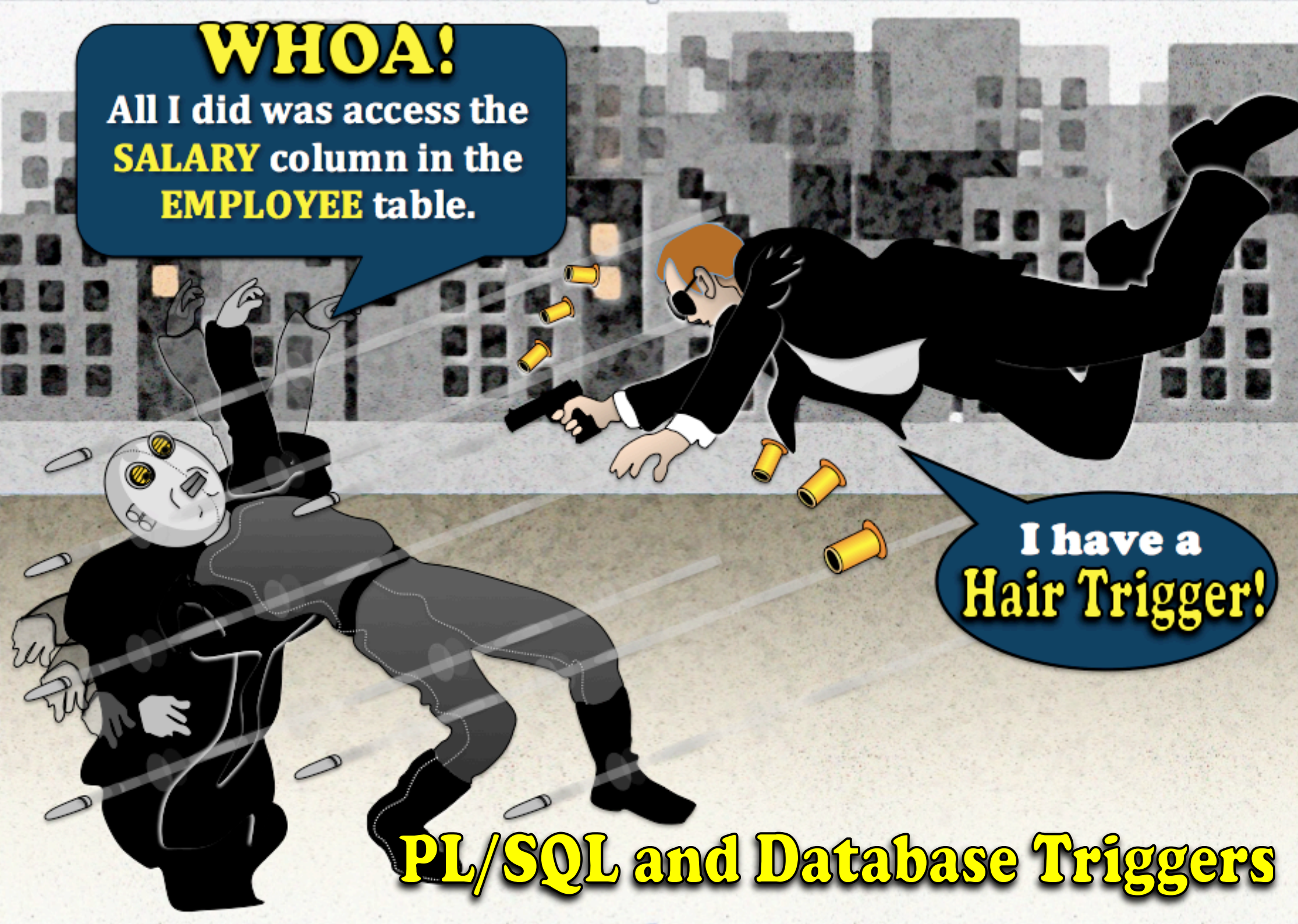
