\pi_{A_1, A_2, \dots, A_n} (R))$$

## Conmutación de proyección con reunión o producto cartesiano.

Si la condición de unión  $C$  involucra sólo atributos de  $L$ , entonces:

$$\pi_L (R \underset{C}{\bowtie} S) = (\pi_{A_1, A_2, \dots, A_n} (R)) \underset{C}{\bowtie} (\pi_{B_1, B_2, \dots, B_m} (S))$$

## Leyes de De Morgan

$$\neg (P \wedge Q) = (\neg P) \vee (\neg Q)$$

$$\neg (P \vee Q) = (\neg P) \wedge (\neg Q)$$

Por definición:

$$R \underset{C}{\bowtie} S = \sigma_C (R \times S)$$

$$R * S = \pi_L (\sigma_C (R \times S))$$

# Leyes que involucren eliminación de duplicados

El operador  $\delta$  puede ser empujado por la mayoría, pero no por todos los operadores. En general en cada paso se reduce el tamaño de la relación. Se tienen la siguientes circunstancias:

$$\delta(R) = R$$

si  $R$  no tiene duplicados, lo cual se puede deber a que

- a) la relación tenga definida una llave primaria o
- b) la relación sea el resultado de una operación de agrupamiento.

Las leyes involucradas son:

$$\delta(R \times S) = \delta(R) \times \delta(S)$$

$$\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$$

$$\delta(\sigma_c(R)) = \sigma_c(\delta(R))$$

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

El operador  $\delta$  no puede ser movido a través de los operadores  $\cup$ ,  $-$  o  $\pi$ .

## Leyes que involucran agrupamiento y agregación.

En este operador hay una dependencia hacia el tipo de agregado que se requiere hacer, por lo tanto no hay una serie de reglas aplicables; sin embargo, se puede escribir

$$\delta (\gamma_L (R) ) = \gamma_L (R)$$

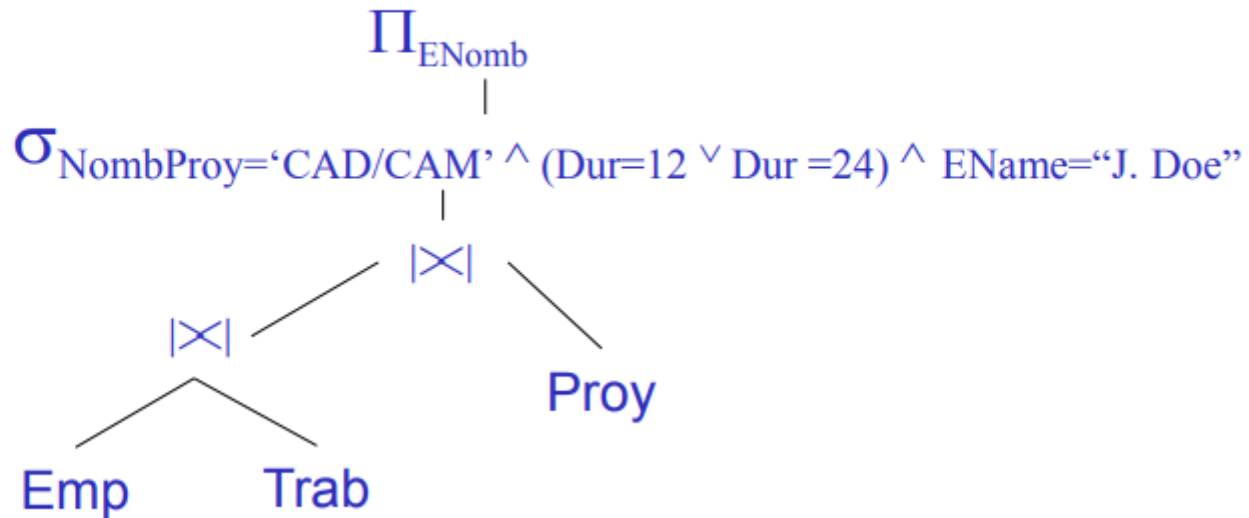
$$\gamma_L (R) = \gamma_L (\pi_M (R) )$$

si  $M$  es una lista de todos los atributos de  $R$  que son mencionados en  $L$ .

# Representación como árboles

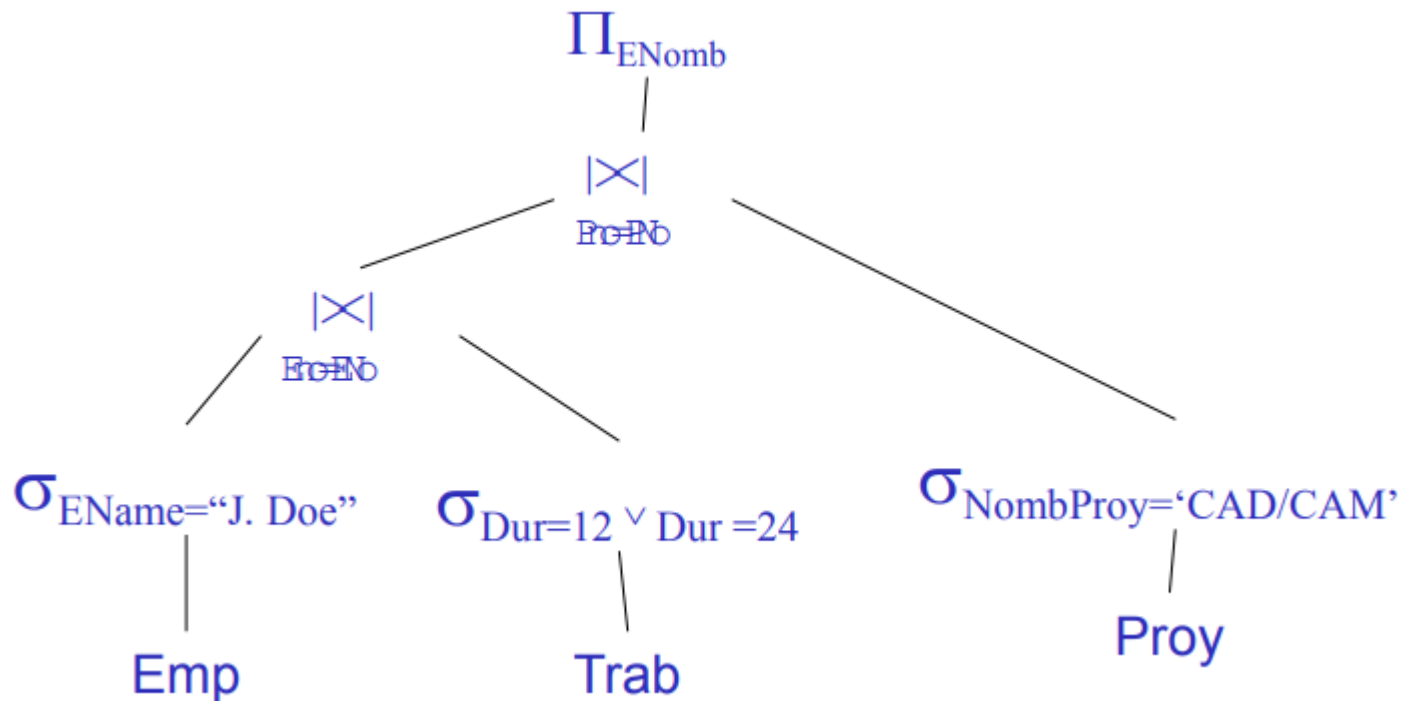
Una forma de visualizar adecuadamente las operaciones de álgebra relacional en una consulta, es mediante la representación en árbol, en donde las tablas representan los nodos y los operadores se van ejecutando en orden ascendente:

$\Pi_{ENomb}(\sigma_{NombProy='CAD/CAM' \wedge (Dur=12 \vee Dur=24) \wedge EName='J. Doe'}(Emp \bowtie Proj \bowtie Trab))$



# Transformaciones del árbol mediante leyes del álgebra relacional

La equivalencia de las leyes del álgebra relacional se muestra en el árbol correspondiente:



## 5.2 SQL

- El SQL surge del SEQUEL (Structured English Query Language), un prototipo de lenguaje relacional del proyecto System R de IBM (1974-95). Posteriormente se denomina SEQUEL II o SQL.
- En 1979 surge el primer SGBDR basado en SQL, de Oracle. Aparecen entonces varios sistemas relacionales como el SQL/DS, DB2, DG/SQL, SYBASE, Informix, RDB, etc.
- En 1982 el Comité de Base de Datos X3H2 de ANSI presenta un lenguaje relacional estructurado basado en SQL, y en 1986 se aprueba para ser el SQL/ANSI, bajo la norma ISO.
- En 1989 se publica una nueva versión de ISO, que añade cierta integridad referencial, permite definir la opción de modificación y borrado restringido y no proporciona cambios en cascada. En ese mismo año, ANSI define un estándar para el SQL incrustado.

---

## 5.2.1 Introducción

- Grupos de trabajo de ANSI y de ISO elaboraron una nueva versión de SQL, el SQL2, y ha sido aprobado como proyecto de norma internacional en diciembre de 1991. Se convirtió en norma internacional en 1992.
  - Se han elaborado nuevas propuestas para extender el SQL (SQL3) con mayor capacidad semántica y ciertos principios del paradigma de orientación a objetos. Se pretende incluir en el lenguaje tipos de datos definidos por el usuario, disparadores, jerarquías de generalización, especialización, llamadas a procedimientos externos, etc.
-



---

# Propósitos

- Simplicidad de los comandos (que sean de un modo natural)
  - Sean pocas sentencias y comandos de aprender
  - Permita construir consultas más complejas y con más capacidad en base a las sentencias
  - El lenguaje pueda ser aplicado por cualquier usuario sin importar el sistema que este accediendo
-

---

# Características

- El SQL esta basado en las teorías del modelo relacional.
  - Es declarativo: especifica las propiedades que deben estar en el resultado, no cómo obtenerlo
  - El lenguaje consiste en declaraciones para insertar, actualizar, borrar, consultar y proteger los datos, basados en un Lenguaje de Definición de Datos (DDL) y en un Lenguaje de Manipulación de Datos (DML)
  - SQL puede ser usado de dos maneras: interpretado, en el cual una declaración es alimentada a una terminal y es ejecutada inmediatamente; o incrustado, en donde las declaraciones son añadidas en un programa escrito a un lenguaje de programación procedural.
-

## 5.2.2 Ambiente de SQL

- De acuerdo al modelo relacional, el SQL consta de un lenguaje de Manipulación de Datos (DML) y de Definición de Datos (DDL), con declaraciones y sentencias que permiten el uso de los datos. Algunas de estas son:

Manipulación de datos (DML)	SELECT	recuperación de datos
	INSERT	altas de datos
	DELETE	bajas de datos
	UPDATE	cambios de datos
Definición de datos (DDL)	CREATE	creación de objetos*
	ALTER	modificación de objetos
	DROP	borrado de objetos
Control de acceso	GRANT	otorgamiento de privilegios
	REVOKE	eliminación de privilegios
Control de transacciones	COMMIT	finaliza la transacción actual
	ROLLBACK	aborta la transacción actual

\* Los objetos administrados son TABLE (tabla), VIEW (vista), INDEX (índice)

- Esta sentencias anteriores forman parte del estándar ANSI/ISO, mientras que otros SDMB en el mercado pueden incorporar algunas otras mejoras. Cabe destacar que no se proporciona una sentencia que construya una base de datos, pero que la mayoría de los SBDM deben tener algún sistema para crearlas.

---

## 5.2.3 Definición de una base de datos en SQL

- En el SQL92, las relaciones y otros objetos de la base de datos existen en un ambiente.
  - Cada ambiente contiene uno o más catálogos, y cada catálogo consiste de un conjunto de esquemas.
  - El *esquema* es una colección nombrada de objetos relacionados de la base de datos.
  - Los objetos en un esquema pueden ser tablas, vistas, dominios, aserciones, índices, usuarios, entre otros. Todos tienen el mismo dueño.
-

# Esquemas

- Crear un esquema

```
CREATE SCHEMA [Nombre_esquema | AUTHORIZATION Nombre_usuario]
```

- Eliminar un esquema

```
DROP SCHEMA Nombre_esquema [RESTRICT | CASCADE]
```

- Con **RESTRICT** (omisión), el esquema debe estar vacío o la operación fallará.
- Con **CASCADE**, la operación eliminará todos los objetos asociados con el esquema en el orden como estuvieron definidos. Si alguna de estas operaciones falla, **DROP SCHEMA** fallará.

