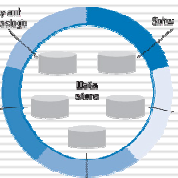
Base de datos II

Preparado por: Eduardo Godoy

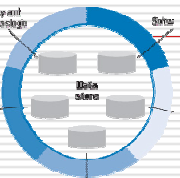
V 1.2



Historia...

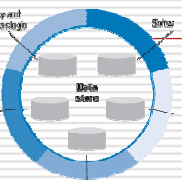
- Egipto y Mesopotamia
  - La biblioteca más antigua data del tercer milenio antes de Cristo, en la ciudad de Nippur en la Antigua Babilonia.
  - La más famosa es la de Alejandría (290 a.c.) en Egipto. Contenía 750.000 volumenes (en papiro escrito a mano!)

  (haciendo un cálculo muy ligero, podemos decir que eran más de 1.440.000.000.000 palabras a indexar)



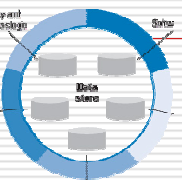
DBMS en $$

- El negocio de los DBMS en el 2001 fue (en Millones de US$) :
  - IBM $ 3.064,2 (34,6%)
  - Oracle $ 2.829,8 (32,0%)
  - Microsoft $ 1.442,7 (16,3%)
  - Sybase $ 234,2 (2,6%)
  - Hitachi $ 127,8 (1,4%)
  - Otros $ 1.144,8 (12,9%)
  - TOTAL: $ 8.843,5



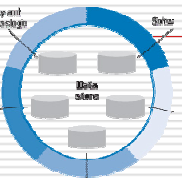
RDBMS

- Los 5 principales vendedores de RDBMS son (en Millones de US$) :
  - Oracle $ 2.829,4 (39,8%)
  - IBM $ 2.421,0 (34,1%)
  - Microsoft $ 1.020,0 (14,4%)
  - Sybase $ 234,2 (3,3%)
  - NCR $ 124,2 (1,7%)
  - Otros $ 478,8 (6,7%)
  - TOTAL $ 7.107,6



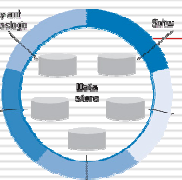
Status actual

- Oracle 10g
- PostgreSQL 8.0
- MySQL 4.1
- Java en las Bases de Dato_



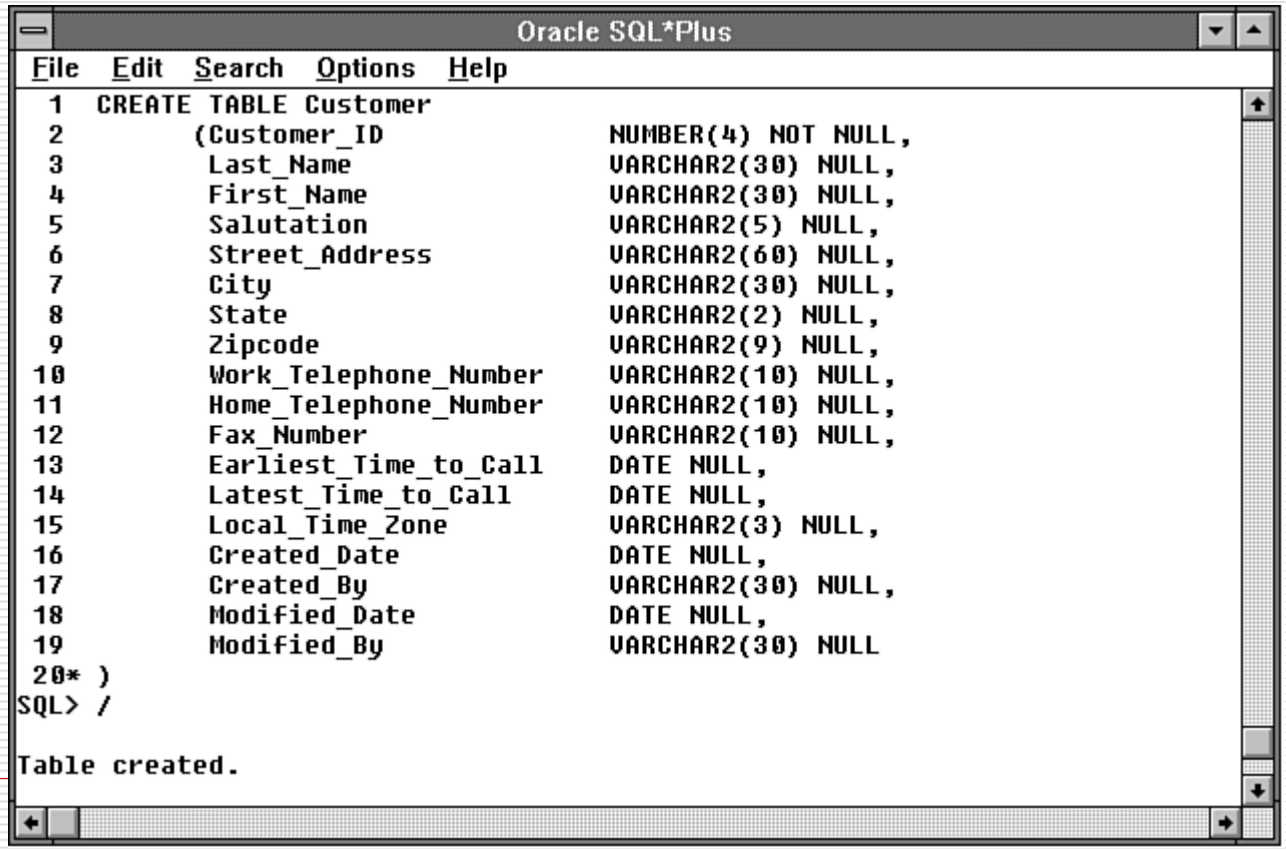
Antes de seguir...