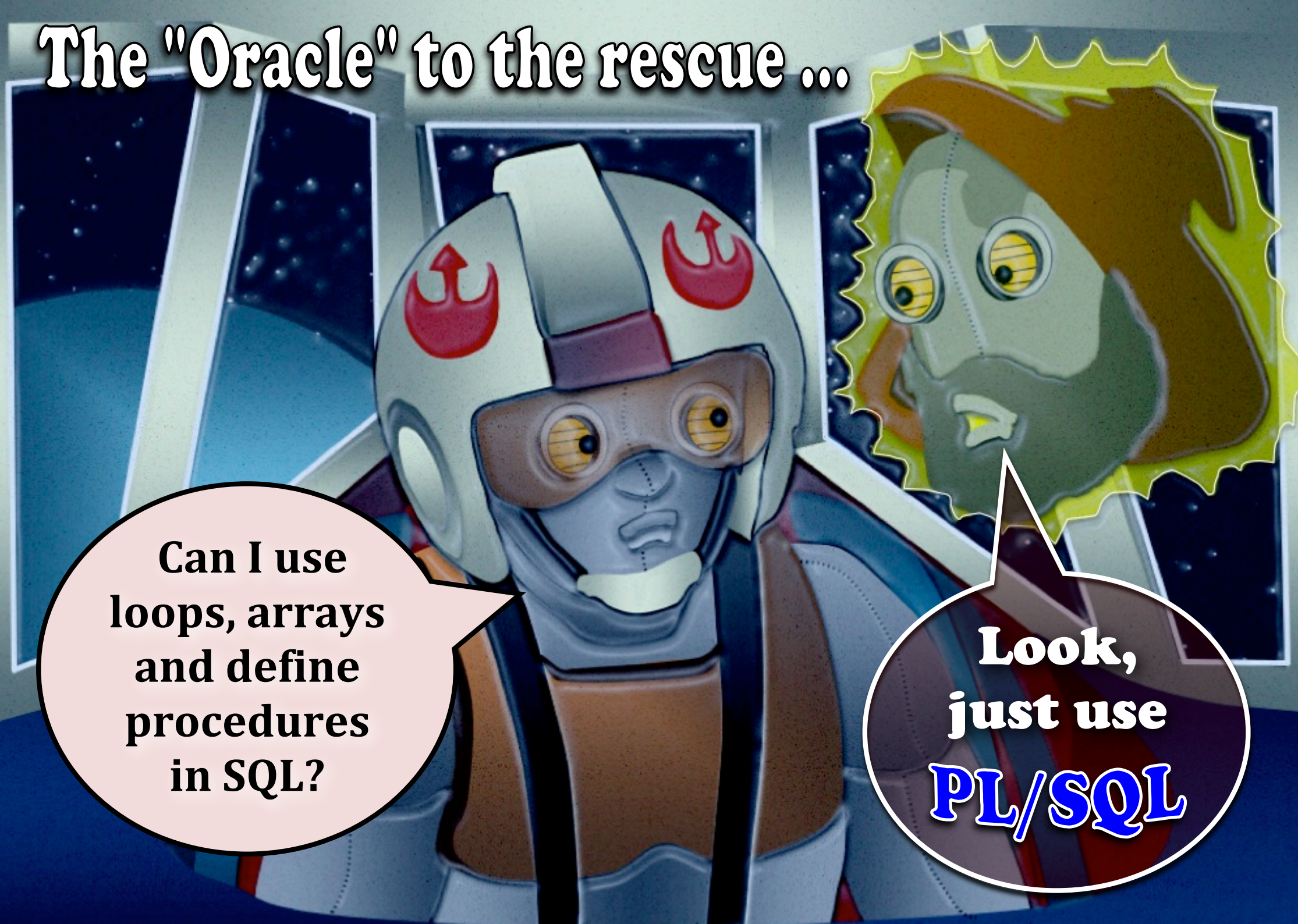