# Tipos Definidos por el usuario

- Creación de un DOMINIO

```
CREATE DOMAIN Nombre_dominio AS tipo_prmitivo[DEFAULT  
valor]  
[CHECK (condición)]
```

- Eliminación de un DOMINIO

```
DROP DOMAIN Nombre_domino
```

- Ejemplo

```
CREATE DOMAIN genero AS CHAR (1) CHECK (VALUE IN ( 'F' ,  
'M' ) ) ;
```

- Generalmente es útil sólo para la modificación sencilla de la especificación de tipos.
- El valor puede ser establecido por defecto mediante **DEFAULT**

# Tablas

- Crear una tabla

Especifica un nuevo esquema de relación

Forma general:

```
CREATE TABLE Nombre_tabla  
(Columna1 Tipo [DEFAULT (valor | NULL)] [restricciones_columna],  
Columna2 Tipo [DEFAULT (valor | NULL)] [restricciones_columna],  
...  
ColumnaN Tipo [DEFAULT (valor | NULL)] [restricciones_columna],  
[restricciones_tabla])
```

- Eliminar una tabla

```
DROP TABLE Nombre_tabla[RESTRICT | CASCADE]
```

Con **RESTRICT**, si otros objetos dependen de la existencia o de la continuación de la existencia de la tabla, no se permite la ejecución.

Con **CASCADE**, se eliminan todos los objetos dependientes (y los dependientes de estos objetos)

# Tipos de Datos Permitidos

Data type	Description
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n) or BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s.
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation.
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
INTERVAL	Composed of a number of integer fields, representing a period of time
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data



# Tipos de datos de Oracle

<b>CHAR(n)</b>	Cadena de caracteres de longitud fija, tiene un tamaño n bytes. Si no se especifica n la ORACLE le da un tamaño de 255 bytes. El tamaño máximo es 2000 bytes y el mínimo 1 byte. El tamaño máximo en PL/SQL es 32767 bytes y el mínimo 1 byte. CHARACTER es sinónimo de CHAR.
<b>VARCHAR2(n)</b>	Cadena de caracteres de longitud variable, tiene un tamaño máximo de n bytes. Es obligatorio especificar el tamaño. El tamaño máximo es 4000 bytes y el mínimo 1 byte. El tamaño máximo en PL/SQL es 32767 bytes y el mínimo 1 byte. STRING y VARCHAR son sinónimos de VARCHAR2.
<b>NUMBER(p,s)</b>	Número de p dígitos de los cuales s son decimales. No es obligatorio especificar el tamaño. El tamaño de p va de 1 a 38 y el s desde -84 a 127. El tamaño en PL/SQL 1E-130 .. 10E125. Sinónimos: números de coma fija: DEC,DECIMAL,NUMERIC enteros: INTEGER (sinónimo de NUMBER(38)), INT, SMALLINT coma flotante: DOUBLE PRECISION FLOAT REAL.
<b>DATE</b>	Fecha válida. Desde el 1 de enero del 4712 AC hasta el 31 de diciembre del 9999 DC. (en Oracle7 = 4712 DC)
<b>LONG</b>	Cadena de caracteres de longitud variable. Es una versión más grande de VARCHAR2. El tamaño máximo en BD es 2 Gigabytes.
<b>CLOB</b>	Cadena de caracteres de longitud variable. Es una versión más grande de VARCHAR2. El tamaño máximo en BD es 4 Gigabytes. Es recomendable usar CLOB o BLOB en lugar de LONG.
<b>BLOB</b>	Objeto binario de longitud variable. Es una versión más grande de RAW. El tamaño máximo en BD es 4 Gigabytes.
<b>BFILE</b>	Apuntador a un archivo en disco. El tamaño máximo en BD es 4 Gigabytes
<b>TIMESTAMP (f)</b>	Es una fecha que contiene fracciones de segundo. Con f se define el número de dígitos en la fracción de segundo. Así, f puede valer desde 0 hasta 9, el valor por defecto es 6.
<b>INTERVAL YEAR (y) TO MONTH</b>	Periodo de tiempo definido en años y meses donde y es el número de dígitos del año. Puede valer de 0 a 9. (por defecto es 2)
<b>INTERVAL DAY (d) TO SECOND (f)</b>	Periodo de tiempo definido en días, horas, minutos y segundos. d es el máximo número de dígitos en el día, f es el máximo número de dígitos en el campo de segundos. d va de 0 a 9. (por defecto es 2), f va de 0 a 9. (por defecto es 6)
<b>ROWID</b>	Cadena hexadecimal que representa de forma única una fila en una tabla (pero no única en cualquier tabla).
<b>UROWID</b>	Cadena hexadecimal que representa de forma única una fila ORDENADA en una tabla (pero no única en cualquier tabla).
<b>RAW(n)</b>	Objeto binario de longitud variable. Es obligatorio especificar el tamaño. El tamaño máximo es 2000 bytes y el mínimo 1 byte. El tamaño máximo en PL/SQL es 32767 bytes y el mínimo 1 byte.
<b>LONG RAW</b>	Objeto binario de longitud variable. El tamaño máximo es 2 Gigabytes. El tamaño máximo en PL/SQL es 32767 bytes y el mínimo 1 byte.

# Restricciones de Atributo

- `restricciones_columna` puede ser:

**NOT NULL** | -Especifica que un atributo no puede contener valores nulos

[**CONSTRAINT** Nombre\_restriccion] -Nombre de la restricción

**UNIQUE** -Especifica que un atributo no puede contener duplicados

| **PRIMARY KEY** -Designa a la columna como la llave primaria de la tabla

| **CHECK** (condicion) -Verifica que el valor cumpla la condición especificada

| **REFERENCES** Nombre\_tabla[(Nombre\_columna)] - Designa la columna como la llave foránea en las restricciones referenciales

[**ON {DELETE | UPDATE} {CASCADE | SET NULL | NO ACTION | SET DEFAULT}**] -Cuando un renglón sea eliminado, también lo será en la tabla referenciada, tomando las acciones correspondientes

# Restricciones de Tabla

- restricciones\_tabla puede ser:

[**CONSTRAINT** Nombre\_restriccion]

**UNIQUE** (Nombre\_columna[, Nombre\_columna...]) -Especifica que un atributo no puede contener duplicados

| **PRIMARY KEY** (Nombre\_columna[, Nombre\_columna...]) -Designa la(s) columna(s) como la llave primaria de la tabla

| **CHECK** (condicion) -Verifica que el valor cumpla la condición especificada

| **FOREIGN KEY** (Nombre\_columna[, Nombre\_columna...])

**REFERENCES** Nombre\_tabla[(Nombre\_columna[, Nombre\_columna...])] - Designa la(s) columna(s) como la llave foránea en las restricciones referenciales

[**ON {DELETE | UPDATE} {CASCADE | SET NULL | NO ACTION | SET DEFAULT}**] -Cuando un renglón sea eliminado, también lo será en la tabla referenciada, tomando las acciones correspondientes

# Ejemplo

- Diseño

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Definición de **Proyecto**

```
CREATE TABLE Proyecto
(NoProy          CHAR (3) ,
NombProy        VARCHAR (20) ,
Presupuesto     DECIMAL (10,2) ,
Ciudad          CHAR (9) ) ;
```

# Restricciones Referenciales

- Acción disparada referencialmente
  - Integridad referencial: La llave de una relación aparece como un atributo (llave foránea) de otra relación.

Ejemplo:

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- La eliminación o actualización de la tupla de la llave primaria requiere una acción en la tupla de la llave foránea. Se especifica la restricción en *delete* o *update*.
- Como manejarlas?
  - rechazarlas
  - en cascada: (en *delete*) automáticamente se remueve las llaves foráneas si una llave referenciada es removida o (en *update*) se cambia el valor de la llave foránea al nuevo valor de la llave referenciada.
  - establecer nulo
  - establecer el valor predefinido

# Ejemplo

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

```
CREATE TABLE Proyecto
(NoProy      CHAR (3) PRIMARY KEY,
NombProy     VARCHAR (20) UNIQUE CONSTRAINT
    const_uniq_Project NOT NULL,
Presupuesto  DECIMAL (10,2) DEFAULT 0.00 CHECK(Presupuesto
    >= 0.0),
Ciudad       CHAR (3) REFERENCES Ciudades(Cod) ON UPDATE
    CASCADE ON DELETE NO ACTION);
```

# Ejemplo

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

```
CREATE TABLE Trabaja
```

```
(NoEmp CHAR (3),
```

```
NoProy CHAR (3),
```

```
Resp CHAR (15),
```

```
Dur INT,
```

```
PRIMARY KEY (NoEmp, NoProy ),
```

```
FOREIGN KEY (NoEmp) REFERENCES Emp (NoEmp)
```

```
ON DELETE SET NULL ON UPDATE CASCADE,
```

```
FOREIGN KEY (NoProy ) REFERENCES Proyecto (NoProy )
```

```
CHECK (NoProy < 'P5' OR Dur <18));
```

# Modificación de una tabla

- La modificación del esquema de una tabla puede ser como:
  - Agregar, eliminar o modificar una columna de una tabla
  - Agregar o eliminar una restricción de tabla
  - Agregar o eliminar una restricción de columna
- Forma general:

```
ALTER TABLE table_name
```

```
[ADD {(col_name type | col_constraint | tab_constraint)}]
```

```
| [DROP {col_name, [col_name, ...] } ]
```

```
| [DROP CONSTRAINT constraint_name]
```

```
| [DROP PRIMARY KEY]
```

```
| [MODIFY {col_name type}]
```

- MySQL, Oracle

```
| [ALTER COLUMN {col_name type}]
```

- SQLServer, Access

```
| [ENABLE | DISABLE constraint_name]
```

- Solo en Oracle



# Ejemplo

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Agregando la columna *Direccion* a *Emp*

```
ALTER TABLE Emp
```

```
ADD COLUMN (Direccion VARCHAR(30));
```

- Modificando la definición de *NombProy* en *Proyecto*

```
ALTER TABLE Proyecto
```

```
MODIFY (NombProy VARCHAR(35) NOT NULL DEFAULT 'Finanzas');
```

- Eliminando la restricción en *NombProy* de *Proyecto*

```
ALTER TABLE Proyecto
```

```
DROP CONSTRAINT const_uniq_Proyecto
```

---

## 5.2.4 Inserción de renglones

- Forma general:

```
INSERT INTO Nombre_tabla[lista_columnas]  
VALUES (lista_valores)
```

- Insertando el resultado de una consulta

```
INSERT INTO Nombre_tabla  
(consulta)
```

- Los valores de la lista deben coincidir en número, posición y tipo de dato y deberán estar separadas por comas. Aquellas columnas que no tengan valor deberán ser establecidas a NULL. Los tipos de datos numéricos son los únicos que no deben estar entre comillas sencillas.
-

# Ejemplo

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Insertar un nuevo registro para el empleado *John Smith* quién es asignado con el número *E24*, es *programador* y trabaja en *Waterloo*.

```
INSERT INTO Emp (NoEmp, NombEmp, Titulo, Ciudad)
```

```
VALUES ('E24', 'John Smith', 'Programador', 'Waterloo')
```

## 5.2.4 Eliminación de renglones

- Forma general:

```
DELETE FROM Nomb_tabla  
[WHERE  condicion]
```

- Si la condición es omitida, se eliminan todos los renglones de la tabla.
- Ejemplo: eliminar todos los empleados que sean *Programadores*

```
DELETE FROM Emp  
WHERE titulo = 'Programador'
```

## 5.2.4 Actualización de renglones

- Forma general:

```
UPDATE Nombre_tabla
```

```
SET Nombre_columna_1 = valor_1,
```

```
    [, Nombre_columna_2 = valor_2 ...]
```

```
[WHERE condicion]
```

- Si se omite la cláusula *WHERE*, se actualizan todas las tuplas en la columna especificada.
- Los valores deben ser compatibles con los tipos de datos de las columnas correspondientes

# Ejemplo

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Aumentar en un 5% el presupuesto de los proyectos que se localizan en *Edmonton*.