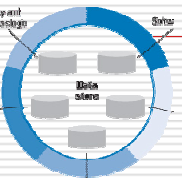
- Qué es una base de datos?
- Qué diferencia “datos” de “información” ?
- Qué le debo exigir a una base de datos?
- Qué es un motor de base de datos?



SQL Interactivo

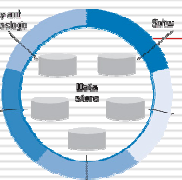
- Hasta el momento, el SQL interactivo es el más conocido:





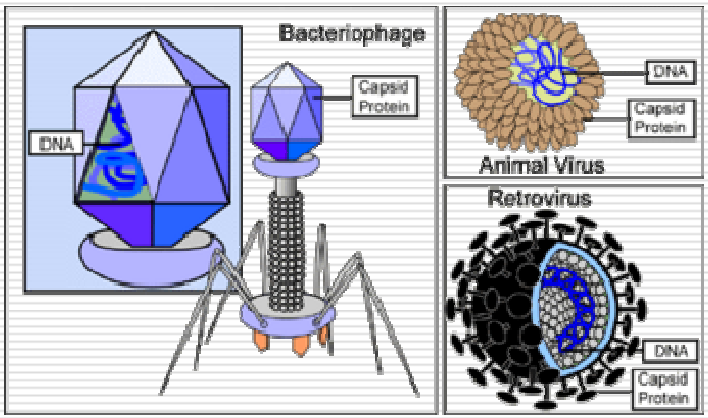
SQL Interactivo

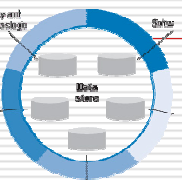
- No es procedural.
- Especifica QUE se requiere, no COMO se requiere.
- Es bueno para:
  - Definir estructuras de bases de datos.
  - Pequeñas consultas.
  - Prototipos.
- No es bueno para consultas más complejas.



SQL Embedded

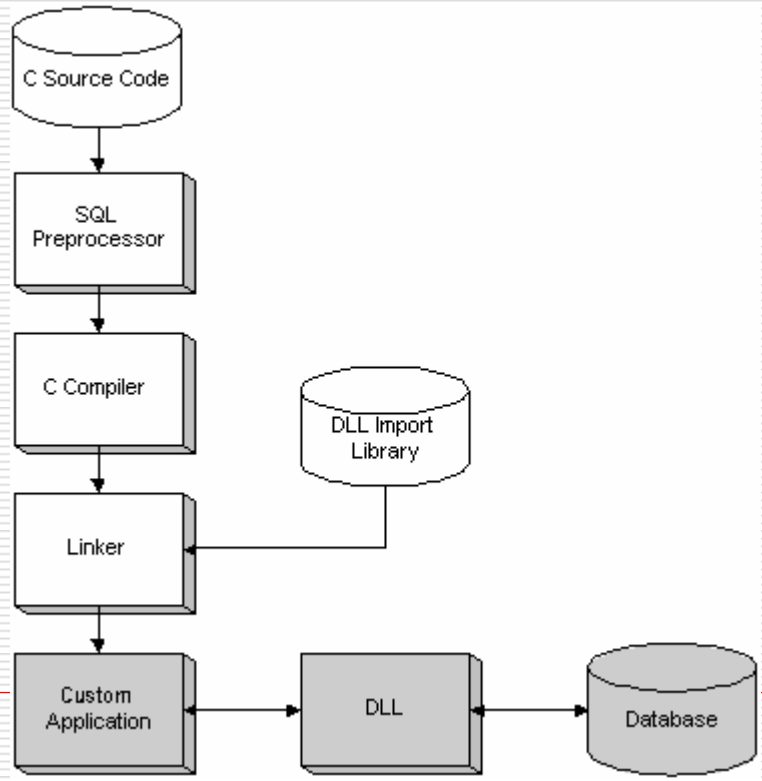
- Es colocar código SQL dentro de un lenguaje de programación tradicional.

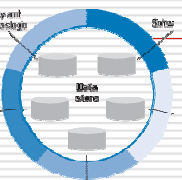




SQL Embedded

- Al vivir dentro de un lenguaje 3GL, permite aprovechar todas sus ventajas:
  - Mayor nivel de personalización.
  - Consultas más complejas y con mayor nivel de cálculo.

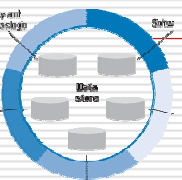




SQL Embedded

- Se comparten variables entre el ambiente HOST y el SQL

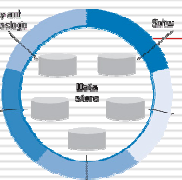
```
EXEC SQL begin declare section;
    varchar userid[10],password[10],cname[15];
    int cno;
EXEC SQL end declare section;
```



Conectandose

```
EXEC SQL connect :userid identified by :password;
```

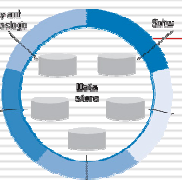
- Antes de hacer algo, debemos conectarnos.
- El prefijo : define variables compartidas entre ambos ambientes.



