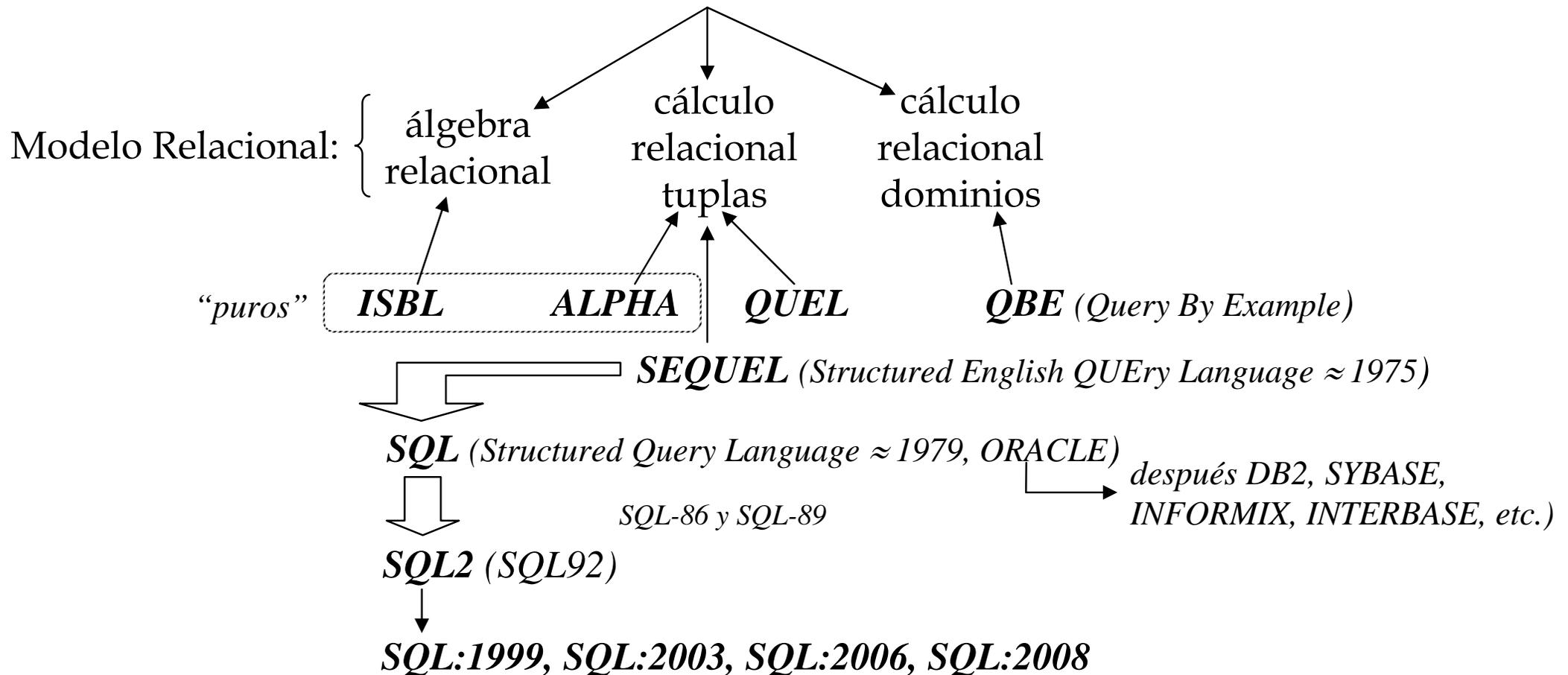


6 LENGUAJES RELACIONALES: SQL

- 6.1 Introducción. Tipos de lenguajes relacionales: SQL
- 6.2 Interrogación y actualización de datos en SQL
- 6.3 Tablas y vistas. Índices. El Diccionario de datos
- 6.4 Otros aspectos de SQL. Protección y acceso
- 6.5 Acceso a la BD desde entornos de programación

6.1 Introducción. Tipos de lenguajes relacionales: SQL

Lenguaje Datos \equiv *semántica (modelo datos) + sintaxis*



presentación de las sentencias básicas de SQL (ORACLE)

| | | |
|-----------------------------------|---|--|
| Lenguaje Descripción Datos (LDD) | $\left\{ \begin{array}{l} \textit{creación} \rightarrow \text{CREATE} \\ \textit{modificación} \rightarrow \text{ALTER} \\ \textit{eliminación} \rightarrow \text{DROP} \end{array} \right\}$ | SCHEMA DOMAIN TABLE VIEW ASSERTION INDEX USER |
| Lenguaje Manipulación Datos (LMD) | $\left\{ \begin{array}{l} \textit{selección} \rightarrow \text{SELECT} \\ \textit{inserción} \rightarrow \text{INSERT} \\ \textit{eliminación} \rightarrow \text{DELETE} \\ \textit{modificación} \rightarrow \text{UPDATE} \end{array} \right\}$ | |
| Lenguaje Control (LC) | $\left\{ \begin{array}{l} \textit{validación transacciones} \rightarrow \text{COMMIT, ROLLBACK} \\ \textit{autorización} \rightarrow \text{GRANT, REVOKE} \end{array} \right\}$ | |

- *no se distingue entre mayúsculas y minúsculas*
- *el final de línea es como un espacio, y las sentencias finalizan con un “;”*
- *comentarios: /* ... */ y desde -- hasta final de línea*

introducción al Lenguaje Definición Datos de SQL

elementos básicos de la estática en SQL ⇒

*definición de **tablas** (relaciones) → lista de **columns** (atributos - dominios + restricciones)*

```
CREATE TABLE nombreTabla (  
    nombreAtrib1 dom1 [restricción1],  
    ...,  
    nombreAtribN domN [restricción1]  
    [, restricciones intrarrelación] );
```

*las **vistas** (también las tablas) se definen en base a una consulta*
└─→ esquema externo y cálculos intermedios

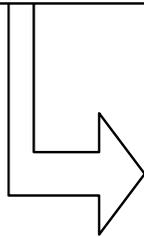
```
CREATE VIEW nombreVista [(nombreAtrib1, ..., nombreAtribM)] AS  
    consulta;
```

ejemplo de definición de tablas en SQL

dominios básicos: char(n); number(n); number(n,m); date;

Ejemplo:

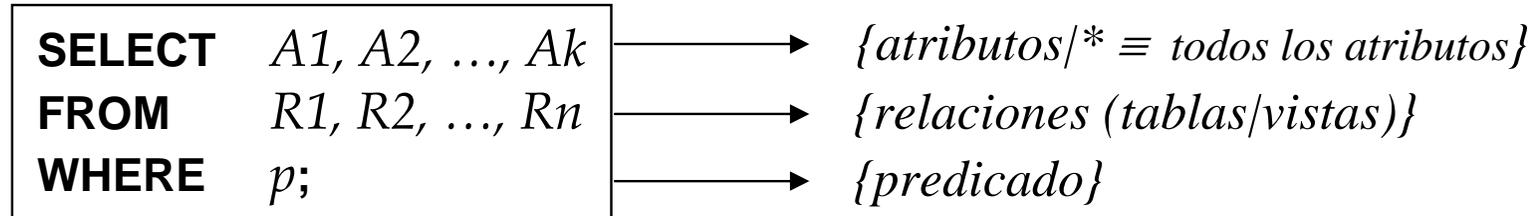
Pieza (clvPieza: entero; nombPieza : tpNombre, NO NULO; color: tpNombre);
Proveedor (clvProv: entero; nombProv: tpNombre, NO NULO);
suministrar (clvProv: entero; clvPieza: entero)
clvProv clave ajena de Proveedor; clvPieza clave ajena de Pieza;



```
CREATE TABLE Pieza (  
  clvPieza NUMBER(9) PRIMARY KEY,  
  nombPieza CHAR(32) NOT NULL,  
  color CHAR(32) );  
  
CREATE TABLE Proveedor (  
  clvProv NUMBER(9) PRIMARY KEY,  
  nombProv CHAR(32) NOT NULL, UNIQUE);  
  
CREATE TABLE suministrar (  
  clvProv NUMBER(9),  
  clvPieza NUMBER(9),  
  PRIMARY KEY (clvProv, clvPieza),  
  FOREIGN KEY (clvProv) REFERENCES Proveedor(clvProv),  
  FOREIGN KEY (clvPieza) REFERENCES Pieza(clvPieza));
```

6.2 Interrogación y actualización de datos en SQL

consulta básica:



cuya semántica es:

$$\Pi_{A_1, A_2, \dots, A_k} (\sigma_p (R_1 \times R_2 \times \dots \times R_n))$$



$$\Pi_{A_1, A_2, \dots, A_k} (R) \longrightarrow \text{SELECT } A_1, A_2, \dots, A_k \text{ FROM } R;$$

$$\sigma_p(R) \longrightarrow \text{SELECT } * \text{ FROM } R \text{ WHERE } p;$$

$$R_1 \times R_2 \longrightarrow \text{SELECT } * \text{ FROM } R_1, R_2;$$

☞ pueden producirse tuplas (filas) repetidas \Rightarrow **SELECT** [ALL | *DISTINCT*] ...

aspectos básicos de la sentencia de consulta **SELECT**

además, operadores relacionales: $\left\{ \begin{array}{l} \text{UNION} \\ \text{MINUS} \\ \text{INTERSECT} \end{array} \right\}$ de consultas (relaciones)

$R1 \cup R2 \longrightarrow \text{SELECT } * \text{ FROM } R1 \text{ UNION SELECT } * \text{ FROM } R2;$

los predicados de la cláusula WHERE se pueden construir en base a:

- *nombres de atributos:* clvProv, nombPieza
 - *valores constantes:* 'TUERCAS', 23
 - *operadores lógicos:* **AND, OR, NOT**
 - *operadores de comparación:* =, !=, <>, >, >=, <, <=
- IS [NOT] NULL**
- [NOT] BETWEEN ... AND ...**
- LIKE ...** \longrightarrow $\left\{ \begin{array}{l} \% \text{ cualquier cadena} \\ _ \text{ cualquier caracter} \end{array} \right.$
- ... [NOT] IN (listaValores)** \longrightarrow o subconsulta
- opRel ANY (listaValores)** \longrightarrow \uparrow
- opRel ALL (listaValores)** \longrightarrow \uparrow
- (Note: A dotted arrow points from the 'opRel' labels to the 'operadores de comparación' bullet point.)*

ejemplos de consultas sencillas sobre una tabla (1)

ejemplos:

① *Piezas de color 'verde'*

```
SELECT *  
FROM Pieza  
WHERE color = 'VERDE';
```

② *Nombres de todas las piezas*

```
SELECT DISTINCT nombPieza  
FROM Pieza;
```

③ *Nombres de todas las piezas
que empiezan por 'T'*

```
SELECT DISTINCT nombPieza  
FROM Pieza  
WHERE nombPieza LIKE 'T%';
```

④ *Piezas que son
'TUERCA' o 'TORNILLO'*

```
SELECT DISTINCT *  
FROM Pieza  
WHERE nombPieza = 'TUERCA' OR nombPieza = 'TORNILLO' ;
```

ejemplos de consultas sencillas sobre una tabla (2)

④ *Piezas que son
'TUERCA' o 'TORNILLO'*

```
SELECT DISTINCT *  
FROM Pieza  
WHERE nombPieza = 'TUERCA'  
UNION  
SELECT DISTINCT *  
FROM Pieza  
WHERE nombPieza = 'TORNILLO';
```

④ *Piezas que son
'TUERCA' o 'TORNILLO'*

```
SELECT DISTINCT clvPieza "id Pieza", nombPieza nombre, color  
FROM Pieza  
WHERE nombPieza IN ('TUERCA', 'TORNILLO');
```

④ *Piezas que son
'TUERCA' o 'TORNILLO'*

```
SELECT DISTINCT *  
FROM Pieza  
WHERE nombPieza = ANY ('TUERCA', 'TORNILLO');
```

*para cambiar
la cabecera*

= ANY *equivale a* IN
<> ALL *equivale a* NOT IN

ejemplos de consultas sencillas sobre varias tablas (1)

consultas sobre varias tablas: → varias tablas en la cláusula FROM
y en cláusula WHERE la condición de JOIN → en SQL92 existen estos operadores

✓ los atributos deben quedar perfectamente identificados → *tabla.atributo*

✓ se pueden utilizar “alias” para los nombres de las tablas: *tabla alias*

*tabla AS alias
en el estándar*

⑤ *Nombre de los proveedores que suministran al menos una pieza de color ‘verde’*

```
SELECT DISTINCT nombProv "proveedor", 'suministra piezas de color verde' "función"  
FROM Proveedor V, Suministrar S, Pieza P  
WHERE V.clvProv = S.clvProv AND S.clvPieza = P.clvPieza AND P.color = 'VERDE';
```

⑥ *Piezas de las que no se conoce el color*

```
SELECT * FROM Pieza WHERE color IS NULL;
```

ejemplos de consultas sencillas sobre varias tablas (2)

- ⑦ *Piezas disponibles en varios colores*

```
SELECT DISTINCT P1.clvPieza, P1.nombPieza, P1.color
FROM      Pieza P1, Pieza P2
WHERE     P1.nombPieza = P2.nombPieza AND P1.color <> P2. color;
```

- ⑧ *Proveedores que suministran 'tuercas' y 'tornillos'*

```
SELECT DISTINCT S1.clvProv
FROM      Suministrar S1, Pieza P1, Suministrar S2, Pieza P2
WHERE     S1.clvPieza = P1.clvPieza AND P1.nombPieza = 'TUERCA'
           AND S2.clvPieza = P2.clvPieza AND P2.nombPieza = 'TORNILLO'
           AND S1.clvProv = S2. clvProv;
```

o también

```
SELECT DISTINCT clvProv FROM Suministrar S, Pieza P
WHERE S.clvPieza = P.clvPieza AND P.nombPieza = 'TUERCA'
INTERSECT
SELECT DISTINCT clvProv FROM Suministrar S, Pieza P
WHERE S.clvPieza = P.clvPieza AND P.nombPieza = 'TORNILLO';
```

☞ puede resultar interesante utilizar vistas para resultados intermedios:

ejemplos de consultas sencillas sobre varias tablas (3)

⑨ Clave y nombre de los proveedores que no suministran piezas

```
CREATE VIEW noSuministran AS
  SELECT clvProv FROM Proveedor
MINUS
  SELECT clvProv FROM Suministrar;
```

```
SELECT *
FROM Proveedor V, noSuministran N
WHERE V.clvProv = N.clvProv;
```

DROP VIEW noSuministran;

☞ también se puede incluir subconsultas en la cláusula WHERE

⑨ Clave y nombre de los proveedores que no suministran piezas

```
SELECT *
FROM Proveedor
WHERE clvProv NOT IN (SELECT clvProv FROM Suministrar);
```

↔ ALL

ejemplos de consultas sencillas sobre varias tablas (4)

- ☛ las subconsultas pueden utilizar atributos de las consultas en las que están incluídas (correladas)
- ☛ un predicado interesante en las subconsultas es: **[NOT] EXISTS**

obtención
más costosa

⑨ *Clave y nombre de los proveedores que no suministran piezas*

```
SELECT *  
FROM Proveedor V  
WHERE NOT EXISTS (SELECT * FROM Suministrar WHERE clvProv = V.clvProv);
```

⑩ *Clave y nombre de los proveedores que no suministran tuercas verdes*

```
SELECT * FROM Proveedor V  
WHERE NOT EXISTS (SELECT * FROM Pieza P  
WHERE (nombPieza, color) IN (('TUERCA', 'VERDE'))  
AND EXISTS (SELECT * FROM Suministrar  
WHERE clvProv = V.clvProv AND clvPieza = P.clvPieza));
```

→ también como: todos menos los que suministran alguna tuerca verde

ejemplos de consultas sencillas sobre varias tablas (5)

☞ el interés mayor es para consultas con cuantificador “todas”

- ⑩ *Clave y nombre de los proveedores que suministran tuercas de todos los tipos es decir, los proveedores tales que no hay ningún tipo de tuerca que no suministren*

```
SELECT * FROM Proveedor V
WHERE NOT EXISTS (SELECT * FROM Pieza P
                  WHERE P.nombPieza = 'TUERCA'
                  AND NOT EXISTS (SELECT * FROM Suministrar
                                  WHERE clvProv = V.clvProv AND clvPieza = P.clvPieza));
```

También se podría implementar la división:

$$R1 \div R2 = \Pi_{A'}(R1) - \Pi_{A'}((\Pi_{A'}(R1) \times R2) - R1)$$

$$R1 = \Pi_{clvPieza} (\sigma_{nombPieza = 'TUERCA'} (Pieza)) \equiv \text{piezas que son tuercas}$$

$$R5 = Suministrar \div R1 \equiv \text{suministradores de todas las piezas de R1}$$

$$R6 = \Pi_{nombProv} (Proveedor \bowtie R5)$$

ejemplos de consultas sencillas sobre varias tablas (6)

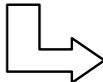
- ⑩ Clave y nombre de los proveedores que suministran tuercas de todos los tipos

```
CREATE VIEW noSumTuercas AS
  SELECT clvProv, Pieza.clvPieza FROM Proveedor, Pieza WHERE nombPieza = 'TUERCA'
MINUS
  SELECT clvProv, P.clvPieza FROM Suministrar S, Pieza P
  WHERE S.clvPieza = P.clvPieza AND nombPieza = 'TUERCA' ;
```

```
SELECT clvProv, nombProv FROM Proveedor
MINUS
SELECT P.clvProv, nombProv FROM Proveedor P, noSumTuercas N WHERE P.clvProv = N.clvProv;
```

DROP VIEW noSumTuercas ;

obsérvese que no se ha implementado exactamente la división, sino que se ha simplificado

 *propóngase una transformación para el caso general*

ordenación y operaciones simples de agrupación

➡ se puede utilizar la cláusula **ORDER BY** para mostrar el resultado ordenado

⑪ *Clave y nombre de los proveedores, ordenados por nombre*

```
SELECT * FROM Proveedor  
ORDER BY nombProv;
```

```
SELECT * FROM Proveedor  
ORDER BY 2, DESC;
```

→ *orden inverso*

➡ se pueden realizar operaciones (aritméticas, contar, max., etc.) sobre las tuplas obtenidas

```
ABS(n)  
MOD (m,n)  
ROUND(n [,m])  
TRUNC(n [,m])  
POWER(m,n)  
SQRT(n)  
...
```

```
CHR(n)  
ASCII (char)  
LOWER(char)  
UPPER(char)  
LENGTH(char)  
SUBSTR(char, m[,n])  
...
```

```
COUNT ([DISTINCT | ALL] expr.)  
COUNT (*)  
MAX (expr.)  
MIN (expr.)  
SUM (expr.)  
AVG (expr.)  
...
```

```
TO_CHAR (expr. [fmt])  
TO_NUMBER (char)  
TO_DATE (char [fmt])  
...
```

...

ejemplos de consultas con agrupaciones simples

⑫ *Total de tipos de piezas existentes*

```
SELECT count(DISTINCT nombPieza) "tipos de piezas" FROM Pieza;
```

⑬ *Total de proveedores*

```
SELECT count(*) "tot. proveedores" FROM Proveedor;
```

⑭ *Mayor valor de clvPieza*

```
SELECT max(clvPieza) "max. clvPieza" FROM Pieza;
```

⑮ *clave y nombre de las piezas con clave impar, y el último dígito de la clave*

```
SELECT clvPieza, MOD(clvPieza, 10) "term.", nombPieza "nombre"  
FROM Pieza  
WHERE mod(clvPieza, 2) = 1;
```

operaciones de agrupación de tuplas

➡ se utiliza la cláusula **GROUP BY** para realizar un tratamiento sobre grupos de tuplas