**UPDATE** Proyecto

**SET** Presupuesto = Presupuesto \* 1.05

**WHERE** Ciudad = 'Edmonton'

## 5.2.5 Procesamiento en tablas

- Forma general:

**SELECT** [**DISTINCT** | **ALL**] { \* | - menciona las columnas a aparecer en el resultado

lista\_columnas\_o\_expresion[**AS** nuevo\_nombre] [, ...] }

**FROM** nombre\_tabla[alias] [, ...] - menciona las tablas a ser usadas

[**WHERE** condicion] - filtra los renglones

[**GROUP BY** lista\_columnas] - forma grupos de renglones con base a los valores en las columnas

[**HAVING** condicion] - filtra grupos con base en una condición

[**ORDER BY** lista\_columnas] - especifica el orden de la salida

- La cláusula *DISTINCT* elimina valores duplicados en el resultado
- El asterisco (\*) equivale a obtener todas las columnas en el resultado
- Se pueden establecer alias en los nombres de las columnas mediante la cláusula *AS*

# Tipo de predicados en la condición

- Predicados simples:
  - *Expresión*  $\theta$  *Valor* donde *expresión* puede ser un atributo o una expresión aritmética que involucra atributos  $\theta = \{<, >, =, <=, >=, <>\}$  y *Valor* puede ser de uno de los tipos de datos

Ejemplo:

Nombre = 'J. Doe'

(Edad + 30)  $\geq$  65

- Predicados compuestos
  - Predicados simples combinados con conectivos lógicos AND, OR, NOT



# Operadores lógicos

Igual	=
Desigual	!= ó <>
mayor que	>
menor que	<
mayor o igual	>=
menor o igual	<=
igual a algún miembro	IN (lista)
valor comprendido entre dos valores	BETWEEN x AND y
prueba de correspondencia	LIKE (% para un patrón y _ para un carácter)
verifica si un valor es nulo	IS NULL
Negación	NOT
o lógico	OR
y lógico	AND

# Manipulación de nulos

- Los valores en los atributos pueden ser nulos.
- NULL no es una constante, ni puede ser usado como un operando.
- Si  $x$  es NULL
  - $x \neq \text{constante}$  es **NULL**
  - $x \neq y$  es **NULL**
- Comparando un valor NULL con cualquier otro valor regresa UNKNOWN (lógica del tres-valores).
- En SQL, UNKNOWN a veces se trata como falso (SELECT) y a veces como verdadero (en restricciones).

a	b	a AND b	a OR b	NOT b
True	True	True	True	False
True	False	False	True	True
True	Null	Null	True	Null
False	False	False	False	
False	Null	False	Null	
Null	Null	Null	Null	

# Ejemplo de Consultas Simples

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Lista los nombres de todos los empleados.

```
SELECT    NombEmp
FROM      Emp
```

- Lista los nombres de los proyectos junto con sus presupuestos.

```
SELECT    NombProy, Presupuesto
FROM      Proyecto
```

- Encontrar todas las ciudades donde por lo menos exista un proyecto.

```
SELECT    DISTINCT Ciudad
FROM      Proyecto
```

# Consultas con Predicados

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar todas las profesiones que ganen más de \$50,000

```
SELECT    Titulo
FROM      Pago
WHERE      Salario > 50000
```

- Encontrar los empleados que trabajen en un proyecto como gerentes por más de 17 meses.

```
SELECT    NoEmp
FROM      Trabaja
WHERE      Dur > 17 AND Resp = 'Gerente'
```

# Ordenando el Resultado

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar los nombres y presupuestos de todos los proyectos con presupuesto mayor que \$250,000 y ordenar el resultado ascendentemente por los valores del presupuesto.

```
SELECT    NombProy, Presupuesto
FROM      Proyecto
WHERE     Presupuesto > 250000
ORDER BY  Presupuesto
```

- El valor por defecto es en orden ascendente, pero se puede especificar el orden descendente con la cláusula DESC.

# Producto cartesiano

- El producto cartesiano se obtiene especificando en la cláusula FROM las tablas empleadas que participarán en el producto

- Forma general:

```
SELECT A1, ...An, B1, ...Bn, ...
```

```
FROM R, S, .....
```

- En el estandar SQL92 es posible especificar un producto cartesiano con la siguiente construcción:

```
SELECT [DISTICT | ALL] {* | lista_columnas}
```

```
FROM nombre_tabla_1 CROSS JOIN nombre_tabla_2
```

- Ejemplo:

- Obtener el producto cartesiano de las tablas de Emp y Pago

```
SELECT *
```

```
FROM Emp, Pago
```

ó

```
SELECT *
```

```
FROM Emp CROSS JOIN Pago
```

# Reuniones

- Para efectuar una reunión, se incluyen las tablas a reunir después de la cláusula FROM, separadas por comas.
- La cláusula WHERE debe incluir condiciones de igualdad en los atributos comunes en la relaciones, y aplicados con el operador lógico AND.

- Forma general:

```
SELECT A1, ...An, B1, ...Bn, C1, ...Cn, ...  
FROM R, S, T.....  
WHERE R.Ai = S.Bi AND S.Bj = T.Cj, ....  
AND condicion
```

- Se pueden especificar alias en los nombres de las tablas para evitar ambigüedades
- Ejemplo:
  - Para seleccionar los nombres de los empleados que estén en el proyecto P3

```
SELECT NombEmp  
FROM Emp e, Trabaja t  
WHERE e.NoEmp = t.NoEmp  
AND NoProy = 'P3'
```

# Reuniones

- El estándar SQL92 proporciona una forma alternativa de especificar las reuniones
- Forma general:

```
SELECT A1, ...An, B1,...Bn,...  
FROM R [INNER]JOIN S ON R.Ai = S.Bi      ó  
FROM R [INNER]JOIN S USING i
```

- En cada caso, FROM reemplaza al original FROM y WHERE. Sin embargo, el primer caso produce una tabla con dos columnas idénticas en el campo de la relación, mientras que el último muestra una vez el atributo en común
- Ejemplo:

```
SELECT NombEmp  
FROM Emp e JOIN Trabaja t USING NoEmp  
WHERE NoEmp = 'P3'
```



# Multireuniones

- Para especificar una reunión que involucre más de dos relaciones se puede usar:

```
SELECT A1, ...An, B1, ...Bn, ...  
FROM (R JOIN S USING i) AS alias JOIN T USING j
```

- Ejemplo:
  - Para seleccionar los nombres de los empleados y de los proyectos que estén en el proyecto P3

```
SELECT NombEmp, NombProy  
  
FROM (Emp e JOIN Trabaja t USING NoEmp) AS emp_proy JOIN  
Proyecto p USING NoProy WHERE NoProy = 'P3'
```

# Reuniones Externas

- Asegura que las tuplas de una o ambas relaciones que no satisfacen la condición de unión todavía aparece en el resultado final con los valores de los otros atributos de la relación establecidos a NULL.

- Forma general:

```
R [NATURAL] JOIN S [ON <condicion>]
```

- Para proporcionar las reuniones externas se usa:

```
R [NATURAL] FULL OUTER JOIN S [ON <condicion>]
```

```
R [NATURAL] LEFT OUTER JOIN S [ON <condicion>]
```

```
R [NATURAL] RIGHT OUTER JOIN S [ON <condicion>]
```

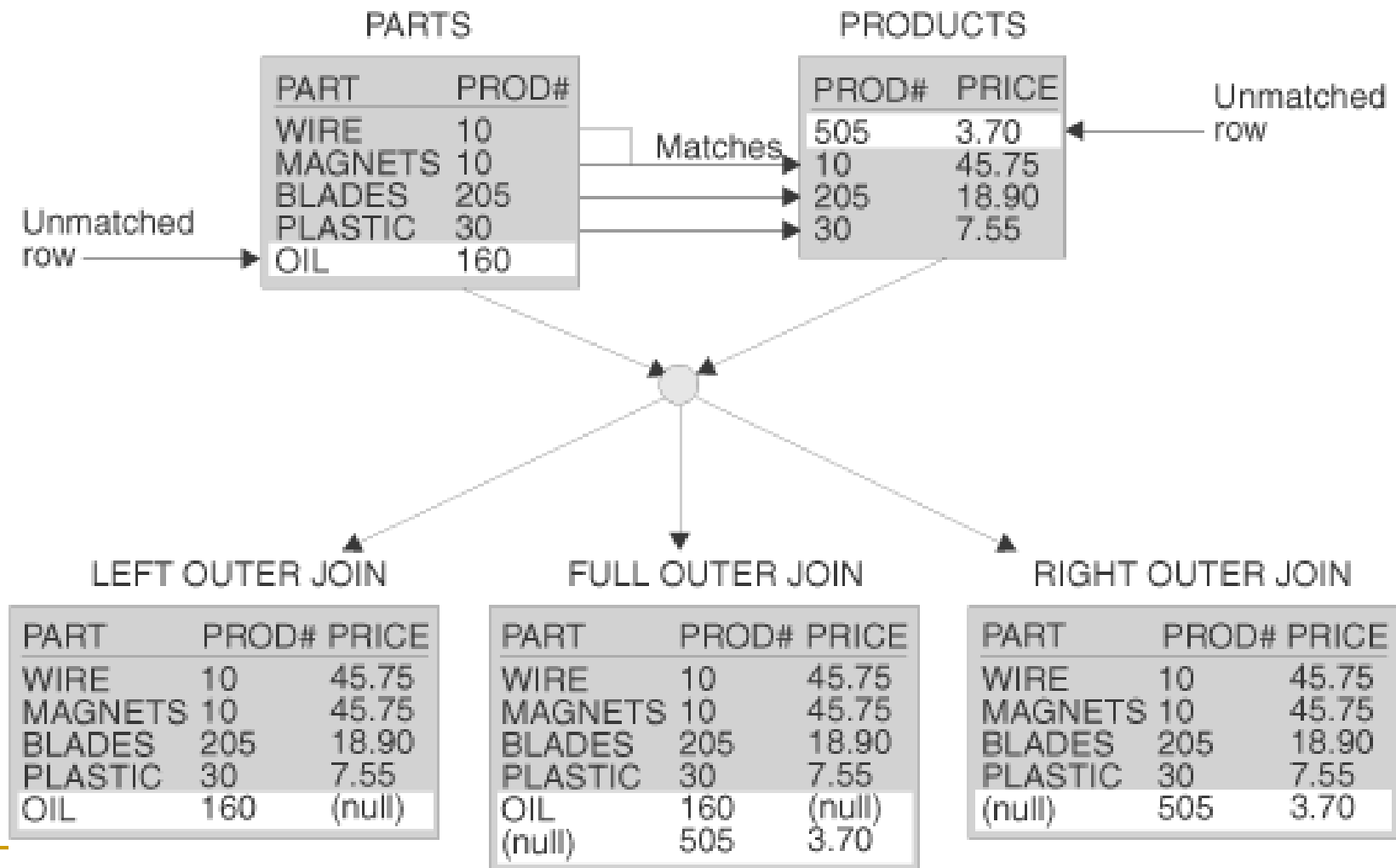
- Ejemplo: Encontrar los empleados y los proyectos en los que trabajan.

```
SELECT e.*, t.*
```

```
FROM Emp e LEFT OUTER JOIN Trabaja t ON e.NoEmp = t.NoEmp
```

- Produce el listado de los empleados aún cuando no estén en ningún proyecto por el momento.