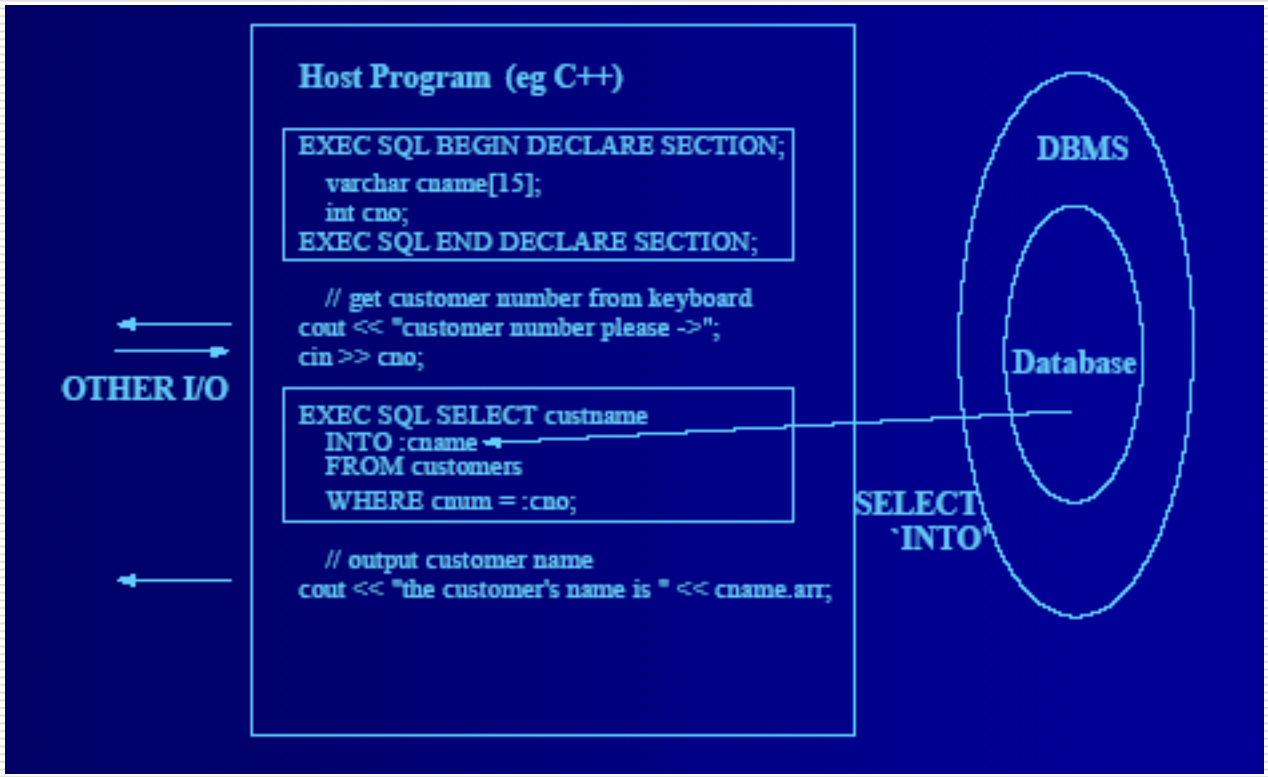
Consultando

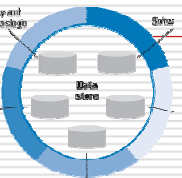
```
EXEC SQL    SELECT    custname
            INTO      :cname
            FROM      customers
            WHERE     cno = :cno;
```

- La columna custname es almacenada en la variable :cname
- La condición de selección es filtrada por el valor de la variable compartida :cno



Ejemplo


Host Program (eg C++)
EXEC SQL BEGIN DECLARE SECTION;
varchar cname[15];
int cno;
EXEC SQL END DECLARE SECTION;
// get customer number from keyboard
cout << "customer number please ->";
cin >> cno;
EXEC SQL SELECT custname
INTO :cname
FROM customers
WHERE cnum = :cno;
// output customer name
cout << "the customer's name is " << cname.arr;
OTHER I/O
DBMS
Database
SELECT
`INTO'

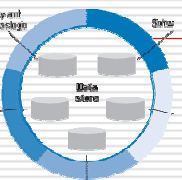


SQL Embedded

- Pero qué ocurre si retorna más de una fila?
  - Solución: Cursores.
- Declaración:

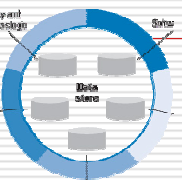
```
EXEC SQL DECLARE <cursor> CURSOR FOR
<query>
```

- Luego se usa EXEC SQL OPEN <cursor>, para abrir el cursor y dejarlo posicionado en el primer registro.



Cursores

- Con la sentencia EXEC SQL FETCH <cursor> INTO <lista de variables> se recuperan los valores de la posición actual del cursor y se avanza al siguiente registro.



SQL Embedded

- Existe una forma de referenciar a la fila actual de un cursor y se usada para modificar los datos.
- EXEC SQL DELETE FROM <table> WHERE CURRENT OF <cursor>
- La condicion que se debe cumplir para hacer uso de esta funcionalidad es que El cursor hace referencia sólo una tabla (en la declaración en vez de ir una query va el nombre de una tabla).

SQL Embedded.

- Problema de concurrencia.
  - Qué pasa si dos programas tratan de modificar la misma información?
- Solución: Declarar el cursor como INSENSITIVE
- EXEC SQL DECLARE <cursor> INSENSITIVE CURSOR FOR <query>

- Algunas keywords cambian dependiendo del DMBS.

SQL Embedded.

- Navegando en el cursor.
  - Por default un cursor solo permite navegación secuencial y en un solo sentido.
  - Agregando la keyword SCROLL antes de la keyword CURSOR, agregamos algunas funcionalidades:
    - NEXT / PRIOR (default)
    - FIRST / LAST
    - RELATIVE
    - ABSOLUTE
- EXEC SQL FETCH LAST FROM <cursor> INTO <lista de variables>
- EXEC SQL FETCH PRIOR FROM <cursor> INTO <lista de variables>

Código Ejemplo-PostgreSQL

```
exec sql include sqlca;
#include <stdio.h>
int main(int argc, char **argv)
{  exec sql begin declare section;
   int index;
   exec sql end declare section;
   exec sql connect to mm;
   if (sqlca.sqlcode) printf("%ld:%s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
   exec sql create table test (       "index" numeric(3) primary key,       "payload" int4 NOT NULL);
   if (sqlca.sqlcode) printf("%ld:%s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
   exec sql commit work;
   if (sqlca.sqlcode) printf("%ld:%s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
  for (index=0;index<10;++index)
   {  exec sql insert into test (payload, index)  values (0, :index);
      if (sqlca.sqlcode) printf("%ld:%s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
   }
```