①⑥ *Nombres de las piezas y cantidad de cada tipo*

```
SELECT nombPieza "tipo", count(clvPieza) "total" FROM Pieza
GROUP BY nombPieza;
```

①⑦ *Colores de las piezas y número de piezas de cada color* → *Pb.: puede haber nulos*

```
SELECT color, count(*) "total", count(color) "colores" FROM Pieza
GROUP BY color;
```

*color NULL es un grupo ⇒
count (*) mostrará el total
de piezas con color NULL,
pero count (color) será 0*

①⑧ *Número medio de piezas por color (incluyendo desconocidos)*

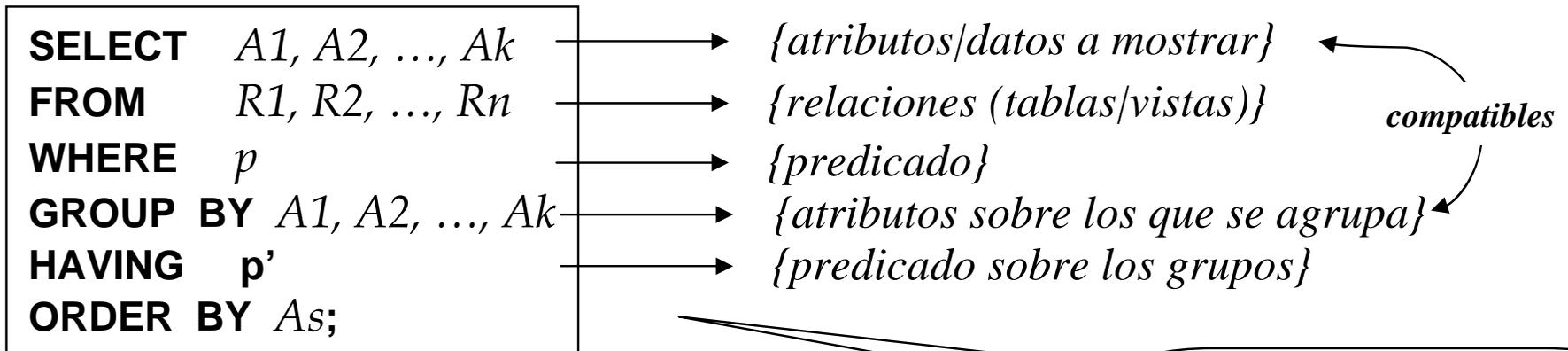
```
SELECT AVG(COUNT(*)) "piezas/color (incluido nulo)" FROM Pieza
GROUP BY color;
```

→ *el valor nulo constituye un grupo*

➡ *en los cálculos (media, max, etc.), los nulos no se cuentan (puede depender del SGBD)*

consultas con predicados sobre las agrupaciones (1)

➡ además, se pueden añadir condiciones de selección sobre los grupos



① *Número medio de piezas de cada color (sólo los conocidos)*

```
SELECT AVG(COUNT(*)) "piezas/color (conocidos)" FROM Pieza
GROUP BY color
HAVING color IS NOT NULL;
```

pasos en la ejecución de una sentencia **SELECT**

- ① Obtención de las filas (tuplas) especificadas en el **FROM** (*producto cartesiano*)
- ② Selección de las tuplas que verifican el predicado especificado en el **WHERE**
- ③ Agrupación de las tuplas por los atributos especificados en **GROUP BY**
- ④ Selección de los grupos de tuplas que verifican el predicado especificado en el **HAVING**
- ⑤ Selección de columnas especificadas en el **SELECT** con ejecución de las operaciones especificadas sobre los grupos (contar, max. etc.) \Rightarrow cada grupo queda como una tupla
- ⑥ Si es **SELECT DISTINCT**, eliminar las tuplas duplicadas
- ⑦ Ordenación de las tuplas según especifica **ORDER BY**

consultas con predicados sobre las agrupaciones (2)

- ②① Nombre (y cantidad) de los tipos de piezas de las que hay más de 1, ordenadas por nombre

```
SELECT nombPieza "tipo", count(clvPieza) "total" FROM Pieza
GROUP BY nombPieza
HAVING count(clvPieza) > 1
ORDER BY nombPieza ASC ;
```

- ②① Colores de las piezas y número de piezas de cada color, pero sólo de los colores con menos piezas que la media, ordenados alfabéticamente

```
SELECT color, COUNT(*) "total" FROM Pieza
GROUP BY color
HAVING count(*) < ( SELECT AVG(count(*) ) FROM Pieza GROUP BY color )
ORDER BY color ;
```

implementación del operador JOIN externo

☛ también hay operadores de JOIN EXTERNO (depende del gestor)

↳ (+) a la dcha del atributo con valores nulos,
dentro del WHERE en la expresión de JOIN

②② clave y nombre de todos los proveedores, junto con las piezas que suministran

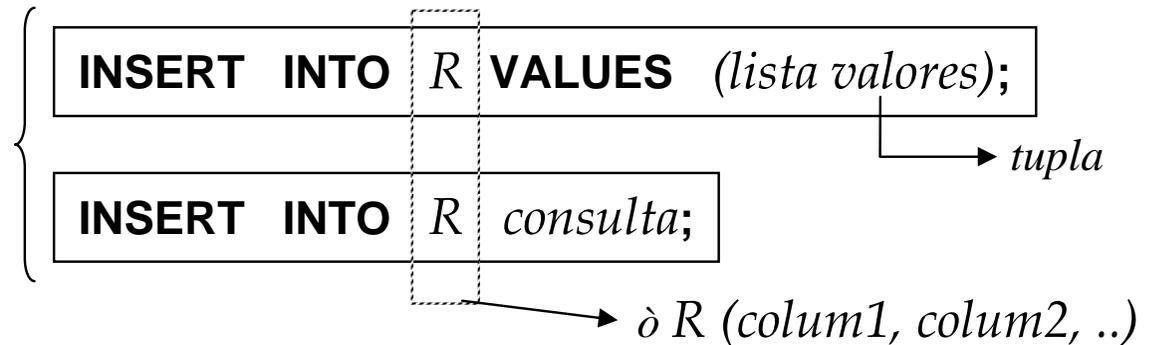
```
SELECT V.clvProv "id.", nombProv "nombre", P.clvPieza "REF.", nombPieza "Pieza", color
FROM Proveedor V, Suministrar S, Pieza P
WHERE V.clvProv = S.clvProv (+) AND S.clvPieza = P.clvPieza (+);
```

②③ clave y nombre de todos los proveedores, junto con las piezas que suministran

```
SELECT V.clvProv, nombProv, P.clvPieza, nombPieza, color
FROM Proveedor V, Suministrar S, Pieza P
WHERE V.clvProv = S.clvProv AND S.clvPieza = P.clvPieza
UNION
SELECT clvProv, nombProv, TO_NUMBER(NULL), NULL, NULL
FROM Proveedor
WHERE clvProv NOT IN (SELECT clvProv FROM Suministrar);
```

6.2 operaciones de actualización en SQL: inserción

➔ operador de adición de tuplas **INSERT**



②④ añadir las piezas: “tuerca amarilla”, de clave 99, y “martillo”, de clave 98

```
INSERT INTO Pieza VALUES (99, 'TUERCA', 'AMARILLO');  
INSERT INTO Pieza VALUES (98, 'MARTILLO', '');
```

②⑤ añadir como piezas la lista de proveedores (aunque no tenga sentido)

```
INSERT INTO Pieza SELECT clvprov, nombprov, TO_CHAR(NULL) FROM Proveedor;
```

operaciones de actualización en SQL: eliminación

➡ operador de eliminación de tuplas **DELETE**

```
DELETE [ FROM ] R [ WHERE condición ];
```

②⑥ *eliminar todas las tuercas*

```
DELETE FROM Pieza WHERE nombPieza = 'TUERCA';
```

②⑦ *eliminar todos los suministros de piezas*

```
DELETE FROM Suministrar;
```

②⑧ *eliminar todos los tipos de piezas de las que sólo hay una pieza*

```
DELETE FROM Pieza P  
WHERE 2 > (SELECT count(*) FROM Pieza WHERE nombPieza = P. nombPieza);
```

➡ *puede haber problema con las restricciones (también en la inserción y eliminación)*

→ *una solución: inhibición temporal*

operaciones de actualización en SQL: modificación

➡ operador de modificación de tuplas **UPDATE**

```
UPDATE R SET col1 = expr1. [,col2 = expr2.] ... [WHERE condición ];
```

```
UPDATE R SET (col1 [,col2 ] ...) = consulta [WHERE condición ];
```

②⑨ *cambiar el color rojo por rojizo*

```
UPDATE Pieza SET color = 'ROJIZO' WHERE color = 'ROJO' ;
```

③⑩ *poner a la pieza 91 el color de la pieza 95*

```
UPDATE Pieza SET color = (SELECT color FROM Pieza WHERE clvPieza=95)  
WHERE clvPieza=91 ;
```

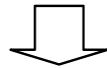
③① *cambiar a “desconocido” el color de las piezas cuyo color es desconocido (nulo)*

```
UPDATE Pieza SET color = 'DESCONOCIDO' WHERE color IS NULL ;
```

6.3 Tablas y vistas. Índices. El Diccionario de datos

especificación de una tabla (relación) \Rightarrow

- columnas (*Atributo-Dominio + restricciones*)
- restricciones intrarrelación
- aspectos nivel físico



CREATE TABLE nombre (....);

dominios básicos (ORACLE):

| | |
|--------------|--|
| char(n); | <i>cadena longitud fija (n caracteres)</i> |
| varchar(n); | <i>cadena longitud variable (hasta n caracteres)</i> |
| number(n); | <i>enteros de hasta n dígitos significativos</i> |
| number(n,m); | <i>números de hasta n dígitos significativos y m decimales</i> |
| date; | |

en estándar ANSI, hay más tipos: integer, float, etc.

restricciones columna:

PRIMARY KEY
NOT NULL
UNIQUE
REFERENCES *column* [ON DELETE CASCADE]
CHECK (*expr.*)

ò **SET NULL**

especificación de restricciones intrarrelación

restricciones tabla:

PRIMARY KEY (col1, ..., colk)
UNIQUE (col1, ..., colk)
FOREIGN KEY (col1, ..., colk) REFERENCES espec.Colum [ON DELETE CASCADE]
CHECK (expr.)

o SET NULL

☞ pueden ir precedidas de indicación de restricción: **CONSTRAINT nombre**
↳ útil para eliminarlas, desactivarlas, etc.

aspectos físicos: indicación de TABLESPACE, CLUSTER, .., ocupación, crecimiento, ..., etc.

ejemplo sencillo de especificación de tablas

ejemplos:

```
CREATE TABLE Pieza (  
  clvPieza    NUMBER(9) CONSTRAINT PK_Pieza      PRIMARY KEY,  
  nombPieza  CHAR(32)  CONSTRAINT NN_nombPieza NOT NULL  
                                     CONSTRAINT UP_nombPieza CHECK(nombPieza = UPPER(nombPieza)),  
  color      CHAR(32)  CONSTRAINT UP_color      CHECK(color = UPPER(color));  
  
CREATE TABLE Proveedor (  
  clvProv    NUMBER(9) CONSTRAINT PK_Proveedor PRIMARY KEY,  
  nombProv  CHAR(32)  CONSTRAINT NN_nombProv  NOT NULL  
                                     CONSTRAINT UN_nombProv UNIQUE  
                                     CONSTRAINT UP_nombProv CHECK(nombProv = UPPER(nombProv));  
  
CREATE TABLE suministrar (  
  clvProv    NUMBER(9),  
  clvPieza   NUMBER(9),  
  CONSTRAINT PK_Suministrar PRIMARY KEY (clvProv, clvPieza),  
  CONSTRAINT FK_SuminProv  FOREIGN KEY (clvProv) REFERENCES Proveedor(clvProv)  
                                     ON DELETE CASCADE,  
  CONSTRAINT FK_SuminPieza FOREIGN KEY (clvPieza) REFERENCES Pieza(clvPieza)  
                                     ON DELETE CASCADE);
```

otras operaciones con tablas: ejemplos

☞ también pueden ser creadas a partir de otra tabla existente:

```
CREATE TABLE PiezasRojas (refPieza, tipoPieza) AS
SELECT clvPieza, nombPieza FROM Pieza WHERE color = 'ROJO';
```

☞ para eliminar tablas:

```
DROP TABLE nombre
```

☞ para modificar tablas:

```
ALTER TABLE nombre ...
```

ejemplos:

```
DROP TABLE PiezasRojas;
ALTER TABLE Pieza MODIFY nombPieza CONSTRAINT NN_nombPieza NOT NULL;
ALTER TABLE Pieza DROP CONSTRAINT NN_nombPieza;
ALTER TABLE Suministrar ADD (cantidad NUMBER(4) CONSTRAINT NN_cantidad NOT NULL);
ALTER TABLE Suministrar DISABLE CONSTRAINT FK_SuminProv;
ALTER TABLE Suministrar ENABLE CONSTRAINT FK_SuminProv;
```

→ no permite eliminar columnas (ORACLE)

operaciones con Vistas

son tablas virtuales (calculadas, no almacenadas), especificadas en base a una consulta:

Ventajas: {
✓ **Seguridad** ≡ el usuario final sólo percibe una parte de la información
✓ **Utilidad** ≡ simplifica la construcción de consultas complejas (+*legibilidad*)

```
CREATE VIEW nombre [(col1,..., colk)] AS consulta;
```

ejemplos:

```
CREATE VIEW PiezasRojas (refPieza, tipoPieza) AS  
SELECT clvPieza, nombPieza FROM Pieza WHERE color = 'ROJO';
```

```
DROP VIEW nombre;
```

➡ *fuertes restricciones en las actualizaciones (eliminación, inserción y modificación)*

otros elementos de SQL: sinónimos e índices

☛ también es útil especificar *sinónimos* de “objetos”:

```
CREATE SYNONYM misPiezas FOR Santiago.Pieza;  
DROP SYNONYM misPiezas;
```

☛ para “optimizar” el acceso se pueden especificar **INDICES** sobre columnas

↳ puede penalizar las operaciones de actualización

ejemplo:

```
CREATE INDEX indx_clvPieza ON Pieza (clvPieza);  
DROP INDEX indx_clvPieza;
```

↳ lista de columnas

el Diccionario de datos

☛ *es el elemento básico de una Base de Datos* ≡ *donde se almacenan todas las especificaciones de la B.D.*

☛ *en el modelo relacional, también está implementado en base a tablas*



sólo las modifica el SGBD

```
SELECT * FROM DICTIONARY;
```

→ *Tabla con: (table_name, comments)*

ejemplo de diccionario de datos (1)

| TABLE_NAME | COMMENTS |
|----------------------|---|
| ALL_ALL_TABLES | <i>Description of all object and relational tables accessible to the user</i> |
| ALL_ARGUMENTS | <i>Arguments in object accessible to the user</i> |
| ALL_ASSOCIATIONS | <i>All associations available to the user</i> |
| ALL_CATALOG | <i>All tables, views, synonyms, sequences accessible to the user</i> |
| • • • | |
| ALL_VIEWS | <i>Description of views accessible to the user</i> |
| USER_ALL_TABLES | <i>Description of all object and relational tables owned by the user's</i> |
| USER_ARGUMENTS | <i>Arguments in object accessible to the user</i> |
| USER_ASSOCIATIONS | <i>All associations defined by the user</i> |
| USER_AUDIT_OBJECT | <i>Audit trail records for statements concerning objects, specifically: table, clus</i> |
| USER_AUDIT_SESSION | <i>All audit trail records concerning CONNECT and DISCONNECT</i> |
| USER_AUDIT_STATEMENT | <i>Audit trail records concerning grant, revoke, audit, noaudit and alter system</i> |
| USER_AUDIT_TRAIL | <i>Audit trail entries relevant to the user</i> |
| USER_CATALOG | <i>Tables, Views, Synonyms and Sequences owned by the user</i> |
| • • • | |
| USER_INDEXES | <i>Description of the user's own indexes</i> |
| • • • | |
| USER_SYNONYMS | <i>The user's private synonyms</i> |
| USER_SYS_PRIVS | <i>System privileges granted to current user</i> |
| USER_TABLES | <i>Description of the user's own relational tables</i> |
| USER_TABLESPACES | <i>Description of accessible tablespaces</i> |
| USER_TAB_COLUMNS | <i>Columns of user's tables, views and clusters</i> |

ejemplo de diccionario de datos (2)

| | |
|------------------------|--|
| USER_TAB_HISTOGRAMS | <i>Histograms on columns of user's tables</i> |
| USER_TAB_MODIFICATIONS | <i>Information regarding modifications to tables</i> |
| USER_TAB_PARTITIONS | |
| USER_TAB_PRIVS | <i>Grants on objects for which the user is the owner, grantor or grantee</i> |
| USER_TAB_PRIVS_MADE | <i>All grants on objects owned by the user</i> |
| USER_TAB_SUBPARTITIONS | |
| USER_TRIGGERS | <i>Triggers owned by the user</i> |
| • • • | |
| DICTIONARY | <i>Description of data dictionary tables and views</i> |
| DICT_COLUMNS | <i>Description of columns in data dictionary tables and views</i> |
| DUAL | |
| CAT | <i>Synonym for USER_CATALOG</i> |
| CLU | <i>Synonym for USER_CLUSTERS</i> |
| COLS | <i>Synonym for USER_TAB_COLUMNS</i> |
| DICT | <i>Synonym for DICTIONARY</i> |
| • • • | |
| IND | <i>Synonym for USER_INDEXES</i> |
| OBJ | <i>Synonym for USER_OBJECTS</i> |
| SEQ | <i>Synonym for USER_SEQUENCES</i> |
| SYN | <i>Synonym for USER_SYNONYMS</i> |
| TABS | <i>Synonym for USER_TABLES</i> |
| • • • | |

359 rows selected.

consultas sobre el diccionario de datos

se puede consultar para obtener información de la BD

```
SELECT TABLE_NAME FROM USER_TABLES; —————> tablas definidas por el usuario
```

```
SELECT * FROM CAT; —————> catálogo del usuario
```

```
SELECT OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME, STATUS  
FROM USER_CONSTRAINTS; —————> restricciones definidas por el usuario
```

```
SELECT OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME, STATUS  
FROM USER_CONSTRAINTS  
WHERE TABLE_NAME = 'Pieza'; —————> restricciones definidas por el usuario para la tabla Pieza
```

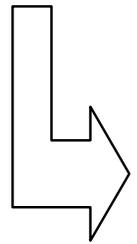
mostrar todas las claves primarias definidas por el usuario

```
SELECT b.OWNER, a.CONSTRAINT_NAME, a.TABLE_NAME, b.COLUMN_NAME  
FROM USER_CONSTRAINTS A, ALL_CONS_COLUMNS B  
WHERE (b.CONSTRAINT_NAME = a.CONSTRAINT_NAME) AND (a.CONSTRAINT_TYPE = 'P');
```

6.4 Protección y acceso

☛ *las vistas limitan la capacidad de acceso a la información pero, además*

una función básica de un SGBD es controlar el acceso a la información de las B.D.