# Outer Joins



# Consultas Sobre Múltiples Relaciones

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Listar el nombre y títulos de todos los empleados que trabajan en un proyecto por más de 17 meses.

```
SELECT    NombEmp, Titulo
FROM      Emp, Trabaja
WHERE     Dur > 17
AND       Emp.NoEmp = Trabaja.NoEmp
```

- Encontrar el nombre y el título de todos los empleados que trabajan en un proyecto localizado en Waterloo.

```
SELECT    NombEmp, Titulo
FROM      Emp E, Trabaja T, Proyecto P
WHERE     P.Ciudad = 'Waterloo'
AND       E.NoEmp = T.NoEmp
AND       T.NoProy = P.NoProy
```

# Consultas Sobre Múltiples Relaciones

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Lista los pares de empleados y proyectos que están en la misma localidad

```
SELECT    NoEmp, NoProy
FROM      Emp, Proyecto
WHERE     Emp.Ciudad = Proyecto.Ciudad
```

- Listar el par de nombres de los empleados que están localizados en la misma ciudad.

```
SELECT    E1.NombEmp, E2.NombEmp
FROM      Emp E1, Emp E2
WHERE     E1.Ciudad = E2.Ciudad
```

# Funciones de Agregación

- El estandar SQL permite especificar una función que calcule un valor numérico de una relación dada, basado en los valores de una o varias columnas o expresiones matemáticas.
  - La función se aplica normalmente a un atributo y regresa un solo valor. Sólo se pueden aplicar en la lista del SELECT y con una clausula HAVING.
  - Las operaciones definidas son el conteo (COUNT), la suma (SUM), el máximo (MAX), el mínimo (MIN) y el promedio (AVG).
  - COUNT, MIN y MAX aplican a columnas con tipos numéricos y no numéricos; SUM y AVG sólo a numéricos.
  - Si la lista del SELECT incluye una función de agregación, debe existir una clausula GROUP BY con referencia a la columna de la función de agregación.
- Forma general:

```
SELECT    funcion_agregacion ([DISTINCT] A1) , . . . ,  
          funcion_agregacion ([DISTINCT] Aj)  
FROM      R1 , . . . , Rm  
WHERE     condicion
```

# Funciones de agregación

```
SELECT SUM(advance_amount)  
FROM orders;
```

orders

ADVANCE_AMOUNT
2000
400
100
700
600
400
1800
1200
600
700
500
800
1000

SUM(ADVANCE_AMOUNT)
19450

result

# Ejemplos de consultas de agregación

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar el presupuesto total de los proyectos en Waterloo.

```
SELECT    SUM (Presupuesto)
FROM      Proyecto
WHERE     Ciudad = 'Waterloo'
```

- Encontrar el número de ciudades donde hay un proyecto en que el empleado E4 trabaje.

```
SELECT    COUNT (DISTINCT Ciudad)
FROM      Proyecto, Trabaja
WHERE     Proyecto.NoProy = Trabaja.NoProy
AND       Trabaja.NoEmp = 'E4'
```



# Consultas de Agrupamiento

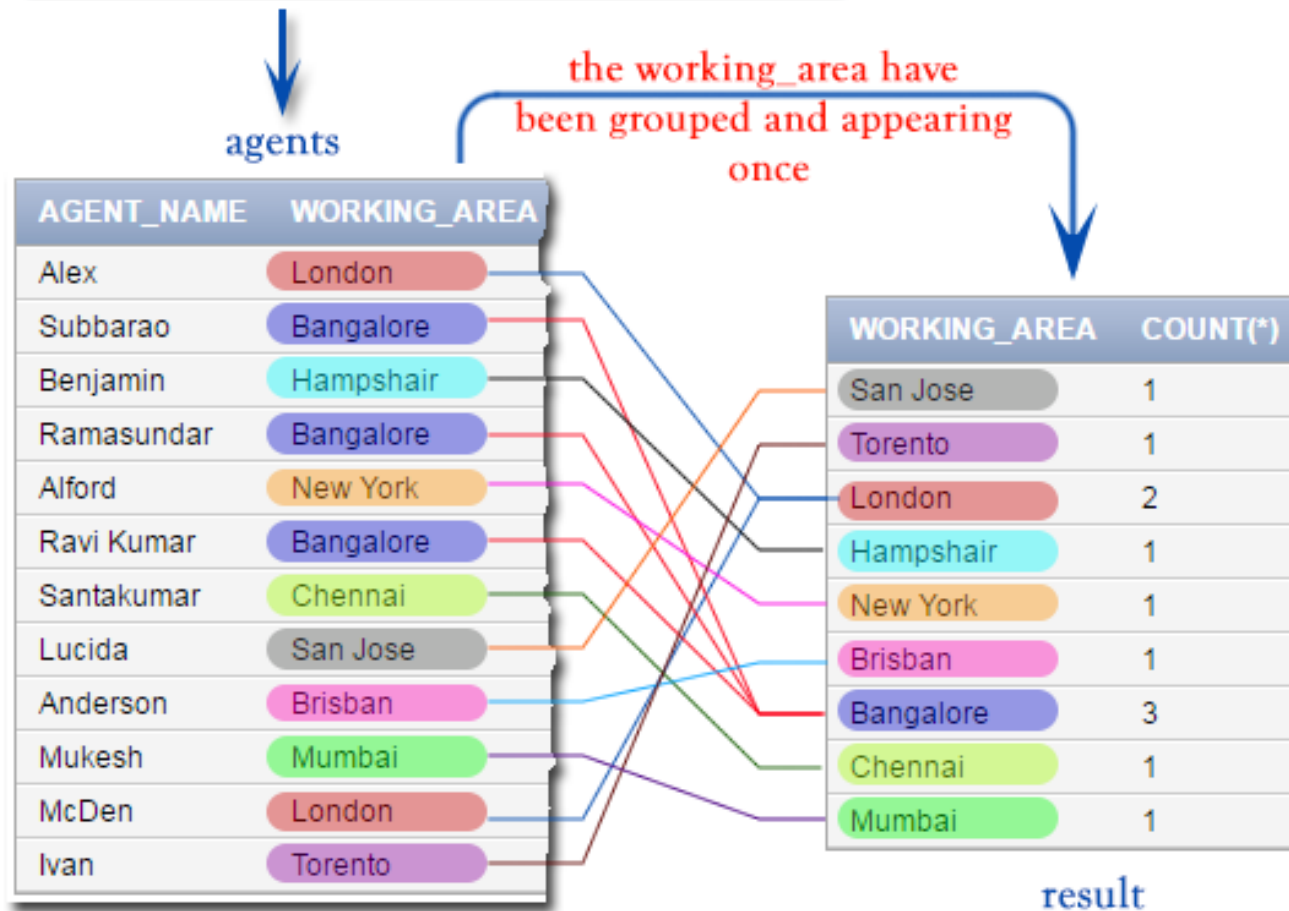
- Agrupa los resultados de acuerdo a un conjunto de atributos.
- Forma general:

```
SELECT    A1, . . . , An  
FROM      R1, . . . , Rm  
WHERE     condicion  
GROUP BY Aj, . . . , Ak
```

- Reglas:
  - Todos los atributos en la cláusula **SELECT** que no están involucrados en una operación de agregación tienen que ser incluidos en la cláusula **GROUP BY**.
  - Si se especifica **WHERE** con **GROUP BY**, se aplica primero el **WHERE**, y entonces los grupos son formados de los renglones que satisfacen el predicado.
  - El estándar SQL considera dos valores nulos como iguales para propósitos de agrupamiento.

# Group by

```
SELECT working_area, COUNT(*)  
FROM agents  
GROUP BY working_area;
```



# Predicados en agrupamiento

- Agrupa los resultados según un juego de atributos si ellos satisfacen una cierta condición.
  - Es similar al WHERE, solo que filtra valores dentro de los grupos formados.

- Forma general:

```
SELECT    A1, . . . , An  
FROM      R1, . . . , Rm  
WHERE      condicion  
GROUP BY  Aj, . . . , Ak  
HAVING     condicion
```

- Reglas:
  - La condición debe tener sólo un valor por grupo.
  - Las columnas en HAVING también deben aparecer en la lista del GROUP BY o estar contenidas en la función de agregación.

# Having / Order by

```
SELECT commission, COUNT (*)
FROM agents
GROUP BY commission
HAVING COUNT (*) > 3;
```

agents

AGENT_NAME	COMMISSION
Alex	.13
Subbarao	.14
Benjamin	.11
Ramasundar	.15
Alford	.12
Ravi Kumar	.15
Santakumar	.14
Lucida	.12
Anderson	.13
Mukesh	.11
McDen	.15
Ivan	.15

GROUP BY commission

COMMISSION	COUNT(*)
.15	4
.11	2
.14	2
.13	2
.12	2

HAVING COUNT (\*) > 3;

COMMISSION	COUNT(*)
.15	4
.11	2
.14	2
.13	2
.12	2

COMMISSION	COUNT(*)
.15	4

```
SELECT cust_city, cust_country,
MAX(outstanding_amt)
FROM customer
GROUP BY cust_country, cust_city
ORDER BY cust_city;
```

CUST_CITY	CUST_COUNTRY	OUTSTAND
Bangalore	India	12000
London	UK	4000
New York	USA	6000
New York	USA	6000
Bangalore	India	8000
London	UK	6000
London	UK	11000
New York	USA	3000
Brisban	Australia	5000
Brisban	Australia	7000
Chennai	India	8000
Mumbai	India	11000
Chennai	India	9000

customer

```
SELECT cust_city, cust_country,
MAX(outstanding_amt)
FROM customer
GROUP BY cust_country, cust_city;
```

CUST_CITY	CUST_CO	MAX(OUTSTA
Bangalore	India	12000
Brisban	Australia	7000
Chennai	India	11000
Hampshair	UK	5000
London	UK	11000
Mumbai	India	12000
New York	USA	6000
San Jose	USA	3000
Toronto	Canada	11000

ORDER BY cust\_city;

CUST_CITY	CUST_COUNTRY	MAX(OUTSTANDING_AMT)
Bangalore	India	12000
Brisban	Australia	7000
Chennai	India	11000
Hampshair	UK	5000
London	UK	11000
Mumbai	India	12000
New York	USA	6000
San Jose	USA	3000
Toronto	Canada	11000

# Ejemplos de consultas de agrupamiento

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar las ciudades en las cuales viven más de 2 empleados.

```
SELECT      Ciudad
FROM        Emp
GROUP BY    Ciudad
HAVING      COUNT(*) > 2
```

- Encontrar los proyectos en los cuales más de 2 empleados comparten una responsabilidad.

```
SELECT DISTINCT NoProy
FROM          Trabaja
GROUP BY      NoProy, Resp
HAVING        COUNT(*) > 2
```

# subquery

```
SELECT agent_code, COUNT(agent_code)
FROM orders GROUP BY agent_code
HAVING COUNT (agent_code)=
```

```
( SELECT MAX(mycount) FROM
```

```
( SELECT agent_code,
COUNT(agent_code) mycount
FROM orders GROUP BY agent_code ));
```

AGENT_CODE	ORD_NUM
A008	200114
A004	200122
A006	200118
A010	200119
A004	200121
A011	200130
A005	200134
A013	200115
A004	200108
A005	200103
A011	200105
A010	200109
A008	200101

orders

```
( SELECT agent_code,
COUNT(agent_code) mycount
FROM orders GROUP BY agent_code ));
```

each agent\_code  
makes a group here

AGENT_CODE	MYCOUNT
A004	4
A002	7
A007	2
A009	1
A011	2
A012	2

maximum agents  
in a group

```
( SELECT MAX(mycount) FROM
[result of inner query ] );
```

MAX(MYCOUNT)
7

```
SELECT agent_code, COUNT(agent_code)
FROM orders GROUP BY agent_code
HAVING COUNT (agent_code)= 7
```

AGENT_CODE	MYCOUNT
A004	4
A002	7
A007	2
A009	1
A011	2
A012	2

AGENT_CODE	COUNT(AGENT_CODE)
A002	7

# Ejemplos de consultas de agrupamiento

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Para cada ciudad donde hay más de tres proyectos, encontrar el nombre de los proyectos.

```
SELECT      Ciudad, NombProy
FROM        Proyecto
WHERE        Ciudad IN
              ( SELECT  Ciudad
                FROM      Proyecto
                GROUP BY Ciudad
                HAVING    COUNT(*) > 3 )
```

# Consultas que involucran operaciones de conjunto

- Las operaciones de conjunto unión, intersección y diferencia están especificadas por las cláusulas UNION, INTERSECT y EXCEPT (MINUS), respectivamente.

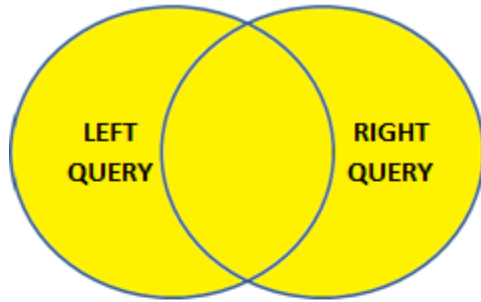
- Forma general:

```
(SELECT      A1,      . . . , An
FROM        R1,      . . . , Rm
WHERE        condicion)
{UNION | INTERSECT | EXCEPT} [ALL] | [CORRESPONDING [BY
columna1, ...]]
(SELECT      B1,      . . . , Bi
FROM        S1,      . . . , Sj
WHERE        condicion) ;
```

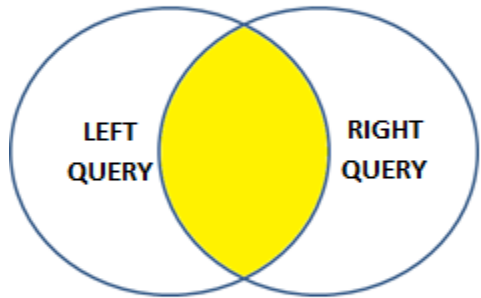
- Si se especifica ALL, devuelve el resultado incluyendo duplicados.
- Si se especifica en las consultas el asterisco (\*), entonces se establecen las columnas mediante CORRESPONDING BY. Si se omite BY, entonces actúa sobre las columnas en común.