WHOA!
All I did was access the SALARY column in the EMPLOYEE table.
I have a Hair Trigger!
PL/SQL and Database Triggers


The "Oracle" to the rescue ...
Can I use loops, arrays and define procedures in SQL?
Look, just use PL/SQL

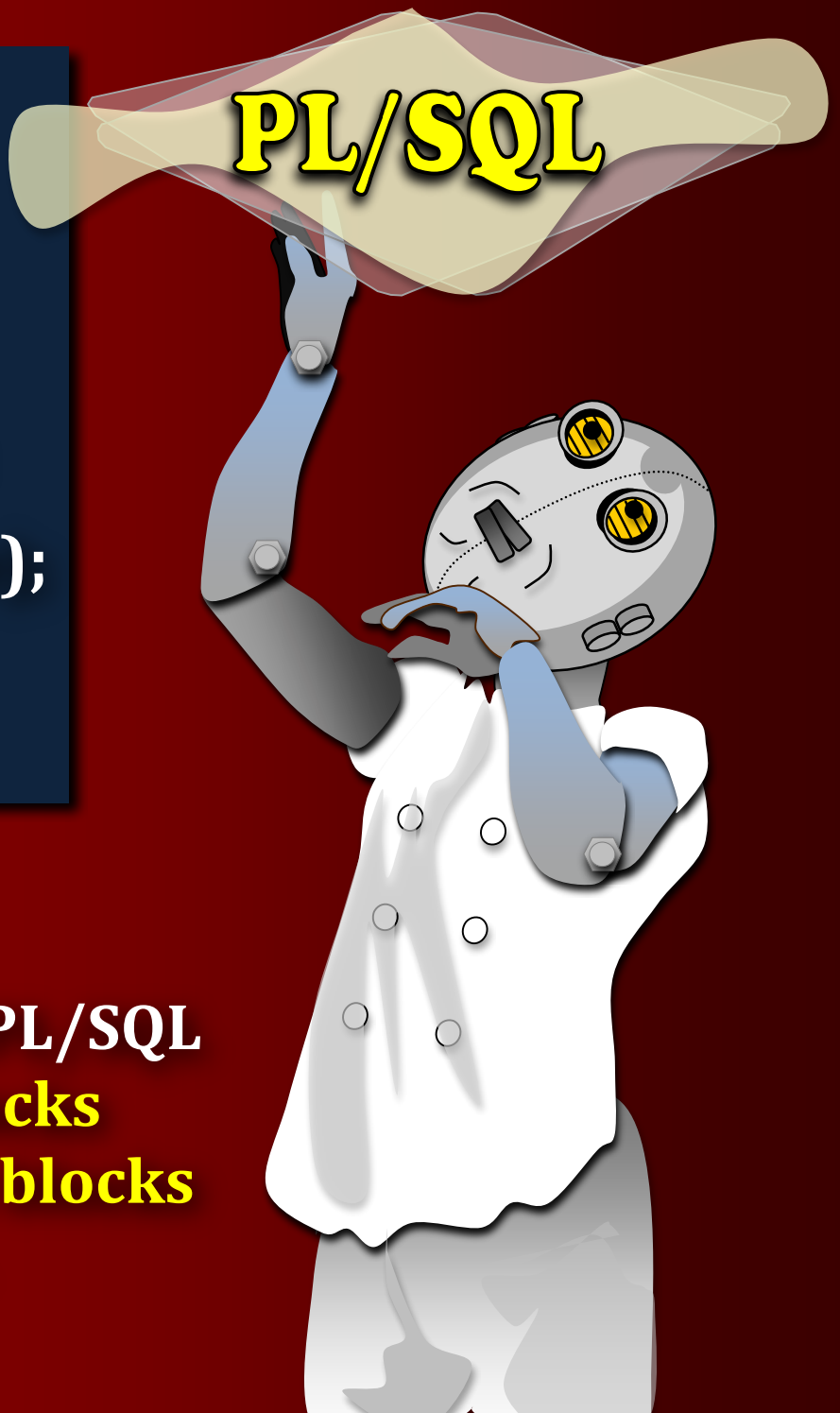
PL/SQL

Programming Language for SQL

PL/SQL is a programming language designed by Oracle and embedded into Oracle implementations of SQL databases.

PL/SQL allows DB managers to program in a procedural language that uses SQL constructs as data structures.

```
DECLARE
  Message Varchar2(20);
BEGIN
  Message := 'Where's my Pizza';
  dbms_output.put_line(Message);
END
```