- 
- ✓ *control del acceso al S.G.B.D. ⇒ **autenticación** del usuario*
 - ✓ *control del acceso a los objetos ⇒ **autorizaciones** del usuario sobre objetos*
 - ✓ *control de la capacidad de deducción sobre las consultas . . .*

para crear un usuario → especificar la autorización correspondiente de acceso

GRANT privilegios TO usuario IDENTIFIED BY password;

→ **DBA, RESOURCE, CONNECT**

ejemplo: **GRANT RESOURCE TO Pedro IDENTIFIED BY secreto;**

ejemplo de operaciones de autorización

para especificar las autorizaciones (privilegios) sobre los objetos de la B.D.:

```
GRANT privilegios | ALL ON objeto TO usuario [WITH GRANT OPTION];
```

→ { *tablas:* ALTER, DELETE, INDEX, INSERT,
REFERENCES, SELECT, UPDATE
vistas: DELETE, INSERT, SELECT, UPDATE
• • •

ejemplos:

```
GRANT REFERENCES (clvPieza), UPDATE (clvPieza, nombPieza) ON Juan.Pieza TO Pedro ;  
GRANT SELECT, UPDATE ON misPiezas TO PUBLIC ;  
GRANT ALL ON suministrar TO Pedro WITH GRANT OPTION;
```

para desautorizar:

```
REVOKE privilegios FROM usuario;
```

→ **REVOKE RESOURCE FROM Pedro;**

```
REVOKE privilegios ON objeto FROM usuario;
```

→ **REVOKE ALL ON Suministrar FROM Pedro;**

gestión de transacciones

➤ *además el gestor dispone de mecanismos de protección de la información (para el DBA):*

→ *restricciones de integridad, etc. + redundancias físicas, backups, etc.*

➤ *la operación + simple con verificación de consistencia es la **transacción***

validación de transacciones pendientes:

COMMIT;

establecer un pto. de “salv guarda”:

SAVEPOINT nombre;

deshacer transacciones pendientes:

ROLLBACK;

ROLLBACK TO savePoint;

otros aspectos de SQL: disparadores (1)

TRIGGERS (disparadores): *especifican la respuesta del SGBD frente a los eventos indicados*

- ✓ *se pueden utilizar para implementar restricciones de integridad*
- ✓ *proporcionan comportamiento activo (automático) a la B.D.*

ejemplos:

```
CREATE or REPLACE TRIGGER ContabHorasProy
AFTER UPDATE ON Participar
FOR EACH ROW
WHEN (NEW.numHoras > 0)
begin
  UPDATE Proyecto
  SET   horas = horas + :NEW.numHoras - :OLD.numHoras
  WHERE numProy = :NEW.numProy;
end ContabHorasProy;
```

*SQL + lenguaje del
SGBD (PL/SQL)*

*añade al proyecto las
horas trabajadas por
un empleado*

☞ *están almacenados en la Base de Datos*

otros aspectos de SQL: disparadores (2)

```
CREATE OR REPLACE TRIGGER GuardarHistorialSalario
BEFORE UPDATE ON Empleado
FOR EACH ROW
begin
  if (:OLD.Salario <> :NEW.salario)
  then INSERT INTO HistorialSalario VALUES (:OLD.codEmp, :OLD.salario, sysdate);
  end if;
end GuardarHistorialSalario;
```

refleja en HistorialSalario el cambio de sueldo de un empleado

para restricciones se pueden utilizar los mecanismos de excepción disponibles:

```
raise_application_error (-20201, 'Causa del error');      (ORACLE)
```

otros aspectos de SQL: procedimientos y funciones

Procedimientos y Funciones:

especifican “nuevas operaciones” definidas por el usuario

- ✓ simplifican el desarrollo de aplicaciones complejas
- ✓ incrementan la eficiencia del SGBD

ejemplo:

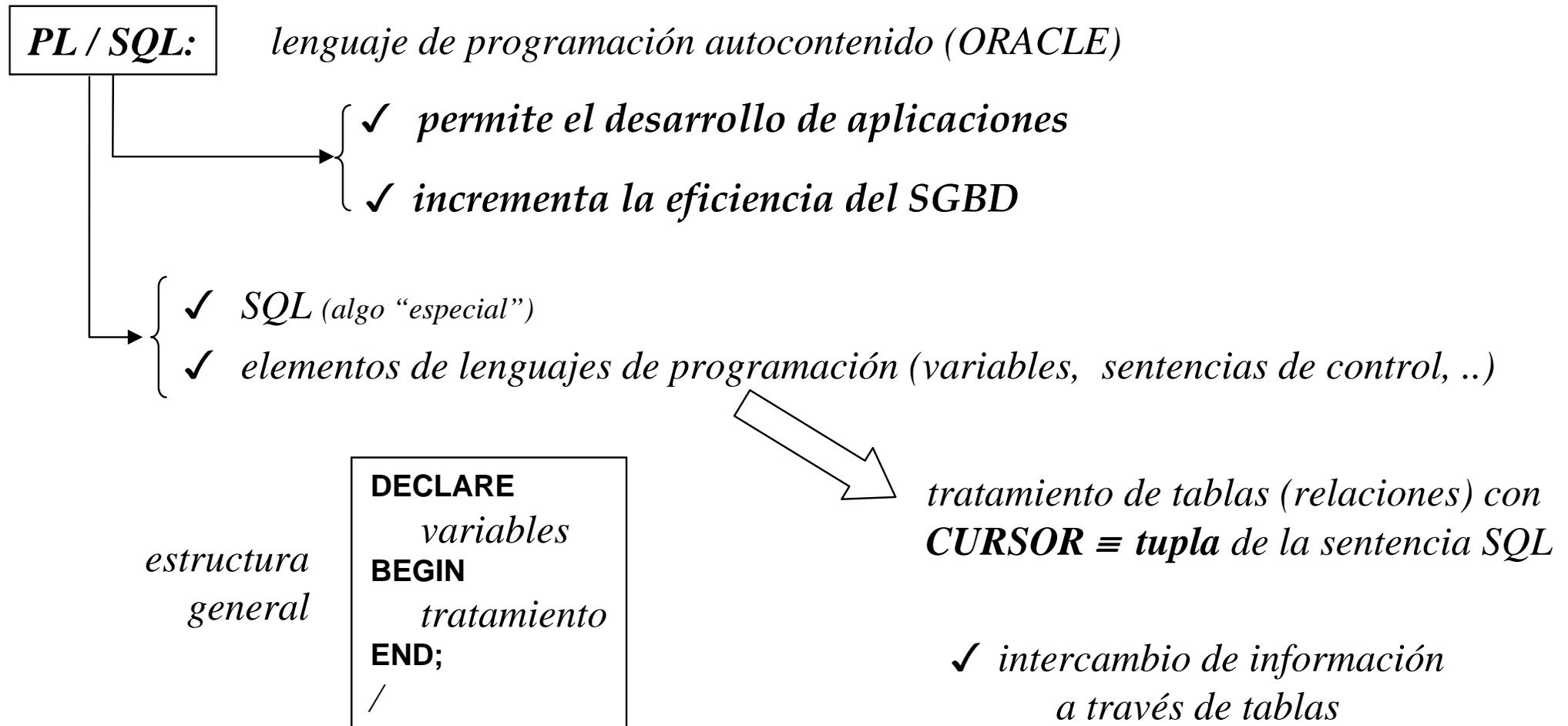
```
CREATE OR REPLACE PROCEDURE Currar (  
  emp    IN Participar.codemp%type,  
  proy   IN Participar.numproy%type,  
  horas  IN Participar.numhoras%type)  
AS  
  begin  
    UPDATE Participar  
    SET    numHoras = numHoras + horas  
    WHERE emp = codemp and proy = numproy;  
  end;
```

SQL + lenguaje del
SGBD (PL/SQL)

☞ *están almacenados en la Base de Datos*

además, • • •

desarrollo de aplicaciones: introducción al PL/SQL (ORACLE)



➡ *los elementos del lenguaje son “similares” a SQL embebido*

ejemplo de programación en PL/SQL (1)

✓ seleccionar los proveedores que suministran más de una pieza

CREATE TABLE temp (refProv number, totPiezas number); → creación de una tabla temporal para resultado

DECLARE

laRefProv number, numPiezas number,

CURSOR selProv **IS SELECT distinct** clvProv **FROM** Suministrar;

BEGIN

OPEN selProv;

LOOP

FETCH selProv **INTO** laRefProv;

EXIT WHEN selProv%NOTFOUND;

SELECT count (**distinct** clvPieza) **INTO** numPiezas
FROM Suministrar
WHERE clvProv = laRefProv;

IF numPiezas > 1 **THEN INSERT INTO** temp **VALUES** (laRefProv, numPiezas); **END IF;**

END LOOP;

CLOSE selProv;

END;

;

SELECT * FROM temp;

DROP TABLE temp;

ejemplo de programación en PL/SQL (2)