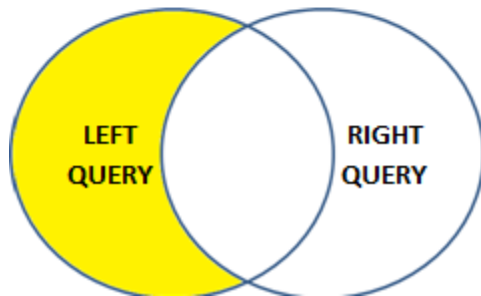
# Union / Intersect / Except



UNION operator returns all the unique rows from both the left and the right query. UNION ALL includes the duplicates as well



INTERSECT operator retrieves the common unique rows from both the left and the right query



EXCEPT operator returns unique rows from the left query that aren't in the right query's results

**Table A**

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male
3	Steve	Male

**Table B**

Id	Name	Gender
2	Mary	Female
3	Steve	Male
4	John	Male

**UNION Result**

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male
4	John	Male

**UNION ALL Result**

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male
3	Steve	Male
2	Mary	Female
3	Steve	Male
4	John	Male

**INTERSECT Result**

Id	Name	Gender
2	Mary	Female
3	Steve	Male

**EXCEPT Result**

Id	Name	Gender
1	Mark	Male

# Consultas con operaciones de conjunto

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar todas las ciudades donde haya un empleado o un proyecto.

```
(SELECT      Ciudad
FROM          Emp)
UNION
(SELECT      Ciudad
FROM          Proyecto)
```

- Encontrar todas las ciudades en las que un empleado trabaje pero no haya un proyecto.

```
(SELECT      Ciudad
FROM          Emp
EXCEPT
(SELECT      Ciudad
FROM          Proyecto)
```

# Consultas con operaciones de conjunto

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar las ciudades donde haya un empleado y un proyecto

```
(SELECT      Ciudad
FROM          Emp)
INTERSECT
(SELECT      Ciudad
FROM          Proyecto)
```

- La lista de los nombres de proyectos y empleados en Waterloo.

```
(SELECT      NombEmp
FROM          Emp
WHERE         Ciudad = 'Waterloo')
UNION ALL
(SELECT      NombProy
FROM          Proyecto
WHERE         Ciudad = 'Waterloo')
```

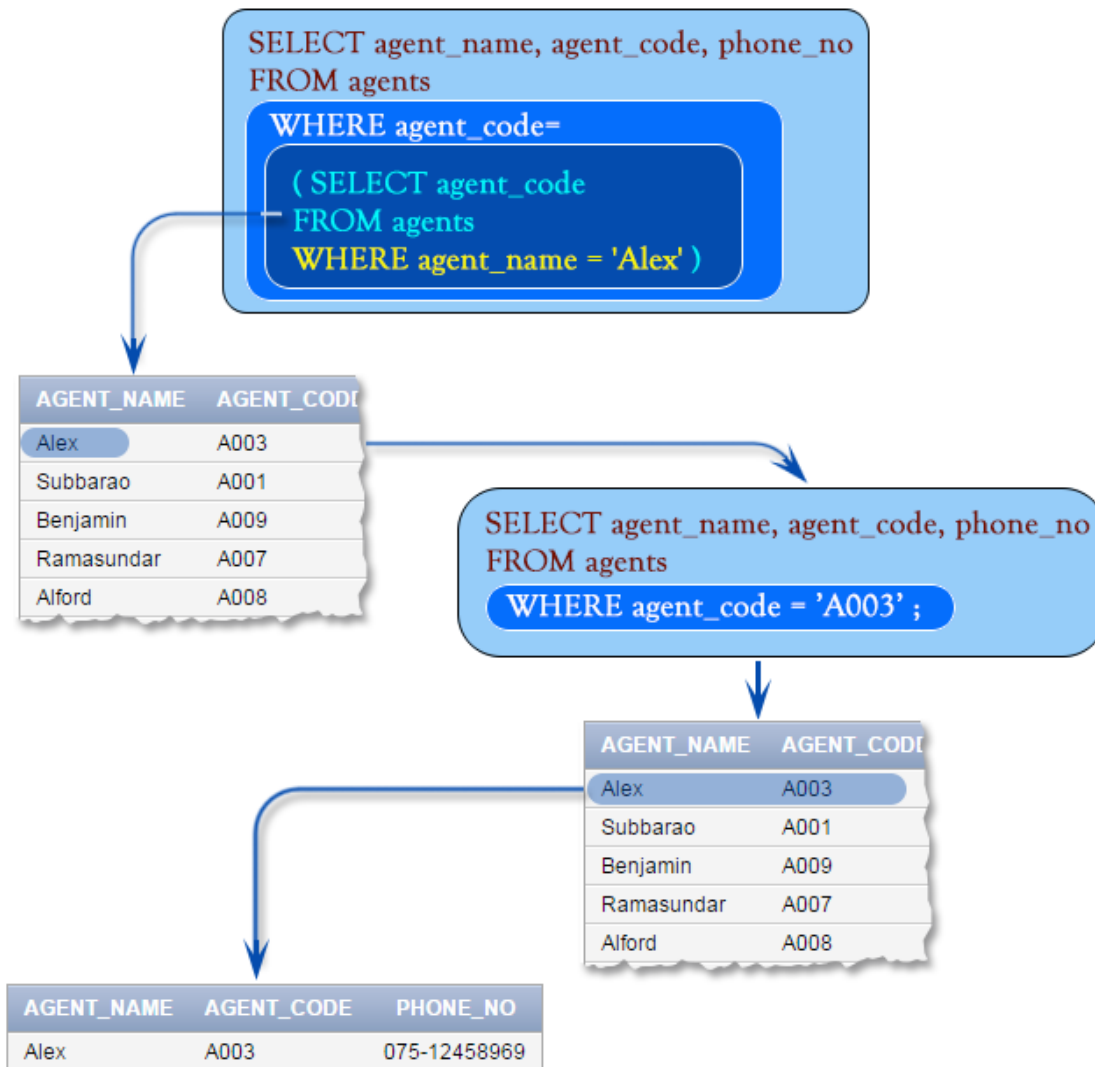
# Consultas con estructuras anidadas

- Pueden aplicarse consultas dentro de la cláusula WHERE que involucren pertenencia
- Forma general:

```
SELECT lista_columnas
FROM lista_tablas
WHERE {IN | [NOT] EXISTS}
      (SELECT lista_columnas
       FROM lista_tablas
       WHERE condicion)
```

- Puede haber niveles múltiples de anidamiento.
- Pueden ser normalmente escritos usando otros constructores (UNION, JOIN, etc.)

# subquery



# Subquery in having

```
SELECT AVG(ord_amount),COUNT(agent_code),  
agent_code FROM orders  
GROUP BY agent_code
```

```
HAVING AVG(ord_amount)=  
( SELECT AVG(ord_amount)  
FROM orders  
WHERE agent_code='A008' );
```

AGENT_CODE	ORD_AMOUNT
A008	3500
A004	2500
A006	500
A010	4000
A004	1500
A011	2500
A005	4200
A013	2000
A004	4000
A005	1500
A011	2500
A010	3500
A008	3000
A008	1000
A004	1500

AVG(ord\_amount)

```
SELECT AVG(ord_amount),COUNT(agent_code),  
agent_code FROM orders  
GROUP BY agent_code
```

```
HAVING AVG(ord_amount)= 2500
```

AVE_ORD_AMT	NO_OF_AGENT	AGENT_CODE
2375	4	A004
1814.29	7	A002
1250	2	A007
500	1	A009
2500	2	A011
1450	2	A012
3400	5	A010
3000	2	A013
800	1	A001
2500	3	A008

AVG(ORD_AMOUNT)	COUNT(AGENT_CODE)	AGENT_CODE
2500	2	A011
2500	3	A008

# Construcción IN

- Forma general:

```
SELECT A1, . . . , An
FROM R
WHERE A1, . . . , An [NOT] IN
      (SELECT A1, . . . , An
       FROM R1, . . . , Rm
       WHERE P)
```

- Semántica: las tuplas con atributos  $A_1, \dots, A_n$  se encuentran en la relación que es devuelta como un resultado de cálculo de la consulta interna.
- Otros operadores de comparación  $\{<, <=, >, >=, <>\}$  pueden ser utilizados con ALL o con ANY en lugar de IN.

# Ejemplo de la construcción IN

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

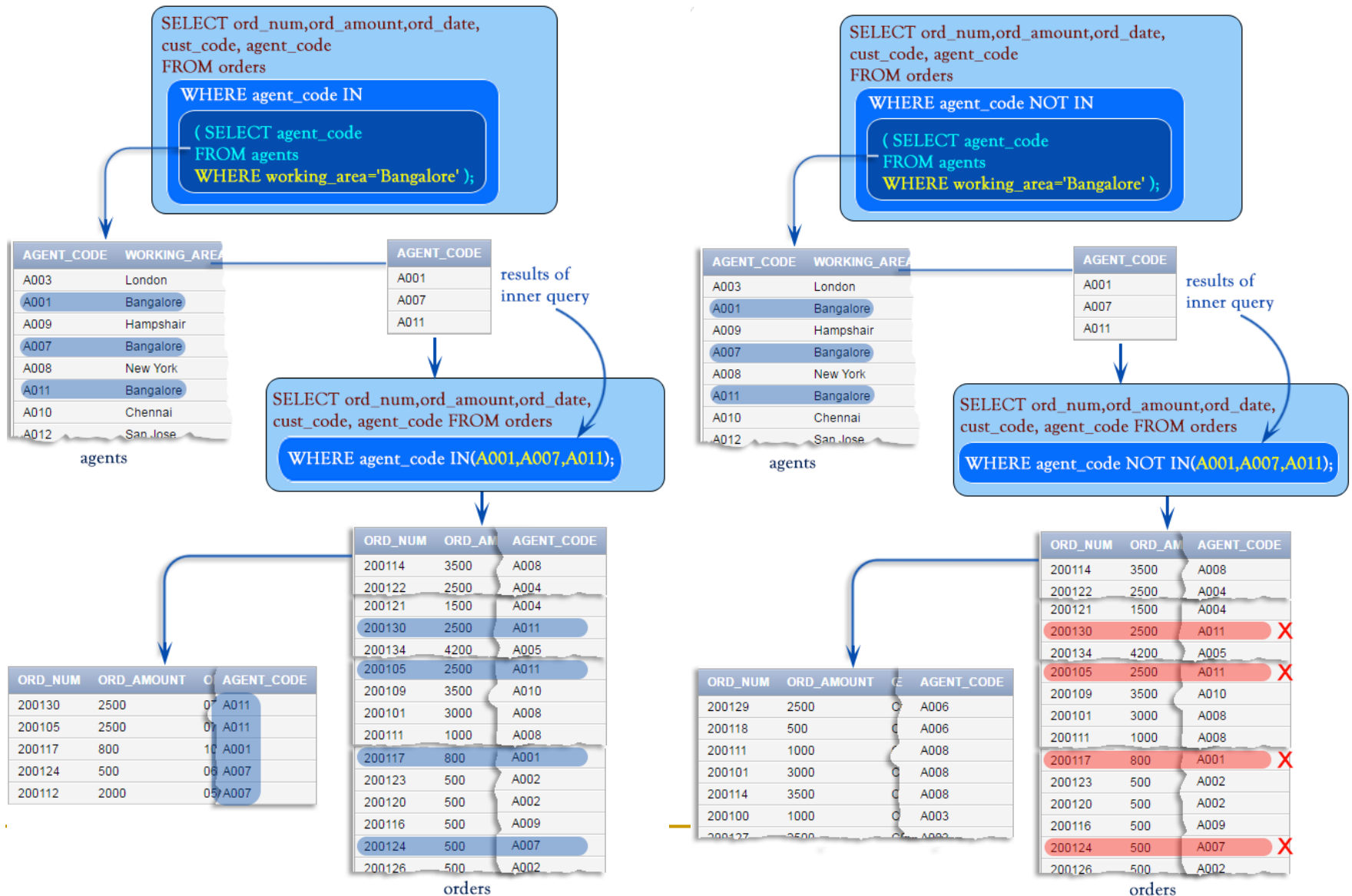
**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar los nombres de todos los empleados que trabajan en una ciudad donde estén localizados los proyectos que tengan presupuestos menores de \$50,000

```
SELECT  NombEmp
FROM    Emp
WHERE   Ciudad IN (SELECT  Ciudad
                    FROM    Proyecto
                    WHERE   Presupuesto < 50000)
```



# Subquery IN / NOT IN



# Ejemplo de la construcción IN

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar los nombres de todos los empleados que trabajan en una ciudad donde un proyecto se localice o que tengan una profesión que pague más de \$60,000.

```
SELECT    NombEmp
FROM      Emp
WHERE     Ciudad IN (SELECT  Ciudad
                     FROM      Proyecto)
OR
          Titulo IN (SELECT  Titulo
                     FROM      Pago
                     WHERE     Salario > 60000)
```

# Ejemplo de la construcción ANY/ALL/SOME

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar los nombres de todos los proyectos que tienen presupuesto mayor que todos los proyectos en Calgary.

```
SELECT    NombProy
FROM      Proyecto
WHERE     Presupuesto > ALL
          (SELECT  Presupuesto
           FROM      Proyecto
           WHERE     Ciudad = 'Calgary')
```

- Es posible usar ANY si lo que desea es encontrar los proyectos cuyos presupuestos son mayores que algún proyecto en Calgary.