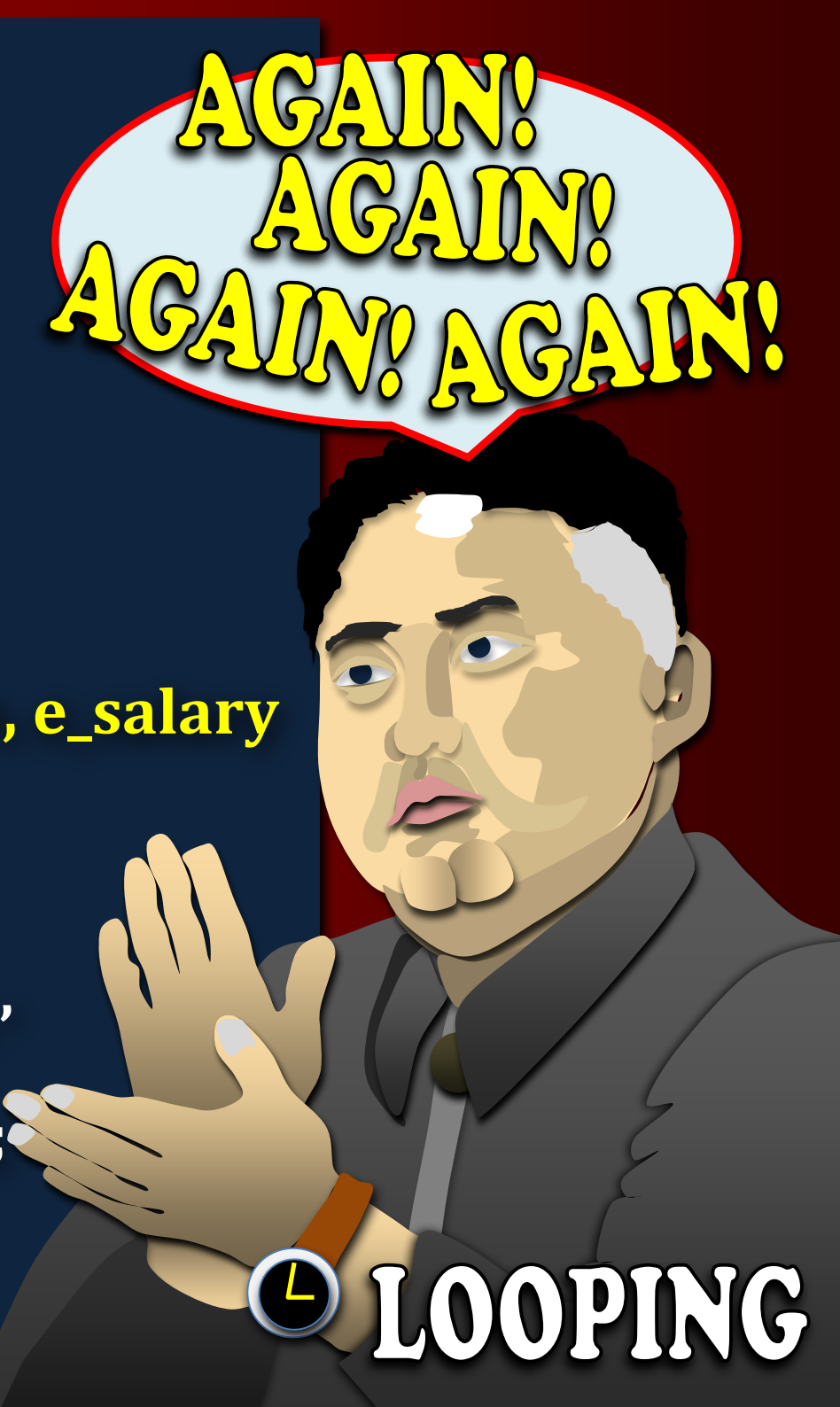
As with most procedural languages, PL/SQL programs contain **declaration blocks** (variable definitions) and **execution blocks** (statements to be executed).

Define your own Data-Types in PL/SQL

```
DECLARE
  Subtype name IS char(20);
  Subtype address IS Varchar2(40);
  who name;
  where  address;
BEGIN
  who := 'Donald Trump';
  where := 'Cloud Cuckoo land';
  dbms_output.put_line('Hello '  ||
                    who || ' of ' || where);
END
```

```
DECLARE
  e_id Employees.ID%type := 1;
  e_name Employees.Name%type;
  e_addr Employees.Address%type;
  e_salary Employees.Salary%type;
BEGIN
  Select Name, Address, Salary
  INTO  e_name, e_addr, e_salary
  From Employees Where  ID = e_id;
  dbms_output.put_line('Employee '
        ||  e_name  || ' from  ' || e_addr
        || ' earns  ' || e_salary );
END
```

```
DECLARE
  e_name Employees.Name%type;
  e_salary Employees.Salary%type;
BEGIN
  << id_loop >>
  FOR e_id in  1...1000 LOOP
     Select Name, Salary INTO e_name, e_salary
     From Employees
     Where  ID = e_id;
     dbms_output.put_line('Employee '
        || e_name || ' earns ' || e_salary);
   END LOOP  id_loop;
END
```




```
DECLARE
  e_name Employees.Name%type;
  type salary_list IS Varray(1000)
                 of Employees.Salary%type;
  e_sals salary_list := salary_list( );
BEGIN
  << id_loop >>
  FOR e_id in  1...1000 LOOP
     Select Name, Salary INTO e_name, e_sals(e_id)
     From Employees  Where  ID = e_id;
     dbms_output.put_line(e_name || ' earns  ' || e_sals(e_id));
   END LOOP  id_loop;
END
```

```
Create or Replace Procedure deny AS
BEGIN
  dbms_output.put_line('Fake News!');
END


BEGIN
   << deny_loop >>
   FOR i in  1...1000 LOOP
      deny;
   END LOOP deny_loop;
END
```