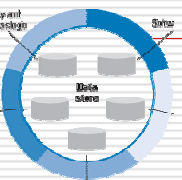
Código Ejemplo -PostgreSQL

```
exec sql commit work;
  if (sqlca.sqlcode) printf("%ld: %s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
    exec sql update test set payload=payload+1 where index=-1;
  if (sqlca.sqlcode!=100) printf("%ld: %s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
    exec sql delete from test where index=-1;
  if (sqlca.sqlcode!=100) printf("%ld: %s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
  exec sql insert into test (select * from test where index=-1);
  if (sqlca.sqlcode!=100) printf("%ld: %s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
  exec sql drop table test;
  if (sqlca.sqlcode) printf("%ld: %s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
  exec sql commit work;
  if (sqlca.sqlcode) printf("%ld: %s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
  exec sql disconnect;
  if (sqlca.sqlcode) printf("%ld: %s\n",sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
  return 0;
}
```

SQL Dinámico

- Consiste en crear sentencias SQL en forma dinámica y luego ejecutarlas:

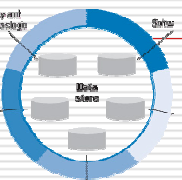
```
EXEC SQL BEGIN DECLARE SECTION;
    const char *stmt = "CREATE TABLE test1 (...);";
EXEC SQL END DECLARE SECTION;
EXEC SQL EXECUTE IMMEDIATE :stmt;
```



SQL Dinámico

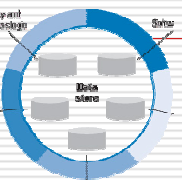
```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "INSERT INTO test1 VALUES(?, ?);";
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE mystmt FROM :stmt;
 ...
EXEC SQL EXECUTE mystmt USING 42, 'foobar';
```



SQL Dinámico

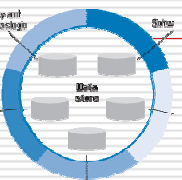
```
EXEC SQL BEGIN DECLARE SECTION;
 const char *stmt = "SELECT a, b, c FROM test1 WHERE a > ?";
int v1, v2,v4,v5;
VARCHAR v3,v6;
EXEC SQL END DECLARE SECTION;
EXEC SQL PREPARE mystmt FROM :stmt;
 ...
EXEC SQL EXECUTE mystmt INTO :v1, :v2, :v3 USING 37;
EXEC SQL EXECUTE mystmt INTO :v4, :v5, :v6 USING 43;
```



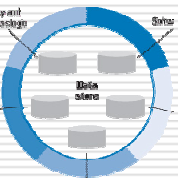
SQL Dinámico

EXEC SQL DEALLOCATE PREPARE *name*;

- Libera los recursos utilizados.

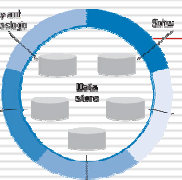


Control de Errores



Control de Errores

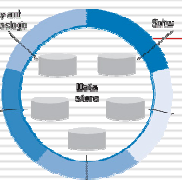
- Existen dos formas de programar el control de errores:
  - SQLCA
  - CALLBACKS



SQLCA

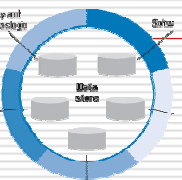
- SQLCA: SQL Communication Area.
- Define un tipo de datos que es usado por el motor para comunicar resultados de ejecuciones y estados.

```
struct {
        char sqlcaid[8];
        long sqlabc;
        long sqlcode;
        struct {
                        int sqlerrml;
                        char sqlerrmc[70]; } sqlerrm;
        char sqlerrp[8];
        long sqlerrd[6];
        char sqlwarn[8];
        char sqlstate[5];
        } sqlca;
```



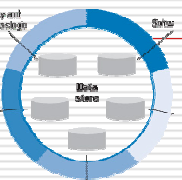
SQLCA

- sqlcaid: This string component is initialized to "SQLCA" to identify the SQL Communications Area.
- sqlcabc This integer component holds the length, in bytes, of the SQLCA structure.
- sqlcode This integer component holds the status code of the most recently executed SQL statement:
  - 0 No error.
  - >0 Statement executed but exception detected. This occurs when Oracle cannot find a row that meets your WHERE condition or when a SELECT INTO or FETCH returns no rows.
  - <0 Oracle did not execute the statement because of an error. When such errors occur, the current transaction should, in most cases, be rolled back.



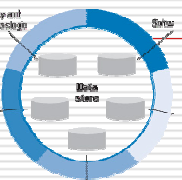
SQLCA

- sqlerrm: This embedded structure contains the following two components:
  - sqlerrml - Length of the message text stored in sqlerrmc.
  - sqlerrmc - Up to 70 characters of the message text corresponding to the error code stored in sqlcode.
- sqlerrp Reserved for future use.
- sqlerrd This array of binary integers has six elements:
  - sqlerrd[0] - Future use.
  - sqlerrd[1] - Future use.
  - sqlerrd[2] - Numbers of rows processed by the most recent SQL statement.
  - sqlerrd[3] - Future use.
  - sqlerrd[4] - Offset that specifies the character position at which a parse error begins in the most recent SQL statement.
  - sqlerrd[5] - Future use.



SQLCA

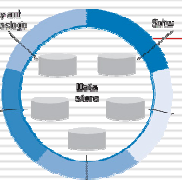
- sqlwarn This array of single characters has eight elements used as warning flags. Oracle sets a flag by assigning to it the character 'W'.
  - sqlwarn[0] Set if any other flag is set.
  - sqlwarn[1] Set if a truncated column value was assigned to an output host variable.
  - sqlwarn[2] Set if a NULL column value is not used in computing a SQL aggregate such as AVG or SUM.
  - sqlwarn[3] Set if the number of columns in SELECT does not equal the number of host variables specified in INTO.
  - sqlwarn[4] Set if every row in a table was processed by an UPDATE or DELETE statement without a WHERE clause.



SQLCA

- sqlwarn[5] Set if a procedure/function/package/package body creation command fails because of a PL/SQL compilation error.
- sqlwarn[6] No longer in use.
- sqlwarn[7] No longer in use.

- sqlext Reserved for future use.