# Subquery ANY

>ANY means greater than at least one value, that is, greater than the minimum.

WHERE 70 > ANY (20, 56, 5, 15, 69, 10);

So 70 > 5 is true, and data returns.

<ANY means less than at least one value, that is, less than the maximum.

WHERE 70 < ANY (20, 56, 5, 15, 69, 10);

So 70 < 69 is false, and no data returns.

>ANY means greater than at least one value, that is, greater than the minimum.

WHERE 4 > ANY (20, 56, 5, 15, 69, 10);

So 4 > 5 is false, and no data returns.

<ANY means less than at least one value, that is, less than the maximum.

WHERE 4 < ANY (20, 56, 5, 15, 69, 10);

So 4 < 69 is true, and data returns.

```
SELECT agent_code, agent_name, working_area,
commission
FROM agents
```

WHERE agent\_code = ANY

```
( SELECT agent_code
FROM customer
WHERE cust_country='UK');
```

AGENT_CODE	CUST_COUNTRY
A007	India
A003	UK
A008	USA
A008	USA
A011	India
A006	UK
A003	UK
A008	USA
A005	Australia

customer

AGENT_CODE
A009
A003
A006
A003
A006

results of inner query

```
SELECT agent_code, agent_name, working_area,
commission FROM agents
WHERE agent_code=ANY( any row from inner
query );
```

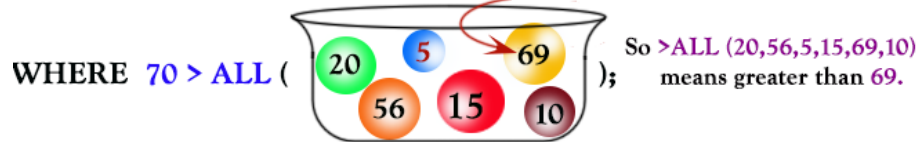
AGENT_CODE	AGENT_NAME	COMMISSION
A003	Alex	.13
A001	Subbarao	.14
A009	Benjamin	.11
A007	Ramasunda	.15
A008	Alford	.12
A011	Ravi Kumar	.15
A010	Santakumar	.14
A012	Lucida	.12

agents

AGENT_CODE	AGENT_NAME	COMMISSION
A009	Benjamin	.11
A003	Alex	.13
A006	McDen	.15

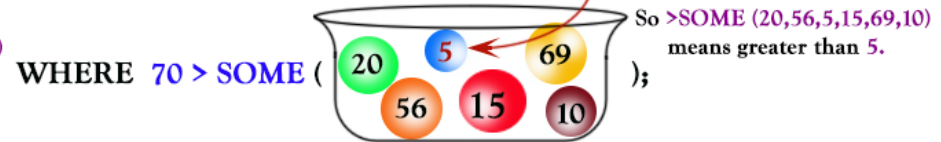
# Subquery ALL /SOME

>ALL means greater than the biggest value,  
that is, greater than the maximum.



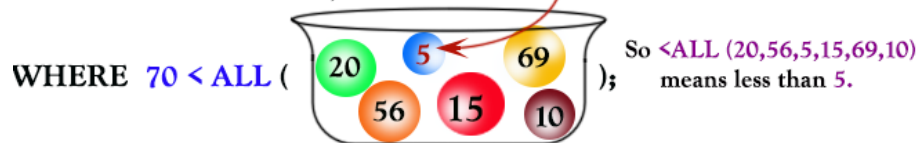
So 70 > 69 is true, and data returns.

> SOME means greater than at least one value,  
that is, greater than the minimum.



So 70 > 5 is true, and data returns.

< ALL means less than the smallest value,  
that is, less than the minimum



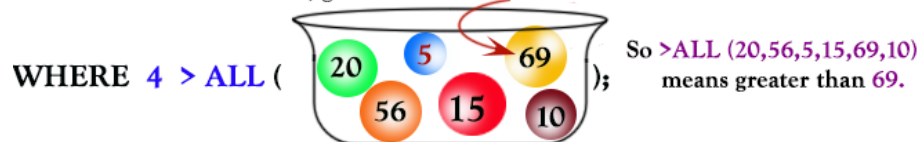
So 70 < 5 is false, and no data returns.

< SOME means less than at least one value,  
that is, less than the maximum.



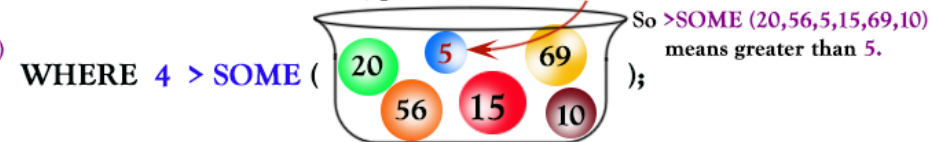
So 70 < 69 is false, and no data returns.

>ALL means greater than the biggest value,  
that is, greater than the maximum.



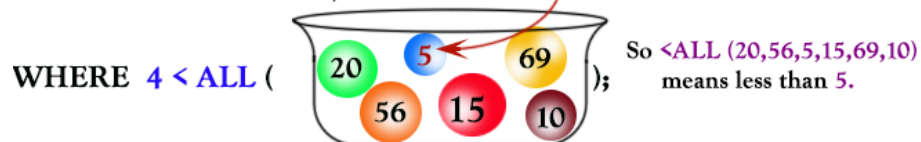
So 4 > 69 is false, and no data returns.

>SOME means greater than at least one value,  
that is, greater than the minimum.



So 4 > 5 is false, and no data returns.

< ALL means less than the smallest value,  
that is, less than the minimum.



So 4 < 5 is true, and data returns.

< SOME means less than at least one value,  
that is, less than the maximum.



So 4 < 69 is true, and data returns.

# Construcción EXIST

- Forma general:

```
SELECT lista_columnas
FROM lista_tablas
WHERE [NOT] EXISTS
      (SELECT *
       FROM   R1, . . . , Rm
       WHERE  P)
```

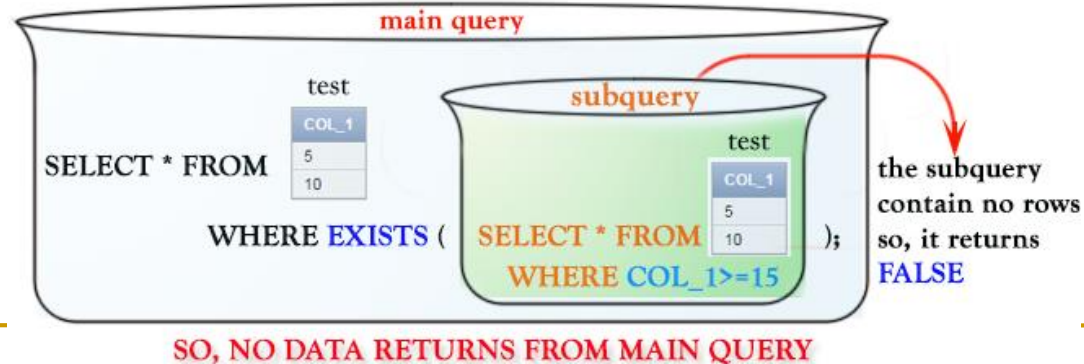
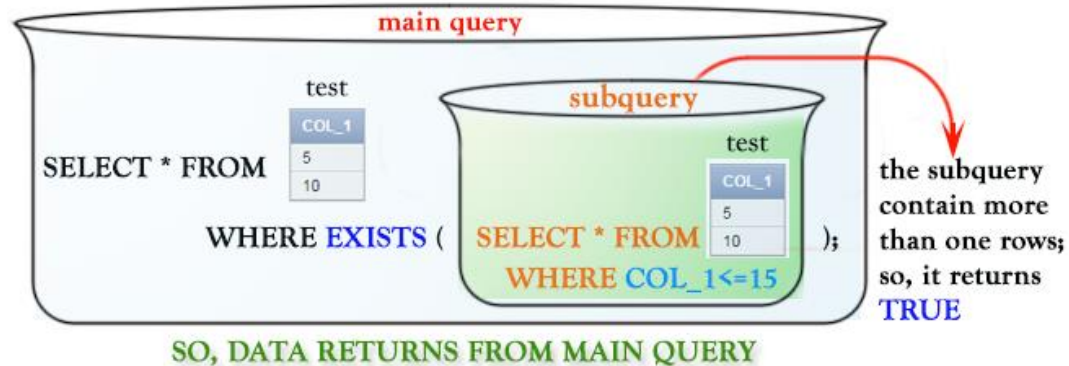
- Semántica: para cada tupla de la consulta externa, ejecuta la consulta interna; si (no) hay al menos una tupla en el resultado de la consulta interna, entonces recupera esa tupla de la consulta exterior.

# Exists

WHERE EXISTS ( **subquery** );

## EXISTS operator

- EXISTS is a Comparison operator
- Used in WHERE clause to validate an "IT EXISTS" condition.
- EXISTS will tell you whether a query returned any results.
- Returns a BOOLEAN, (TRUE or FALSE).
- Returns TRUE if a subquery contains any rows.





# EXIST / NOT EXIST

```
SELECT employee_id, manager_id, first_name,
last_name FROM employees a
```

WHERE EXISTS

```
(SELECT employee_id
FROM employees b
WHERE b.manager_id = a.employee_id);
```

FIRST_NAME	EMPLOYEE_ID	MANAGER_ID
Steven	100	-
Neena	101	100
Alex	102	100
Alexander	103	102
Bruce	104	103
David	105	103
Valli	106	103
Diana	107	103
Nancy	108	101
Daniel	109	108

employees alias b

record matches in  
more than one rows

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME
100	-	Steven
101	100	Neena
102	100	Alex
103	102	Alexander
104	103	Bruce
105	103	David
106	103	Valli
107	103	Diana
108	101	Nancy
109	108	Daniel

employees alias a

EXISTS [ TRUE ]

```
SELECT employee_id, manager_id, first_name,
last_name FROM employees a
```

WHERE EXISTS [ TRUE ]

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME
100	-	Steven
101	100	Neena
102	100	Alex
103	102	Alexander
104	103	Bruce
105	103	David
106	103	Valli
107	103	Diana
108	101	Nancy
109	108	Daniel

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME
100	-	Steven
101	100	Neena
102	100	Alex
103	102	Alexander
108	101	Nancy
114	100	Den
120	100	Matthew
121	100	Adam

results

```
SELECT employee_id, manager_id, first_name,
last_name FROM employees a
```

WHERE NOT EXISTS

```
(SELECT employee_id
FROM employees b
WHERE b.manager_id = a.employee_id);
```