✓ contar los proveedores que suministran más de una pieza

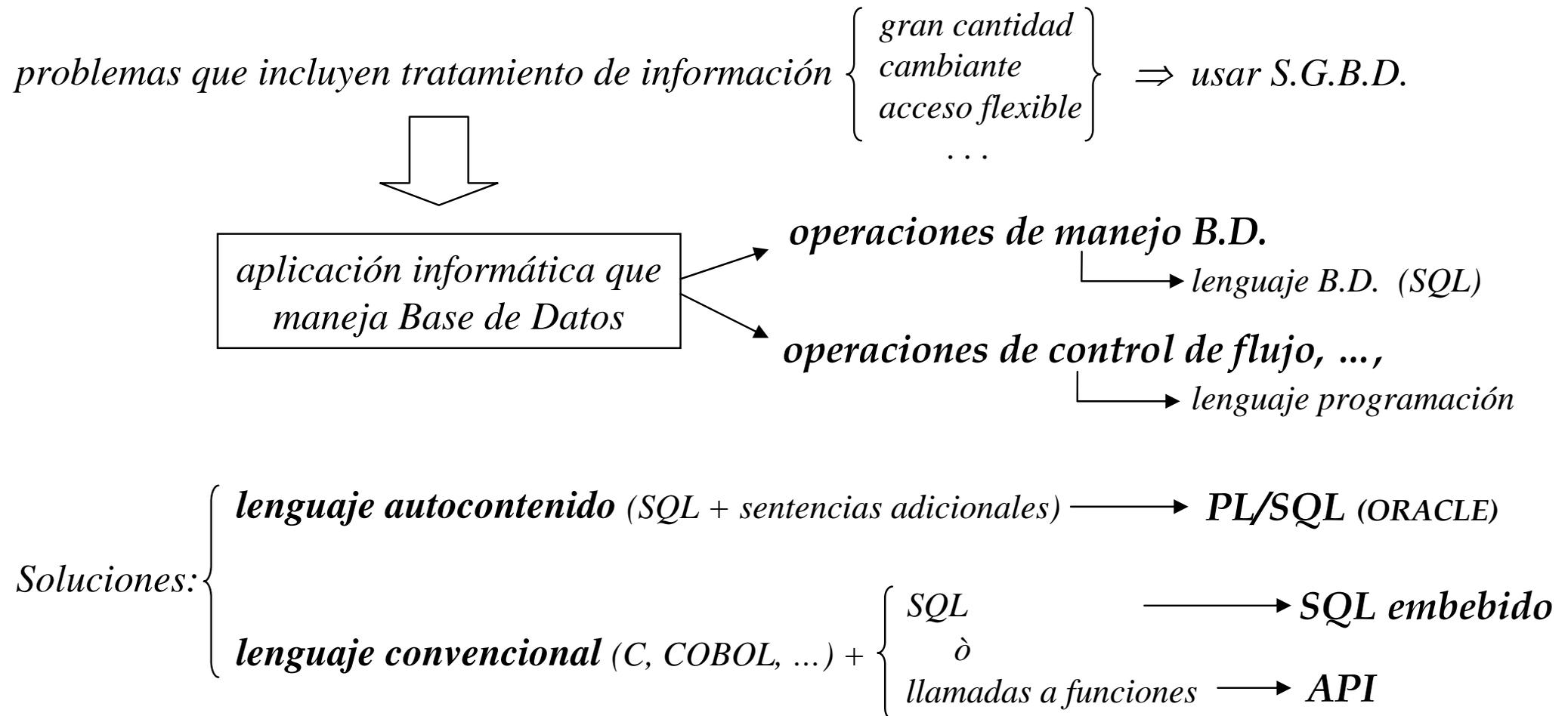
VARIABLE contador NUMBER; —————> variable para resultado

```
DECLARE
  laRefProv number, numPiezas number,
  CURSOR selProv IS SELECT distinct clvProv FROM Suministrar;
BEGIN
  :contador := 0;
  OPEN selProv;
  LOOP
    FETCH selProv INTO laRefProv;
    EXIT WHEN selProv%NOTFOUND;
    SELECT count (distinct clvPieza) INTO numPiezas
    FROM Suministrar
    WHERE clvProv = laRefProv;
    IF numPiezas > 1 THEN :contador := :contador + 1; END IF;
  END LOOP;
  CLOSE selProv;
END;
```

```
;
```

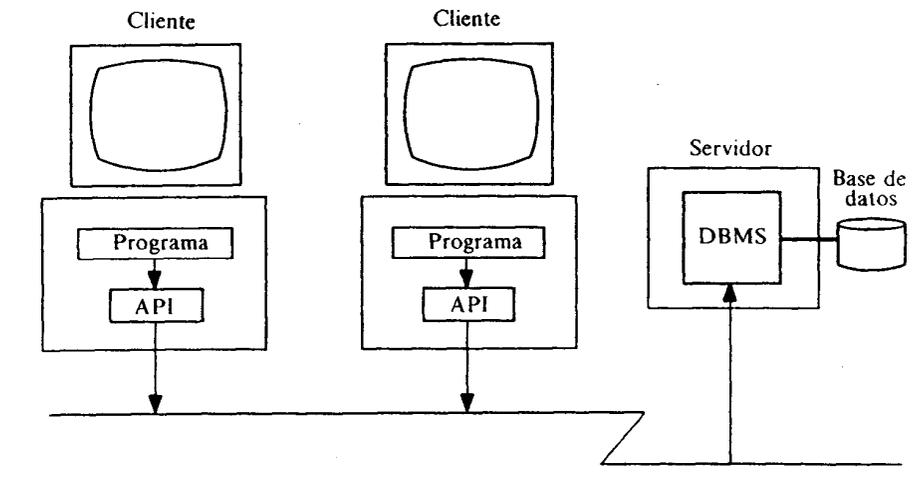
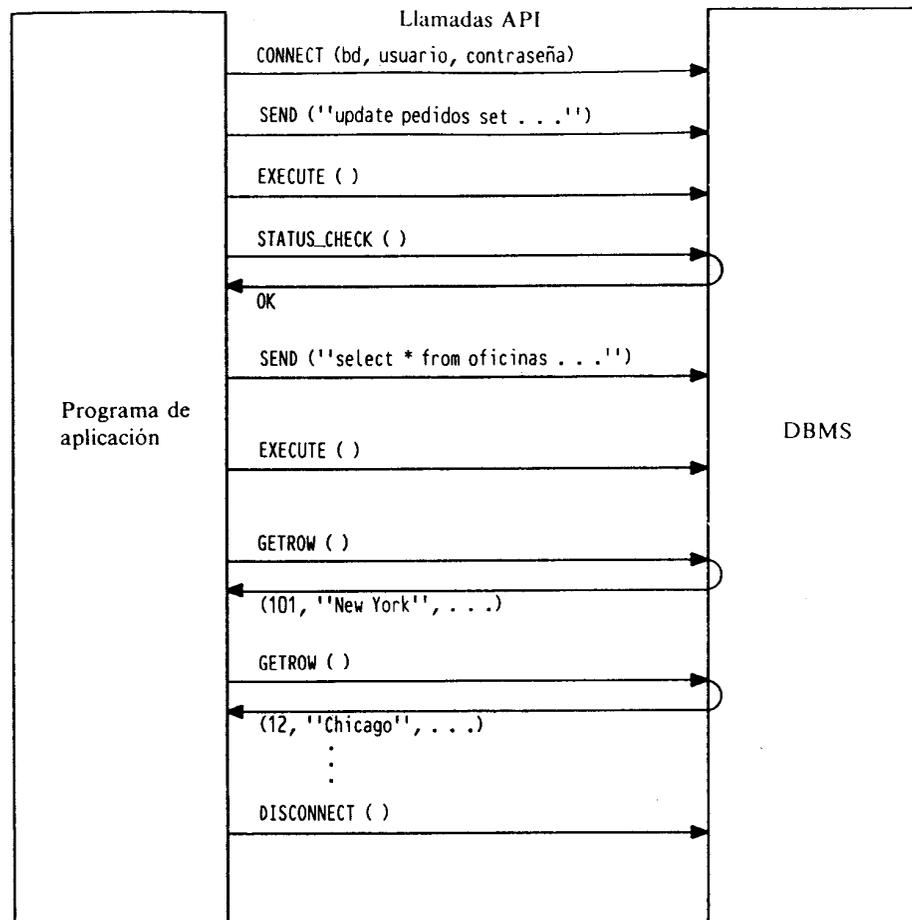
PRINT contador;

6.5 Acceso a la B.D. desde entornos de programación



API ≡ Interfaz de Programas de Aplicación

acceso a la B.D. utilizando una API



Un API SQL en una arquitectura cliente/servidor.

Utilización de un API SQL para acceder al DBMS

operaciones típicas de una API para acceso a la B.D.

| Función | Descripción |
|---|---|
| <i>Conexión/desconexión de la base de datos</i> | |
| <code>dblogin()</code> | Proporciona una estructura de datos para información de presentación. |
| <code>dbopen()</code> | Abre una conexión al SQL Server. |
| <code>dbuse()</code> | Establece la base de datos por omisión. |
| <code>dbexit()</code> | Cierra una conexión al SQL Server. |
| <i>Procesamiento básico de sentencias</i> | |
| <code>dbcmd()</code> | Transfiere el texto de sentencia SQL a <code>dblib</code> . |
| <code>dbsqlexec()</code> | Pide la ejecución de un lote de sentencias. |
| <code>dbresults()</code> | Obtiene resultados de la siguiente sentencia SQL en un lote. |
| <code>dbcancel()</code> | Cancela el resto de un lote de sentencias. |
| <i>Manipulación de errores</i> | |
| <code>dbmshandle()</code> | Establece un procedimiento manipulador de mensajes escritos por el usuario. |
| <code>dberrhandle()</code> | Establece un procedimiento manipulador de errores escritos por el usuario. |
| <i>Procesamiento de resultados de consulta</i> | |
| <code>dbbind()</code> | Liga una columna de resultados de consulta a una variable de programa. |
| <code>dbnextrow()</code> | Accede a la siguiente fila de resultados. |
| <code>dbnumcols()</code> | Obtiene el número de columnas de los resultados de consulta. |
| <code>dbcollname()</code> | Obtiene el nombre de una columna de resultados. |
| <code>dbcolltype()</code> | Obtiene el tipo de datos de una columna de resultados. |
| <code>dbcollen()</code> | Obtiene la longitud máxima de una columna de resultados. |
| <code>dbdata()</code> | Obtiene un puntero a un valor de dato recuperado. |
| <code>dbdatlen()</code> | Obtiene la longitud efectiva de un valor de dato recuperado. |
| <code>dbcancquery()</code> | Cancela una consulta antes de que se haya accedido a todas las filas. |