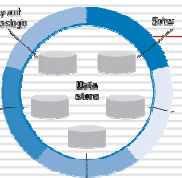


CALLBACKS

- Usando CALLBACKS

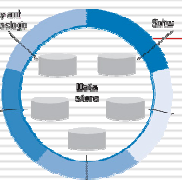
EXEC SQL WHENEVER *condition action*;

- Las condiciones pueden ser:
  - SQLERROR
    - La acción especificada será llamada cuando ocurra un error en la ejecución de una sentencia SQL.
  - SQLWARNING
    - La acción especificada será llamada cuando ocurra una advertencia en la ejecución de una sentencia SQL.
  - NOT FOUND
    - La acción especificada será llamada cuando la sentencia SQL afecte o recupere 0 filas.



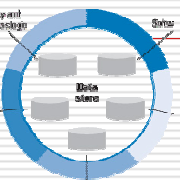
Control de Errores

- Las acciones posibles son:
  - CONTINUE
    - Significa que la condición es ignorada (default).
  - GOTO label
    GO TO label
    - Salta al label indicado (usando el Goto de C)
  - SQLPRINT
    - Imprime el mensaje en la salida estándar. Es útil en pequeños programas o prototipos.
  - STOP
    - Invoca exit(1), terminando el programa.
  - BREAK
    - Ejecuta un break al estilo “C”. Esto sólo puede usarse en ciclos o sentencias switch.
  - CALL *name* (*args*)
    DO *name* (*args*)
    - Llama a una sentencia “C” con los parámetros definidos.



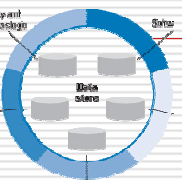
Ejemplo

```
/* code to find student name given id */
/* ... */
for (;;) {
   printf("Give student id number : ");
      scanf("%d", &id);
      EXEC SQL WHENEVER NOT FOUND GOTO notfound;
      EXEC SQL SELECT studentname INTO :st_name
      FROM   student WHERE  studentid = :id;
      printf("Name of student is %s.\n", st_name);
      continue;
      notfound:
            printf("No record exists for id %d!\n", id);
      }
      /* ... */
```



Store Procedures (SP)

- Es una forma de agregar lógica a una báse de datos.
- Cómo su nombre lo indica, son procedimientos o funciones que se almacenan en la base de datos.
- Cada proveedor entrega un lenguaje para escribir SP.
- El lenguaje debe permitir, entre otras cosas:
  - Crear variables
  - Programar ciclos
  - Invocación a otros SP

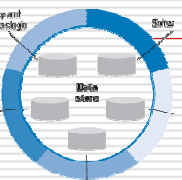


Sintaxis:

CREATE FUNCTION *name* ([[IN | OUT | INOUT] *type* [, ...]]) RETURNS *rtype* LANGUAGE '*langname*' ESPECIFIC *routine SQL-statement*