FIRST_NAME	EMPLOYEE_ID	MANAGER_ID
Steven	100	-
Neena	101	100
Alex	102	100
Alexander	103	102
Bruce	104	103
David	105	103
Valli	106	103
Diana	107	103
Nancy	108	101
Daniel	109	108

employees alias b

unmatched  
rows

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME
100	-	Steven
101	100	Neena
102	100	Alex
103	102	Alexander
104	103	Bruce
105	103	David
106	103	Valli
107	103	Diana
108	101	Nancy
109	108	Daniel

employees alias a

NOT EXISTS [ TRUE ]

```
SELECT employee_id, manager_id, first_name,
last_name FROM employees a
```

WHERE NOT EXISTS [ TRUE ]

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME
100	-	Steven
101	100	Neena
102	100	Alex
103	102	Alexander
104	103	Bruce
105	103	David
106	103	Valli
107	103	Diana
108	101	Nancy
109	108	Daniel

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME
104	103	Bruce
105	103	David
106	103	Valli
107	103	Diana
109	108	Daniel
110	108	John
111	108	Ismael

results



# Ejemplo de la construcción "EXISTS"

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar los nombres de empleados que trabajan en una ciudad en la cual algún proyecto se localiza.

```
SELECT NombEmp
FROM Emp
WHERE EXISTS (SELECT *
               FROM Proyecto
               WHERE Emp.Ciudad = Proyecto.Ciudad)
```

- Encontrar los nombres de empleados que trabajan en una ciudad en la cual no se localiza un proyecto.

```
SELECT NombEmp
FROM Emp
WHERE NOT EXISTS (SELECT *
                  FROM Proyecto
                  WHERE Emp.Ciudad = Proyecto.Ciudad)
```

# Ejemplo de la estructura EXISTS

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Encontrar todos los empleados que trabajan en cada proyecto.
  - Reescrito: encontrar todos los empleados tal que no esté en un proyecto en el cual no trabaje.

```
SELECT  *
FROM    Emp
WHERE   NOT EXISTS
        (SELECT  NoProy
         FROM    Proyecto
         WHERE NOT EXISTS
              (SELECT  *
               FROM    Trabaja
               WHERE   Proyecto.NoProy = Trabaja.NoProy)
              AND     Emp.NoEmp = Trabaja.NoEmp))
```

# Definición de Vista

- Se considera una relación “virtual” resultado de la ejecución de operaciones sobre una o mas relaciones base.
- No existe en la base de datos, sino que es producida al tiempo de la petición.
- Forma general

```
CREATE VIEW nombre_vista[(nombre_columna [, . . .])]  
AS subconsulta  
[WITH CHECK OPTION]
```
- Si se especifica la lista de columnas, debe coincidir con las columnas devueltas por la subconsulta. Si se omite, los nombres de las columnas toman los nombres devueltos por la subconsulta.
- En consultas, se trata la vista como una relación base.

---

# Definición de Vista

- La lista de columnas se debe especificar si existe ambigüedad en los nombres de columnas
  - La subconsulta se conoce como la consulta de definición de la vista
  - La opción **WITH CHECK OPTION** asegura que si un renglón falla en satisfacer la cláusula **WHERE** de la consulta de definición, esta no es agregada a la tabla base en casos de inserciones y actualizaciones mediante la vista.
  - Eliminación de vistas:  

```
DROP VIEW nombre_vista  
[RESTRICT | CASCADE]
```
  - Con **CASCADE** se eliminan todos los objetos dependientes relacionados (vistas definidas en la vista)
  - Con **RESTRICT**, si hay objetos que dependen de la existencia de la vista, se rechaza la eliminación.
-

# Ejemplos de la definición de Vistas

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Crear una vista *proyectos-Waterloo* de proyectos que se localizan en Waterloo.

```
CREATE VIEW proyectos-waterloo
AS SELECT *
FROM Proyecto
WHERE Ciudad = 'Waterloo'
```

- Crear un vista de *empleados\_ricos* que consiste en empleados que ganan más de \$75,000.

```
CREATE VIEW empleados_ricos
AS SELECT NoEmp, NombEmp, Titulo, Ciudad
FROM Emp, Pago
WHERE Salario > 75000
AND Emp.Titulo = Pago.Titulo
```

# Ejemplos de la definición de Vistas

**Emp** (NoEmp, NombEmp, Titulo, Ciudad)

**Proyecto** (NoProy, NombProy, Presupuesto, Ciudad)

**Pago** (Titulo, Salario)

**Trabaja** (NoEmp, NoProy, Resp, Dur)

- Crear una vista *proyecto-empleado* que proporcione, para cada proyecto en cada ciudad, el número de empleados que trabajan en el proyecto y la duración total de empleo.

```
CREATE VIEW proyecto-empleado (Nombre, Ciudad, Numero, Total)
AS SELECT  NombProy, Ciudad, COUNT(NoEmp), SUM(Dur)
FROM      Proyecto P, Trabaja T
WHERE     P.NoProy = W.NoProy
GROUP BY Ciudad, NombProy
```

# Actualización de vistas

- Todas las actualizaciones a las tablas base son reflejadas en todas las vistas que engloban la tabla base. Similarmente, se puede esperar que si una vista es actualizada entonces la tabla base refleje los cambios hechos. El estándar SQL92 especifica que las vistas deben ser actualizables en un sistema que este conforme a este estándar.
- De esta definición, una vista es actualizable si:
  - La cláusula DISTINCT no es especificada
  - Cada elemento en la lista del SELECT de la consulta de definición es un nombre de columna y no hay columnas que aparezcan más de una vez
  - Si los atributos de la vista contienen la llave primaria o algún otro atributo como llave candidata (sin nulos)
  - La cláusula FROM especifique solo una tabla, excluyendo cualquier vista basada en una reunión, unión, intersección o diferencia
  - La cláusula WHERE no incluya ningún SELECT anidado que haga referencia a una tabla en la cláusula FROM
  - No haya una cláusula GROUP BY o HAVING en la consulta de definición
- Una vista agrupada nunca debe ser reunida con una tabla base o con otra vista.
- Además, cada renglón agregado a través de la vista no debe violar las restricciones de integridad de la tabla base

# Resolución de vistas

- La relación virtual producida por una vista se crea de la siguiente forma:
  - a) los nombres de columnas de la vista en la cláusula SELECT son trasladadas en las columnas correspondientes en la consulta que la define
  - b) los nombres de vistas en la cláusula FROM son remplazadas con las listas FROM correspondientes de la consulta de definición
  - c) la cláusula WHERE es combinada (mediante AND) con las condiciones de la consulta de definición
  - d) las cláusulas GRUPO BY y HAVING son copiadas de la consulta de definición
  - e) la cláusula ORDER BY es copiada de la consulta con los nombres de las columnas de la vista traducidos en los nombres de columnas de la consulta de definición
  - f) el resultado es ejecutado para obtener el resultado final



---

# Ventajas y desventajas de las vistas

- Ventajas:
    - Independencia de datos
    - Seguridad
    - Complejidad reducida
    - Conveniencia
    - Personalización
    - Integridad de los datos
  - Desventajas
    - Restricciones de actualización
    - Restricciones en la estructura
    - Rendimiento
-

# Control de acceso

- Un identificador de autorización es el método usado por un SABD para establecer una identidad del usuario. Usualmente tiene asociada una contraseña
- Es utilizado para determinar cuales objetos pueden ser referenciados por el usuario y que operaciones pueden ser ejecutadas sobre esos objetos
- Cada objeto creado mediante sentencias SQL tiene un propietario, como se especifica en la cláusula AUTORIZATION del esquema al cual pertenece el objeto
- El propietario es la única persona que tiene conocimiento de esto

# Conceder privilegios

- Un propietario de una tabla puede ceder los privilegios a otro usuario mediante la clausula GRANT

- Forma general:

```
GRANT {lista_privilegios | ALL PRIVILEGES}
ON  nombre_objeto
TO {lista_id_aut | PUBLIC}
[WITH GRANT OPTION]
```

- *lista\_id\_aut* consiste de uno o más de los privilegios vistos, separados por comas
- *nombre\_objeto* puede ser una tabla base, una vista, un dominio, un conjunto de caracteres, una comparación o una traducción
- ALL PRIVILEGES cede todos los privilegios a un usuario
- PUBLIC habilita a todos los usuarios (presentes y futuros) los privilegios mencionados
- WITH GRANT OPTION permite que un usuario ceda los privilegios obtenidos a otro usuario

# Privilegios

- Las acciones que puede permitir un usuario hacia una tabla base o una vista son:
  - *Select*.- permite el acceso de lectura a la relación, o la capacidad de consultar usando una vista
    - `grant select on branch to user1, user2, user3`
  - *Insert*.- habilita la inserción de tuplas
  - *Update*.- habilita la actualización empleando la sentencia update de SQL
  - *Delete*.- habilita eliminar tuplas
  - *References*.- habilita declarar llaves foráneas cuando se crean relaciones.
  - *Usage*.- en el SQL-92, autoriza a un usuario a usar un dominio específico.
  - *All privileges*.- forma corta de especificar todos los privilegios.
- Se pueden restringir los INSERT/UPDATE/REFERENCES a las columnas con nombre
- Para crear una vista, el usuario debe tener los privilegios de SELECT en todas las tablas que conforman la vista y el privilegio REFERENCES en las columnas nombradas

# Algunos privilegios en Oracle

Object Privilege	Table	View	Sequence	Procedure
ALTER	√		√	
DELETE	√	√		
EXECUTE				√
INDEX	√			
INSERT	√	√		
REFERENCES	√	√		
SELECT	√	√	√	
UPDATE	√	√		

### Privilegios de Sistema

Privilegio de sistema	Operaciones Autorizadas
ALTER ANY CLUSTER	Alterar cualquier cluster en cualquier esquema
ALTER ANY INDEX	Alterar cualquier índice en cualquier esquema
ALTER ANY PROCEDURE	Alterar cualquier procedimiento, función o paquete almacenado en cualquier esquema
ALTER ANY ROLE	Alterar cualquier role en la BD.
ALTER ANY SEQUENCE	Alterar cualquier secuencia en la BD.
ALTER ANY SNAPSHOT	Alterar cualquier snapshot (es una tabla que contiene los resultados de una consulta de una o más tablas, a menudo localizados en una BD remota)
ALTER ANY TABLE	Alterar cualquier tabla o vista en el esquema.
ALTER ANY TRIGGER	Habilitar, deshabilitar o compilar cualquier trigger de la BD en cualquier esquema.
ALTER DATABASE	Alterar la BD.
ALTER PROFILE	Alterar los perfiles
ALTER RESOURCE COST	Actualizar los costes para los recursos de una sesión.
ALTER ROLLBACK	Alterar los segmentos de rollback.
ALTER SESSION	Usar la sentencia ALTER SESSION
ALTER SYSTEM	Usar la sentencia ALTER SYSTEM
ALTER TABLESPACE	Alterar los tablespaces
ALTER USER	Alterar cualquier usuario. Este privilegio autoriza a quien lo recibe a cambiar contraseñas a otros usuarios o el método de autenticación, asignar cuotas en cualquier tablespace, definir los
ANALYZE ANY	Analizar cualquier tabla, cluster o índice en cualquier
AUDIT ANY	Auditar cualquier objeto en cualquier esquema usando la sentencia
AUDIT SYSTEM	Usar la sentencia AUDIT (Sentencias SQL).
BACKUP ANY TABLE	Usar la utilidad Export para exportar objetos incrementalmente desde los esquemas de otros
BECOME USER	Allows grantee to become another user. (Required by any user performing a full database
COMMENT ANY TABLE	Poner un comentario en cualquier tabla, vista o columna en cualquier esquema
CREATE ANY CLUSTER	Crear un cluster en cualquier esquema.
CREATE ANY INDEX	Crear un índice en cualquier esquema en cualquier
CREATE ANY PROCEDURE	Crear procedimientos, funciones y paquetes almacenados en cualquier esquema.
CREATE ANY SEQUENCE	Crear cualquier secuencia en la BD.
CREATE ANY SNAPSHOT	Crear cualquier snapshot en la BD.