Funciones API básicas de `dblib`

introducción a SQL embebido

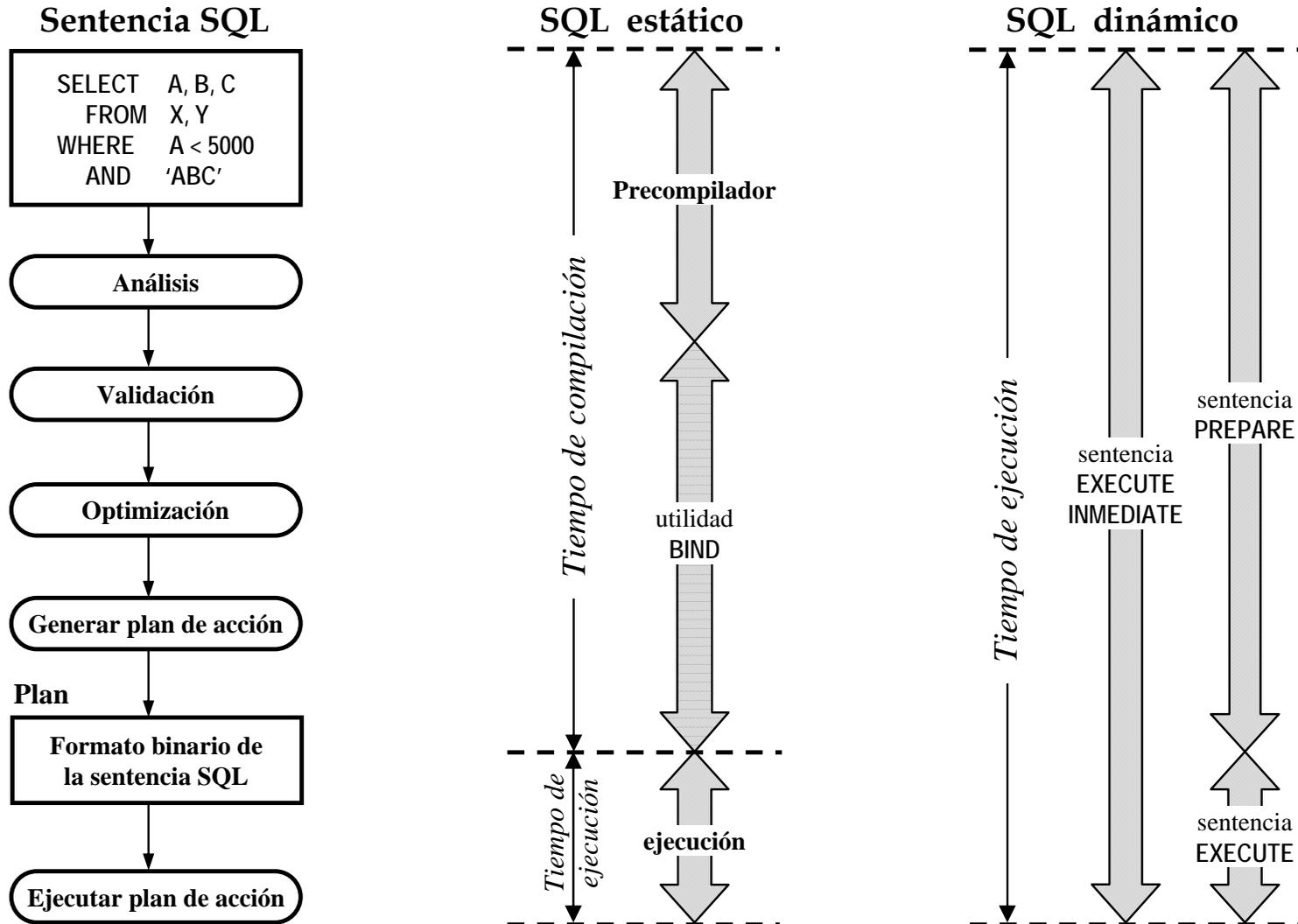
☛ *para utilizar SQL embebido es necesario (para cada lenguaje de programación):
un precompilador + bibliotecas del SGBD*

modo ejecución { **SQL estático** ≡ *sentencias SQL conocidas en tiempo de compilación*
SQL embebido: { **SQL dinámico** ≡ *sentencias SQL conocidas en tiempo de ejecución*

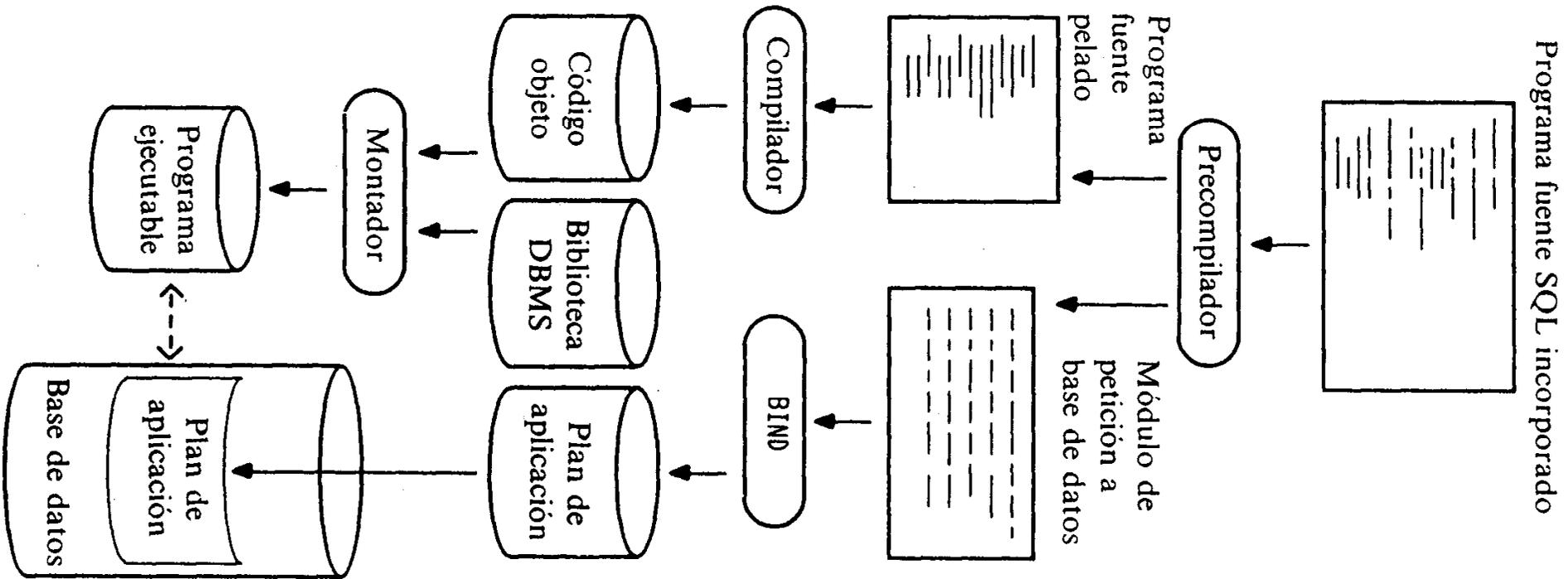
¿ cual es la diferencia entre

SQL interactivo
SQL estático
SQL dinámico ?

ejecución de una sentencia en SQL embebido



pasos en la compilación de SQL embebido



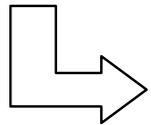
aspectos básicos de SQL embebido (estático) (1)

aspectos básicos:

✓ sentencias SQL “mezcladas” con sentencias del programa \Rightarrow **precompilador**

└─→ marcas sintácticas (**EXEC SQL ...**)

✓ intercambio de información: datos, resultado de operaciones, errores, etc..



Áreas de comunicación compartidas (\approx variables globales):

- **SQLCA** { código error (**SQLCODE**)
indicadores
diagnósticos
nº de filas,
.....
- **ORACA** (ORACLE)

= 0 \rightarrow no error
< 0 \rightarrow error grave
> 0 \rightarrow atención

variables “compartidas”

aspectos básicos de SQL embebido (estático) (2)

✓ variables del lenguaje anfitrión podrán ser referenciadas desde SQL (\equiv compartidas)

└─ como **datos** de operaciones, o **resultado** de consultas

└─ Modificar algunas sentencias SQL (**SELECT ... INTO variable**)

☛ hay que especificar la compatibilidad de tipos

☛ marca sintáctica para distinguirlas (**:variablePrograma**)

= 0 → correcto
< 0 → NULL
> 0 → truncado

└─ Aspectos específicos del M.R.

- **valores nulos** →
- **tablas** →

Variables especiales asociadas a las del programa (**:var:indVar**)

CURSORES (sentencias añadidas a SQL para tratamiento fila a fila)

✓ tratamiento de errores

- consulta de variable de estado **SQLCODE**
- utilizando la sentencia **WHENEVER**

SQLWARNING
SQLERROR
NOT FOUND

CONTINUE
STOP
GOTO ...

aspectos básicos de SQL embebido (estático) (3)

tipos de sentencias

declarativas

EXEC SQL BEGIN DECLARE SECTION
EXEC SQL END DECLARE SECTION → para variables “comunes”

EXEC SQL DECLARE ... → para objetos del SGBD (cursores, etc.)

EXEC SQL INCLUDE ... → para áreas de comunicación

EXEC SQL WHENEVER ... → para tratamiento de errores

ejecutables

a) definición (CREATE, ALTER, DROP, RENAME
CONNECT, GRANT, REVOKE, LOCK TABLE, ..)

b) manipulación (DELETE, INSERT, UPDATE,
OPEN, CLOSE, FETCH, SELECT
COMMIT, ROLLBACK, SAVEPOINT, ..)