- `Ejemplo:`

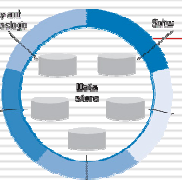
```
CREATE FUNCTION sales_tax(subtotal real)
    RETURNS real AS $$
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```



Bibliografía

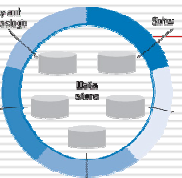
- **Chapter 35. PL/pgSQL - *SQL* Procedural Language**

  http://www.postgresql.org/docs/current/static/plpgsql.html



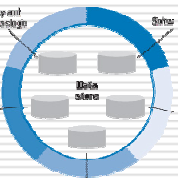
Usando Store Procedures

- Desde SQL Embedded:



Cliente Servidor

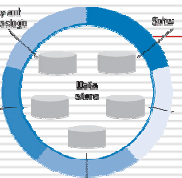
SQL en un ambiente real



1-Tier: Personal Computer 80s

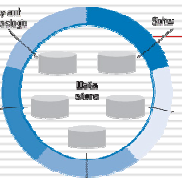
The Mac Plus.

The tiny Timex Sinclair 1000.

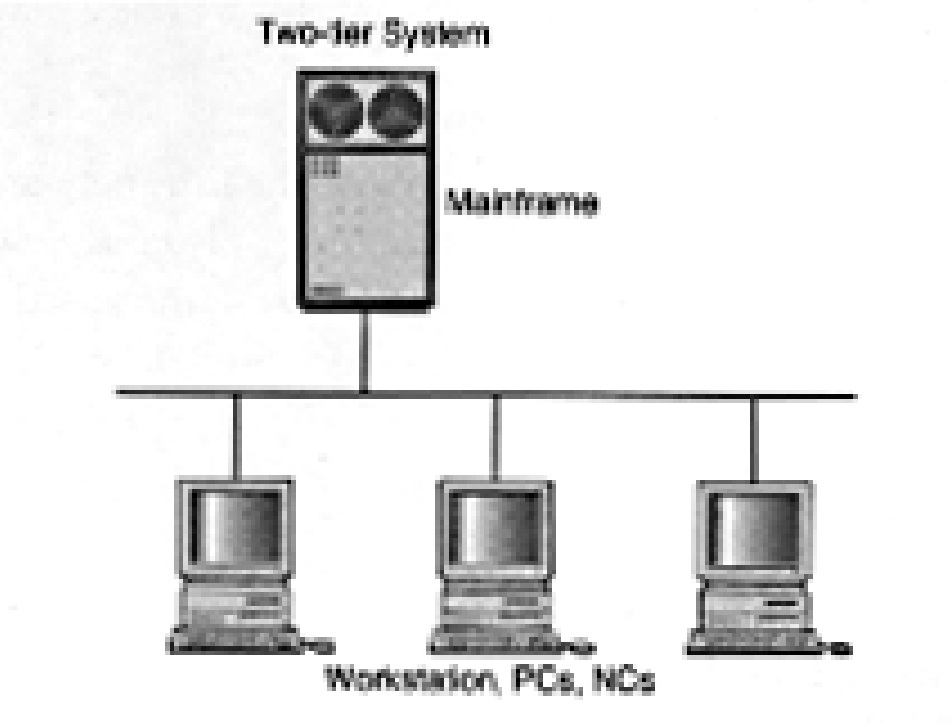


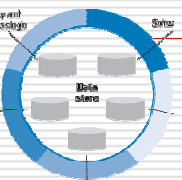
1-Tier: Mainframe

A modern mainframe computer center 1980s

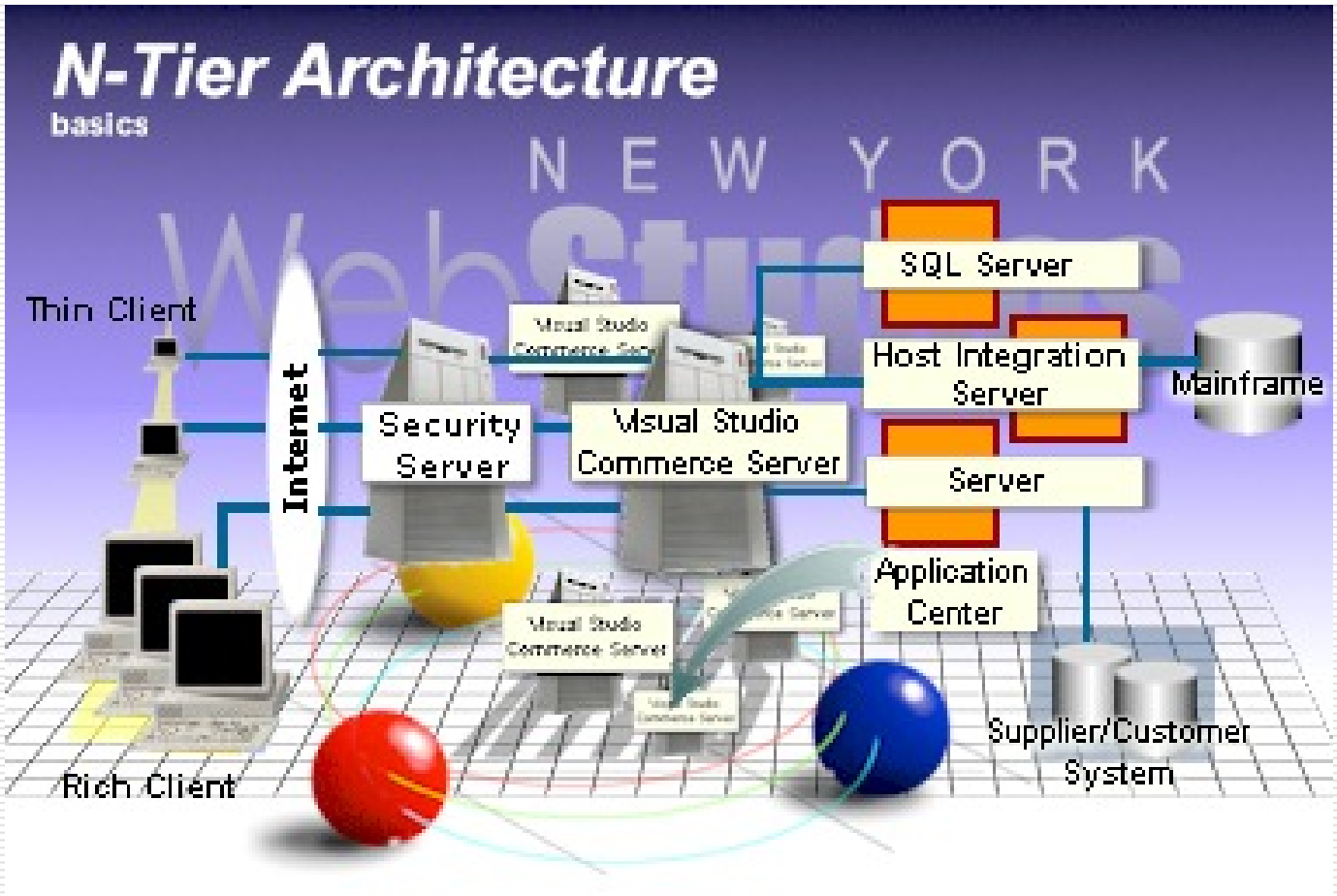


2 Tier



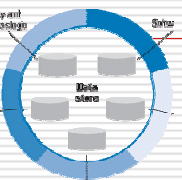


N-Tier


N-Tier Architecture
basics
NEW YORK
Thin Client
Internet
Security Server
Visual Studio Commerce Server
SQL Server
Host Integration Server
Mainframe
Server
Application Center
Supplier/Customer System
Rich Client

Clientes y Servidores en un Ambiente SQL.

- En los sistemas coexisten dos actores: servidor SQL y cliente SQL.
- Servidor SQL: soporta operaciones sobre los objetos de la base de datos.
- Cliente SQL: Permite a los usuarios conectarse a un servidor SQL y ejecutar sentencias sobre él.
- Generalmente el Servidor SQL y el Cliente SQL son dos máquinas diferentes, pudiendo ser la misma.



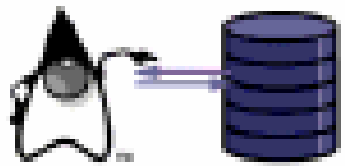
Clientes y Servidores SQL

- Debe existir un mecanismo que permita comunicar a un cliente con un servidor.

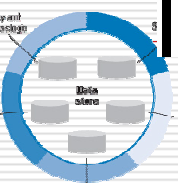
CONNECT TO <server> AS <nombre coneccion>
AUTHORIZATION <usuario y password>

- Existe el concepto de sesión, que coexiste con la conección; cada sesión tiene un conjunto de privilegios y de características. (sobre este punto volveremos más adelante).

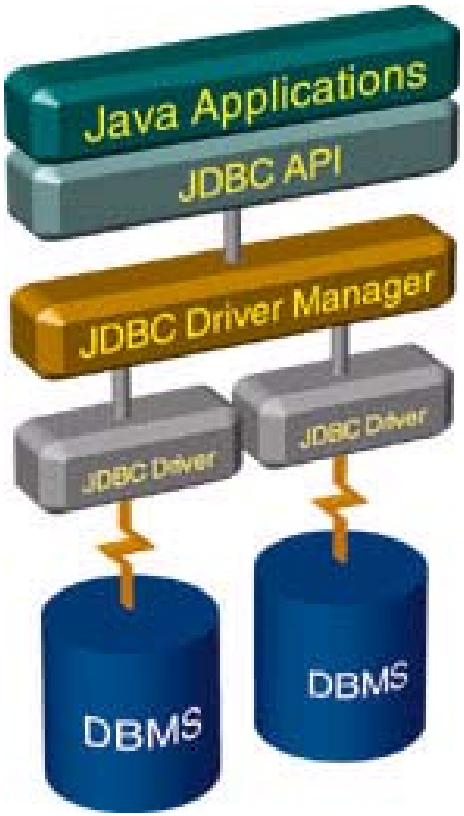
JDBC



- "Java DataBase Connectivity"
- Pasos para usar JDBC:
  - Cargar un "driver" para la base de datos que se usará, esto dependiente de la instalación y de la implementación.
    - Un objeto DriverManager es creado.
    - En PostgreSQL se usa Class.forName("org.postgresql.Driver")
  - Establecer una conección con la base de datos.
    - Connection myCon = DriverManager.getConnection(< URL>, < nombre>, < password>);



Overall Architecture

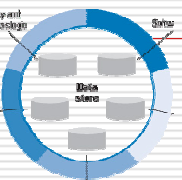


Laboratoire de Bases de Données

EPFL

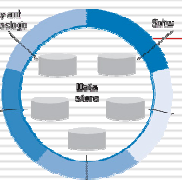
JDBC: Creando sentencias

- Existen 2 métodos que se aplican a la conección para crear sentencias:
  - createStatement();
  - createStatement(Query);
- Para ejecutar la sentencia existen 4 diferentes métodos:
  - executeQuery(Query);
  - executeQuery();
  - executeUpdate(Update);
  - executeUpdate();
- Los métodos executeQuery(Query) y executeQuery retornan un objeto del tipo ResultSet.



JDBC: Cursores

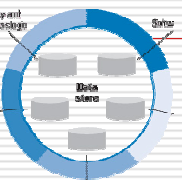
- Al ejecutar una query se obtiene un objeto del tipo ResultSet, y sobre él podemos correr un cursor; ResultSet provee los siguientes métodos:
  - next();
  - getString(i);
  - getInt(i);
  - getFloat(i);
  - Etc (para los otros tipos de datos).



JDBC: Sentencias con parámetros.

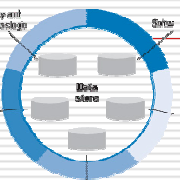
- Se utiliza el caracter "?" en la sentencia para indicar que será reemplazado por el valor de una variable.
- Se debe crear un objeto del tipo PreparedStatement y luego, con métodos setString, setInt, etc. Asignar los valores.
- Ejemplo:

```
PreparedStatement myInsert = myCon.createStatement("Insert
    into Studio(name, address) values (?,?)");
/* obtener los valores que se desean insertar*/
myInsert.setString(1,studioName);
myInsert.setString(2,studioAddr);
myInsert.executeUpdate();
```



JDBC Bibliografía

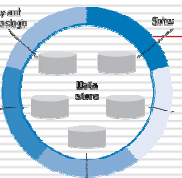
- Leer capítulo 8 del manual del programador PostgreSQL 7.2
- Especificación 3.0 de JDBC de SUN.



Triggers

- Del inglés Trigger, que significa “disparador” (porque gatillo suena feo ☺)
- Son códigos que se adhieren a un objeto, en particular, a una tabla.
- Los triggers no se ejecutan directamente, sino que responden a eventos.
  - BEFORE
  - AFTER

  Sobre una tabla.

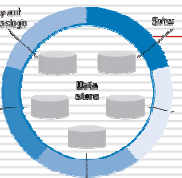


Triggers

- Sintaxis

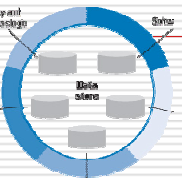
  CREATE TRIGGER *nombretrigger*
  [BEFORE|AFTER][INSERT|DELETE|UPDATE [OR ...]
  on *nombretabla* for each [row|statement]
  EXECUTE PROCEDURE *procedure*
- Borrado: DROP TRIGGER *nombretrigger*
- Ejemplo

  Create trigger audit_empleados
  After insert on empleados for each row
  Referencing new row as NewRow
  Insert into audit_emp(rut, fecha_creacion) values
  (NewRow.rut, now());



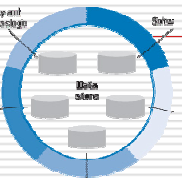
Trigger

- BEFORE | AFTER
  - Indican el “cuando” se debe ejecutar el trigger, si antes de ejecutar la sentencia o despues.
- [INSERT|DELETE|UPDATE [OR ..]
  - Indica qué sentencia es la que gatilla la ejecución del trigger, multiples opciones se pueden especificar separadas por el keywork OR
- FOR EACH [row|statement]
  - Indica si el trigger se ejecuta por cada fila que es afectada o al final (o antes) de la ejecución de toda la sentencia SQL.



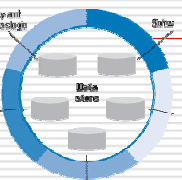
Triggers

- Usos más frecuente de los triggers.
  - Auditar tablas.
  - Control de reglas de negocio.
  - Replicación de datos.
  - Estafas ☹



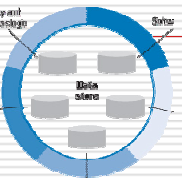
Constraint

- Traducción de RESTRICCIONES.
- Se usan para agregar-definir restricciones sobre los datos.
- Los más conocidos son NULL o NOT NULL.
- Se pueden definir para un campo o un conjunto de tuplas.
- Existen restricciones que aplican a la tabla completa o a un subconjunto de tuplas.



Constraint

- Llaves
  - Primarias
  - Foraneas
- Unicidad
- Check
- Integridad.

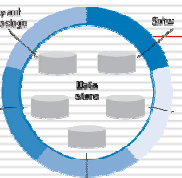


Constraint

- Llaves

```
Create table foo (
fid decimal(3) primary key,
Name varchar(40));

Create table foo2 (
fid decimal(3) primary key,
Name varchar(40),
fid1 decimal(3) references foo(fid));
```



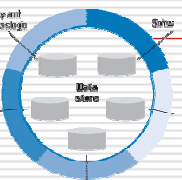
Constraint

- A nivel de columna.