CREATE ANY SYNONYM	Crear cualquier sinónimo en cualquier esquema
CREATE ANY TABLE	Crear tablas en cualquier esquema. El que crea la tabla debe tener cuota de espacio en el tablespace que
CREATE ANY TRIGGER	Crear un trigger en cualquier esquema asociado con una tabla de cualquier esquema.
CREATE ANY VIEW	Crear vistas en cualquier esquema
CREATE CLUSTER	Crear un cluster en cualquier esquema
CREATE DATABASE LINK	Crear enlaces de BD privados en nuestro esquema. Crear procedimientos, funciones y paquetes
CREATE PROFILE	Crear perfiles
CREATE PUBLIC DATABASE	Crear enlaces de BD públicos
CREATE PUBLIC SYNONYM	Crear sinónimos públicos
CREATE ROLE	Crear roles
CREATE ROLLBACK	Crear segmentos de rollback
CREATE SEQUENCE	Crear secuencias en nuestro esquema
CREATE SESSION	Conectar a la BD
CREATE SNAPSHOT	Crear snapshots
CREATE SYNONYM	Crear sinónimos en nuestro esquema
CREATE TABLE	Crear tablas en nuestro esquema. Para crear una tabla quien recibe el permiso debe tener espacio en la cuota
CREATE TABLESPACE	Crear tablespaces.
CREATE TRIGGER	Crear trigger de BD en nuestro esquema.
CREATE USER	Crear usuarios. Este privilegio también permite al creador asignar cuotas en cualquier tablespace, definir tablespace temporal y por defecto para el usuario y asignar un perfil como parte de una sentencia
CREATE VIEW	Crear vistas en nuestro esquema
DELETE ANY TABLE	Borrar tuplas de tablas o vistas en cualquier esquema o vaciar tablas en cualquier esquema
DROP ANY CLUSTER	Borrar clusters en cualquier esquema
DROP ANY INDEX	Borrar índices en cualquier esquema
DROP ANY PROCEDURE	Borrar procedimientos, funciones y paquetes almacenados, en cualquier esquema.
DROP ANY ROLE	Borrar roles.
DROP ANY SEQUENCE	Borrar secuencias en cualquier esquema
DROP ANY SNAPSHOT	Borrar snapshots en cualquier esquema
DROP ANY SYNONYM	Borrar sinónimos privados en cualquier esquema.
DROP ANY TABLE	Borrar tablas en cualquier esquema.
DROP ANY TRIGGER	Borrar trigger de BD en cualquier esquema
DROP ANY VIEW	Borrar vistas en cualquier esquema
DROP PROFILE	Borrar perfiles
DROP PUBLIC DATABASE	Borrar enlaces de BD públicos

# Ejemplo de privilegios cedidos

- Dar todos los privilegios de la tabla Staff al administrador:

```
GRANT ALL PRIVILEGES  
ON Staff  
TO manager WITH GRANT OPTION;
```

- Dar sólo los privilegios de selección y actualización en la columna salary:

```
GRANT SELECT, UPDATE (salary)  
ON Staff  
TO manager;
```

- Dar el privilegios de selección a los usuarios personal y contabilidad:

```
GRANT SELECT  
ON Staff  
TO personal, contabilidad;
```

# Revocación de privilegios

- Para revocar los privilegios cedidos a un usuario mediante GRANT, se usa la cláusula REVOKE

- Forma general:

```
REVOKE [GRANT OPTION FOR] {lista_privilegios | ALL PRIVILEGES}
ON    nombre_objeto
FROM {lista_id_aut | PUBLIC}
[RESTRICT | CASCADE]
```

- ALL PRIVILEGES se refiere a todos los privilegios cedidos a un usuario por el usuario que cedió los privilegios
- GRANT OPTION FOR habilita el paso de los privilegios mediante WITH GRANT OPTION del GRANT para ser revocados separadamente de los privilegios por sí mismos
- REVOKE falla si resulta en un objeto abandonado, como una vista, a menos que sea especificado CASCADE
- Los privilegios otorgados hacia un usuario por otro usuario no son afectados



# Ejemplo de privilegios revocados

- Revocar el privilegio SELECT en la tabla *Branch* para todos los usuarios

```
REVOKE SELECT
```

```
ON Branch
```

```
FROM PUBLIC;
```

- Revocar todos los privilegios otorgados a la tabla *Empleados* para el usuario Personal

```
REVOKE ALL PRIVILEGES
```

```
ON Empleados
```

```
FROM Personal;
```

---

# Roles

- Un rol permite agrupar privilegios comunes para una clase de usuarios.
- Los privilegios pueden ser cedidos o revocados desde los roles, así como a los usuarios.
- Los roles pueden se asignados a los usuarios, e inclusive a otros roles.
- El estándar SQL:99 soporta roles

```
create role manager
```

```
grant select on branch to manager
```

```
grant update(balance) on account to manager
```

```
grant all privileges on account to manager
```

```
grant manager to user1, user2
```

---

# Transacciones

- El SQL92 define el modelo de transacciones basado en las sentencias COMMIT y ROLLBACK
- Una transacción es una unidad lógica de trabajo con una o más sentencias SQL, garantizando que sean atómicas con respecto a su recuperación
- Una transacción en SQL comienza automáticamente con una sentencia SQL (ej. SELECT, INSERT, etc). Los cambios hechos por la transacción no son visibles a otra transacción ejecutando concurrentemente hasta que la transacción termina
- Una transacción puede ser completada en una de cuatro formas:
  - COMMIT termina la transacción exitosamente, haciendo los cambios permanentes
  - ROLLBACK aborta la transacción, deshaciendo cualquier cambio hecho por la transacción
  - En el SQL programático, la terminación exitosa del programa finaliza la transacción exitosamente, inclusive si un COMMIT no hubiese sido ejecutado
  - En el SQL programático, una terminación anormal del programa aborta la transacción

# Niveles de serialización en el SQL92

- El estándar SQL no asume que cada transacción ejecute en una forma serializable, y los sistemas comerciales le permiten al usuario especificar el grado de control de concurrencia que desean.
  - En el SQL2, el máximo nivel de aislamiento es **serializable**, que significa que una transacción ejecutará en un instante y las otras transacciones ejecutarán hasta que se haya completado completamente antes o después.
  - Un nivel de **lectura repetible (repeatable read)** es cuando solo se leen registros que se han comprometido; cuando se lee el mismo dato varias veces, siempre se lee el mismo valor.
  - El nivel de **lectura comprometida (read committed)** lee datos que se han comprometido, pero diferentes lecturas del mismo registro puede resultar en valores distintos. Es posible que una transacción pueda leer el elemento A dos veces y obtenga diferentes valores, cada uno escrito por transacciones comprometidas.
  - El cuarto nivel es el de **lectura no comprometida (read uncommitted)**, en donde no hay restricciones del todo para las transacciones que operan en este nivel. Solo a este nivel es permitido que una transacción lea datos sucios.

# Soporte de transacciones en el SQL92

- Cada transacción tiene un modo de acceso, un tamaño del área de diagnósticos y un nivel de aislamiento.

<b>Nivel de aislamiento</b>	<b>Lectura sucia (dirty read)</b>	<b>Lectura no repetible (nonrepeatable read)</b>	<b>Registros fantasmas (phantom problem)</b>
Read Uncommitted	Pueden existir	Puede existir	Puede existir
Read committed	No	Puede existir	Puede existir
Repeatable reads	No	No	Puede existir
Serializable	No	No	No

---

# Terminación de Consultas SQL

- **COMMIT**
    - Si se quiere hacer la actualización de resultados permanente.
    - Incrustado: **EXEC SQL COMMIT;**
  - **ROLLBACK**
    - Si se quiere descartar los resultados
    - Incrustado: **EXEC SQL ROLLBACK;**
-

---

# Transacciones

- Una nueva transacción comienza con la siguiente sentencia que involucre una transacción
  - Las transacciones en SQL no pueden ser anidadas
  - La sentencia SET TRANSACTION configura el comportamiento de una transacción:
  - `SET TRANSACTION [READ ONLY | READ WRITE] |  
[ISOLATION LEVEL READ UNCOMMITTED | READ COMMITTED  
| REPEATABLE READ | SERIALIZABLE]`
-

# Restricciones inmediatas y diferidas

- No siempre se desea que se verifiquen las restricciones inmediatamente, sino hasta que una transacción comprometa
- La sentencia SET CONSTRAINTS es usada para establecer el modo de la restricción especificada para la transacción actual:
  - SET CONSTRAINTS
  - {ALL | nombre\_restriccion[,...]}
  - {DEFERRED | IMMEDIATE}
- Las restricciones pueden ser definidas como INITIALLY IMMEDIATE o INITIALLY DEFERRED, indicando el modo de restricción que se asume al iniciar la transacción
- En el último caso, también es posible especificar cuando un modo puede ser cambiado subsecuentemente usando el calificador [NOT] DEFERRABLE
- El modo por omisión es INITIALLY IMMEDIATE



# Disparadores (triggers)

- Un disparador es un procedimiento que es invocado automáticamente por el SABD en respuesta a cambios en la base de datos.
- Consta de tres partes:
  - El evento
    - El cambio en la base de datos que inicia la ejecución del disparador
  - La condición
    - Una prueba o una consulta que necesita ser verificada cuando un disparador es activado
    - Para las consultas, la condición es verdad (TRUE) si la consulta regresa cualquier resultado
  - La Acción
    - El procedimiento que es ejecutado cuando el disparador es activado y la condición es verdadera
    - Puede ser antes o después de la activación del disparador
    - Hace referencia al valor anterior y al nuevo valor de la tupla, limitado a uno o más atributos
    - Una vez por cada tupla modificada o para todos los cambios

# Disparadores

- Forma general:

```
CREATE TRIGGER nombre_disparador
{AFTER | BEFORE} {UPDATE [ON nombre_columna[, ...]] | INSERT |
DELETE} ON nombre_tabla
[REFERENCING {OLD [TABLE | ROW] AS valor_anterior} | {NEW
[TABLE | ROW] AS valor_nuevo}]
[FOR EACH STATEMENT | FOR EACH ROW]
[WHEN (condicion)]
sentencias_procedimentales
```

- Un disparador no puede ser llamado directamente; sólo se activa por eventos en la base de datos
- Puede ser síncrono o asíncrono con respecto a la transacción que causa que sea disparado
- Ejecutado en la base de *por sentencia*
- Hace referencia a nuevos y/o viejos valores

# Ejemplos de disparadores

- Se usa el siguiente disparador para verificar que el valor de descuento para un nuevo cliente no exceda a 15.0; sino, manda un mensaje de error

```
CREATE TRIGGER discount
AFTER INSERT ON Customers
REFERENCING NEW AS NewTuple
FOR EACH ROW
WHEN (NewTuple.discnt > 15.0)
mensaje_error
```

- Se usa el siguiente disparador para aumentar la cuenta de estudiantes en la tabla *StatisticsTable* para cada nuevo estudiante insertado y su edad sea menor de 18 años

```
CREATE TRIGGER SetCount
AFTER INSERT ON Students
REFERENCING NEW TABLE AS InsertedTuples
FOR EACH STATEMENT
INSERT INTO StatisticsTable((ModifiedTable, ModificationType, Num)
SELECT 'Students', 'Insert', COUNT(*))
FROM InsertedTuples I
WHERE I.age < 18
```

---

# Acciones ajenas del mundo

- Algunas veces se requieren acciones externas del mundo para que se dispare una actualización de la base de datos
    - Por ejemplo, ordena un producto cuya cantidad en bodega es pequeña, o prender una luz de alarma
  - Los disparadores no pueden ser usados directamente para implementar acciones externas del mundo, pero
    - Pueden ser usados para registrar acciones a ser tomadas en una tabla separada
    - Tener un proceso externo que repetidamente busque en la tabla, realice las acciones externas del mundo y realice eliminaciones de la tabla.
-

# Cuando no usar disparadores

- Los disparadores son usados para tareas como
  - Mantener datos sumariados.
  - Replicar bases de datos, grabando los cambios a relaciones especiales (llamadas relaciones de cambio) y teniendo un proceso separado que aplica los cambios sobre las replicas
- Hay mejores formas de hacer esto ahora:
  - Las bases de datos actuales proporcionan facilidades de vistas materializadas interconstruidas para mantener datos sumariados
  - Las bases de datos proporcionan soporte interconstruido para la replicación

---

# Los Problemas del disparador

- Es específico del fabricante del SABD (ahora es una parte del estándar SQL99)
  - Los disparadores pueden hacer al sistema difícil de entender:
    - Debe tenerse cuidado con el orden en que se ejecutan los disparadores
    - Puede haber disparadores recursivos o anidados
    - Pueden ser difíciles de optimizar
  - Pero pueden ser muy poderosos:
    - Pueden ser usados para la replicación de datos
    - Ayudan en la auditoria o estadísticas para la modificación de tablas
    - Administración del flujo de trabajo y forzamiento de las reglas del negocio
    - Son usados para crear bases de datos “activas”, de alto rendimiento
-