→ todas con **EXEC SQL** delante

ejemplo de programación con SQL embebido (estático) (1)

ejemplo_1:

```
{ Este es un sencillo ejemplo de utilización del SGBD de ORACLE desde un programa escrito en THINK }
{ PASCAL. Para ello realiza sencillas operaciones de acceso a la base de datos del ejemplo bancario: }
program EjPregunta_1 (input,output);

uses utilidades; {Se utiliza la biblioteca de utilidades}

label 1234,9999; {etiquetas correspondientes al tratamiento de error, y fin de programa }

const CR = chr(13); {caracter de final de línea}

type
    EXEC SQL INCLUDE ORATYPE; { inclusión de los tipos utilizables del gestor }

var
    EXEC SQL begin DECLARE SECTION;
        username      :      Varchar[10];      {Oracle username}
        password      :      Varchar[10];      {Oracle Password}
    {Declaración de las variables globales usadas en la comunicación con el SGBD}
        totCuentas    :      integer;          {total de cuentas abiertas}
        numCuenta     :      OraLong;          {numero de la cuenta bancaria}
        elSaldo       :      OraLong;          {saldo de la cuenta bancaria}
    EXEC SQL end DECLARE SECTION;

    EXEC SQL INCLUDE SQLCA; { inclusión del área de comunicación de SQL (errores, etc.) }
```

ejemplo de programación con SQL embebido (estático) (2)

```
begin
  username.body := 'PRACTICAS';    username.length := 9;
  password.body := 'PRACTICAS';    password.length := 9;

  EXEC SQL WHENEVER SQLERROR goto 1234;
  EXEC SQL CONNECT :username IDENTIFIED BY :password; {acceso a la Base de Datos}
  writeln(CR, 'Conectado a la Base de Datos como ', username.body, CR);

  EXEC SQL SELECT count(*) INTO :totCuentas FROM cuenta;

  if totCuentas <= 0
  then writeln(CR, 'No hay cuentas abiertas en el banco', CR)
  else begin
    writeln(CR, 'Hay un total de ', totCuentas:1, ' cuentas abiertas');
    write('nº de la cuenta ? '); readln(numCuenta);

    EXEC SQL SELECT saldo INTO :elSaldo FROM cuenta WHERE numero=:numCuenta;

    if SQLCA.SQLCODE>0
    then writeln('Err:', SQLCA.SQLCODE:1, ': No existe la cuenta ', numCuenta)
    else writeln('Saldo de cuenta ', numCuenta:1, ' = ', elSaldo, ' Ptas.');

    EXEC SQL COMMIT RELEASE; {validar las transacciones realizadas y desconectarse}
    writeln (CR, 'Desconexión de la Base de Datos', CR)
  end {else};
  goto 9999;

1234: writeln(SQLCA.SQLERRM);                {escribe mensaje de error}
      EXEC SQL WHENEVER SQLERROR CONTINUE;  {continuar aunque exista error}
      EXEC SQL ROLLBACK RELEASE;           {deshacer las últimas transacciones}

9999: writeln ('Pulsar <CR> para acabar'); readln
end.
```

recorrido de las tablas resultado de consultas: cursores

manejo de CURSORES (≈ fichero secuencial)

☛ *cada cursor representa una tupla de la tabla que se “recorre” ⇒ está asociado a una consulta*

DECLARE CURSOR nombreCursor FOR consulta;

OPEN nombreCursor;

FETCH nombreCursor INTO listaVariables;

CLOSE nombreCursor;

genera condición de error al intentar posicionarse sobre tupla no existente

→ *para operar con la tupla seleccionada:*

DELETE FROM nombreTabla WHERE CURRENT OF nombreCursor;
UPDATE

ejemplo de manejo de cursores en SQL embebido (1)

ejemplo_2:

```
{ Este es un sencillo ejemplo de utilización del SGBD de ORACLE desde un programa escrito en THINK PASCAL.
  Para ello realiza sencillas operaciones de acceso a la base de datos del ejemplo bancario, usando CURSORES
}
program EjPregunta_2 (input,output);
label 1234,9999;      { etiquetas correspondientes al tratamiento de error, y fin de programa }
const CR = chr(13); { caracter de final de línea}
type
  EXEC SQL INCLUDE ORATYPE; { inclusión de los tipos utilizables del gestor }
var
  EXEC SQL begin DECLARE SECTION;
  username      :      Varchar[10];      {Oracle username}
  password      :      Varchar[10];      {Oracle Password}
  {Declaración de las variables globales usadas en la comunicación con el SGBD}
  elDNI         :      Oralong;           {DNI de un cliente}
  nam           :      Varchar[25];       {nombre de un cliente}
  numAg         :      integer;           {número de agencia}
  numCli        :      integer;           {contador de clientes}
  EXEC SQL end DECLARE SECTION;
  EXEC SQL INCLUDE SQLCA; { inclusión del área de comunicación de SQL (errores, etc.) }
  numClientes  : integer;                  { variable local para contar clientes }
```

ejemplo de manejo de cursores en SQL embebido (2)

```
begin
  Pro_Pascal_Init;      { iniciar interface con SGBD (variables compartidas, etc.) }
  username.body := 'PRUEBAS';      username.length := 8;
  password.body := 'PRUEBAS';      password.length := 9;

  EXEC SQL WHENEVER SQLERROR goto 1234;

  EXEC SQL CONNECT :username IDENTIFIED BY :password; {acceso a la Base de Datos}

  writeln(CR, 'Conectado a la Base de Datos como :', username.body, CR);
  numClientes := totalDeClientes;

  if numClientes <= 0
  then writeln(CR, 'No hay clientes en el banco', CR)
  else begin
    writeln(CR, 'Hay un total de ', numClientes:1, ' clientes del banco', CR);
    ClientesMasAgenciasDe (1, numCli);

    EXEC SQL COMMIT RELEASE; {validar las transacciones realizadas y desconectarse}

    writeln (CR, 'Desconexión de la Base de Datos', CR)
  end {else};
  goto 9999;

1234:writeln(SQLCA.SQLERRM);      {escribe mensaje de error}
  EXEC SQL WHENEVER SQLERROR CONTINUE; {continuar aunque exista error}
  EXEC SQL ROLLBACK RELEASE;      {deshacer las últimas transacciones}

9999:writeln ('Pulsar <CR> para acabar'); readln
end.
```

ejemplo de manejo de cursores en SQL embebido (3)

```
{Este procedimiento devuelve a través del parámetro 'numClientes' el número de clientes}
{que tienen cuenta en un nº de sucursales mayor que 'minAg'. Visualiza el DNI de éstos.}
procedure ClientesMasAgenciasDe (minAg: integer; var numClientes: integer);
label 77,99;
begin
    numClientes:=0;
    writeln('los que tienen cuentas en más de ', minAg:1, ' agencias son');
    writeln('      DNI', CR, '-----');

{Definición de un cursor para obtener información de los clientes}
    EXEC SQL DECLARE unCliente CURSOR FOR SELECT DISTINCT dni FROM apercuenta;

    EXEC SQL OPEN unCliente;                {iniciar el cursor para interrogar a la B.D.}

    EXEC SQL WHENEVER not FOUND goto 77; {definir una excepción para fin de tratamiento de clientes}
    while true do begin {obtener el DNI de un nuevo cliente}

        EXEC SQL FETCH unCliente INTO :elDNI;

        {obtención del nº de agencias en que tiene cuenta este cliente}
        EXEC SQL SELECT count(DISTINCT agencia) INTO :numAg FROM apercuenta WHERE dni=:elDNI;

        if numAg > minAg then begin
            writeln(elDNI:10, ' ':3, ' tiene cuentas en ', numAg:1, ' sucursales');
            numClientes:=numClientes+1
        end
    end; {while}

77: EXEC SQL CLOSE unCliente; {final de la utilización del cursor}
    writeln(CR, 'hay ', numClientes:1, ' clientes con cuentas en más de ', minAg:1, ' agencias');
99:
end;
```

ejemplo de manejo de cursores en SQL embebido (4)

```
{Esta función devuelve el número total de clientes del banco}
function totalDeClientes : integer;
label 99;
begin
    totalDeClientes := 0;
    EXEC SQL WHENEVER not FOUND goto 99; {definir la excepción correspondiente}
    EXEC SQL SELECT count(*) INTO :numCli FROM cliente;
    totalDeClientes:=numCli
99:
end;
```

introducción a SQL embebido dinámico (1) (en ORACLE)

☛ *SQL dinámico* ⇒ la sentencia *SQL* es conocida sólo en tiempo de ejecución

→ es algo más compleja su utilización

Modo 1 ⇒ para sentencias que no sean *SELECT* y que no utilicen variables del programa

```
EXEC SQL EXECUTE IMMEDIATE :string
```

string

```
orden ← UPDATE Pieza SET color = 'ROJO' WHERE nombPieza = 'TUERCA';
```

```
EXEC SQL EXECUTE IMMEDIATE :orden;
```

introducción a SQL embebido dinámico (2) (en ORACLE)

Modo 2 ⇒ para sentencias que no sean *SELECT* y con *nº* de variables del programa conocido

```
EXEC SQL PREPARE sentencia FROM :string
```

+

```
EXEC SQL EXECUTE sentencia USING variables
```

string

```
orden ← UPDATE Pieza SET color = :c WHERE nombPieza = 'TUERCA';
```

```
EXEC SQL PREPARE actualizar FROM :orden;
```

*string con
valor a signar*

```
EXEC SQL EXECUTE actualizar USING :miColor;
```

introducción a SQL embebido dinámico (3) (en ORACLE)

Modo 3 ⇒ para sentencias *SELECT* con lista de item y n° de variables del programa conocidos

```
EXEC SQL PREPARE sentencia FROM :string
```

+ sentencias de
manejo de cursor

```
PREPARE sentencia FROM :string;  
DECLARE CURSOR nombreCursor FOR sentencia;  
OPEN nombreCursor [USING variables];  
FETCH nombreCursor INTO variables;  
CLOSE nombreCursor;
```

```
orden ← SELECT clvPieza, color WHERE nombPieza = :nombPieza;
```

string

```
EXEC SQL PREPARE consulta FROM :orden;
```

```
EXEC SQL DECLARE fila CURSOR FOR consulta ;
```

```
EXEC SQL OPEN fila USING :nombP;
```

```
EXEC SQL FETCH fila INTO :laClave, :elColor;
```

```
EXEC SQL CLOSE fila;
```

variables
de programa

introducción a SQL embebido dinámico (4) (en ORACLE)

Modo 4 ⇒ *para cualquier sentencia SQL*

EXEC SQL PREPARE *sentencia* FROM *:string*

+ *sentencias de manejo de cursor*

☛ *un poco más complejo que el modo 3*

☛ *se necesitan áreas de comunicación específicas (SQLDA, ..)*

Area de Descriptores de datos SQL

ejemplos: véase la documentación de ORACLE y servidor de prácticas