```
create table foo (
  fid     decimal(3) check (fid > 10),
  name  varchar(40)
);
```

- A nivel de tabla

```
create table foo (
  fid     decimal(3),
  name  varchar(40)
  CONSTRAINT con1 CHECK (fid > 10 AND name <> "")
);
```



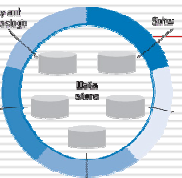
Constraint

- Uso en llaves

```
create table foo (
  fid      decimal(3),
  name  varchar(40)
  CONSTRAINT foo_key  PRIMARY KEY (fid)
);
```

- Nulo o no nulo

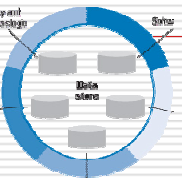
```
create table foo (
  fid      decimal(3) CONSTRAINT no_nulo NOT NULL,
  name  varchar(40)
);
```



Constraint-Assertion

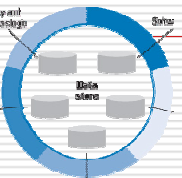
- Para condiciones más complejas, que puede incluir a más de una tabla, se usan los "Assertions".

```
CREATE ASSERTION restrictrabajo CHECK (
NOT EXIST (
    select rut from empleado_proyecto
    group by rut
    having count(*) > 5
    )
);
```



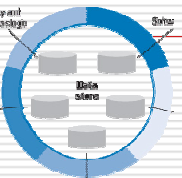
Constraint - Assertions

```
Create assertion restricc_salario
Check (not exist
          (select *
           from empleado E,
                empleado M,
                departamento D
           where E.salario > M.salario
           and E.NDEPTP = D.numero
           and D.rut_JEFE = M.rut)
         );
```



Constraint - Integridad

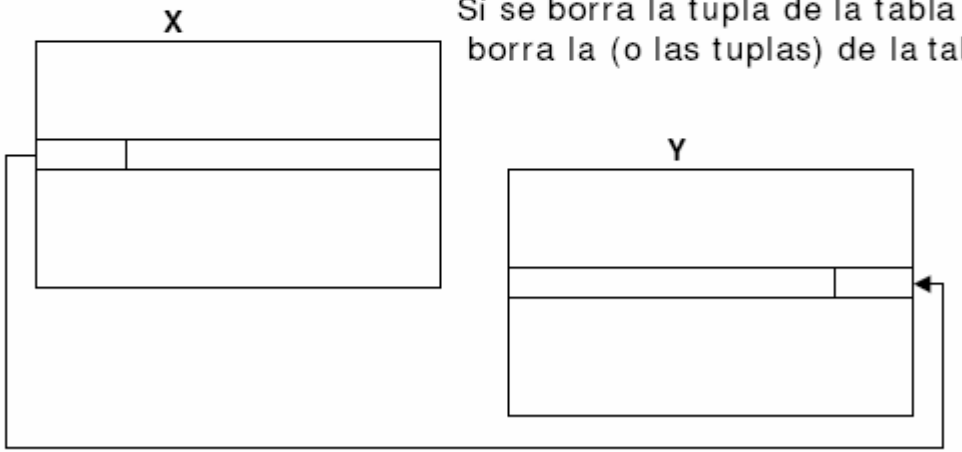
- Se refiere a la problemática de eliminar registros cuando existen referencias (references).
- ON DELETE SET NULL
- ON DELETE CASCADE
- ON UPDATE CASCADE
- ON UPDATE SET NULL



Constraint - Integridad

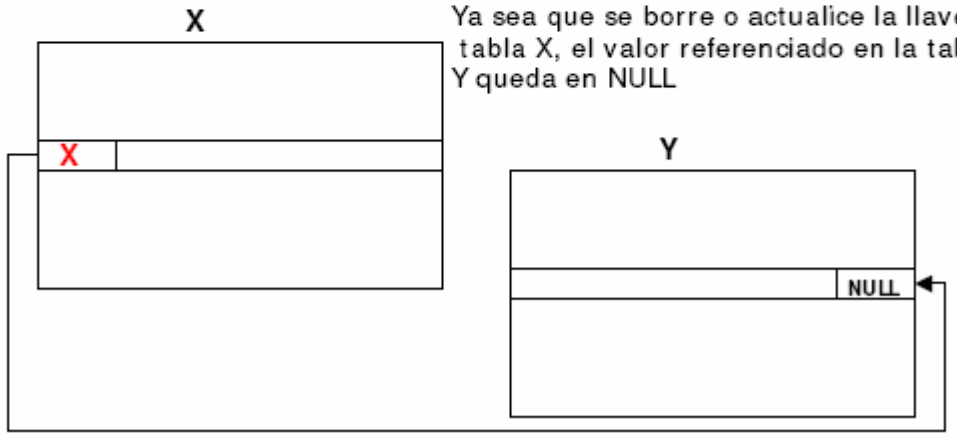
- CASCADE
  - Propaga los cambios

Si se borra la tupla de la tabla X, se borra la (o las tuplas) de la tabla Y



Constraint - Integridad

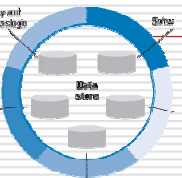
- SET NULL
  - Al producirse un cambio en una llave primaria, sus referencias quedan nulas.



Ya sea que se borre o actualice la llave de la tabla X, el valor referenciado en la tabla Y queda en NULL

Constraint

- Más información en PostgreSQL 7.2 Reference Manual, capítulo 1 CREATE TABLE



Fin Primera Parte

- Al final de esta primera parte hemos revisado los siguientes conceptos:
  - SQL Embedded.
  - JDBC
  - Aspectos sistémicos de SQL.
  - Triggers
  - Constraints