Defining and Calling PROCEDURES

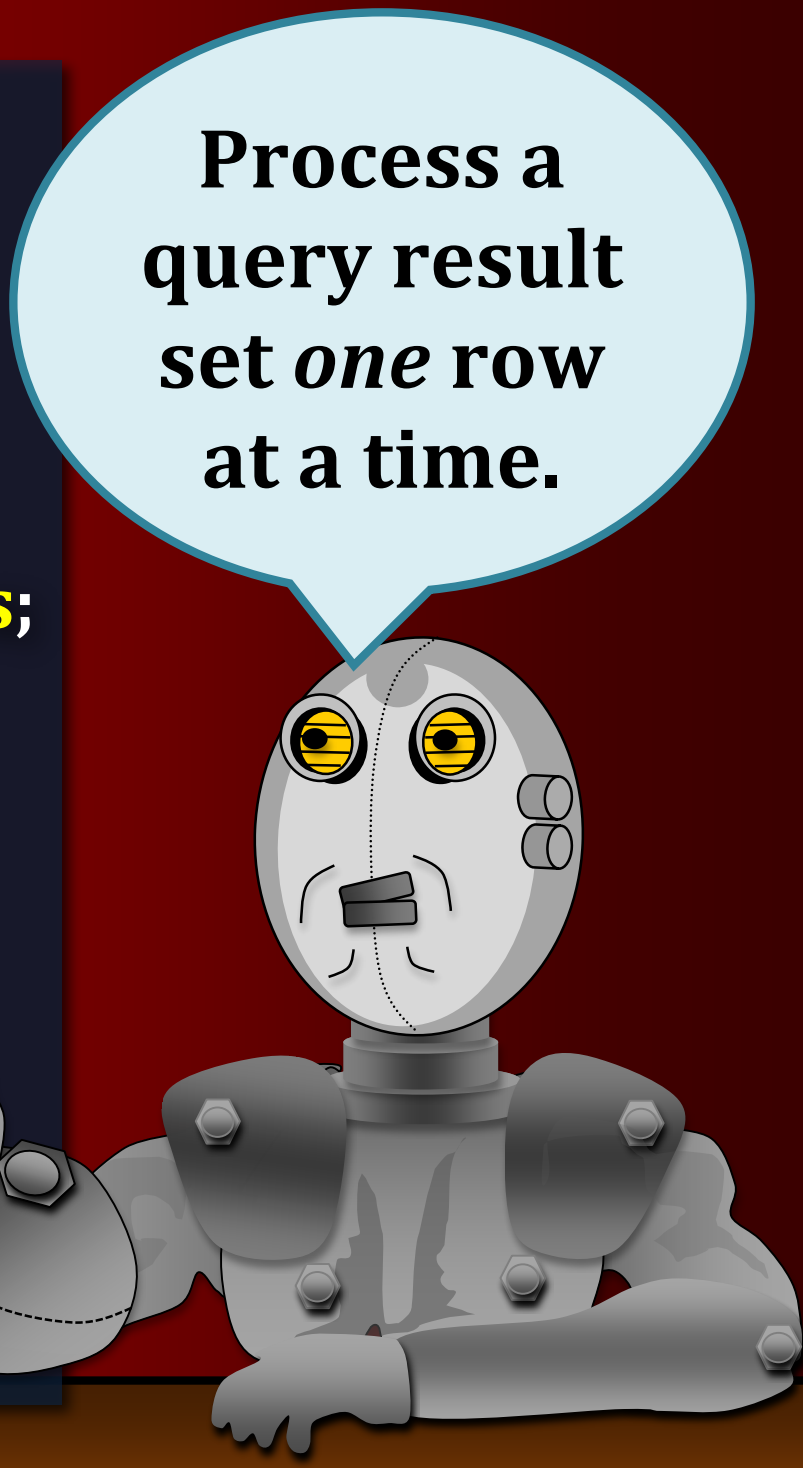
```
Create or Replace Function avgSalary
Return number IS
  avgSal number (7,2) := 0;
BEGIN
  Select avg(Salary) INTO avgSal
    FROM Employees;
  Return avgSal;
END
DECLARE mean number(7,2) :=  0;
BEGIN
   mean =  avgSalary( );
END
```

Defining and Calling FUNCTIONS

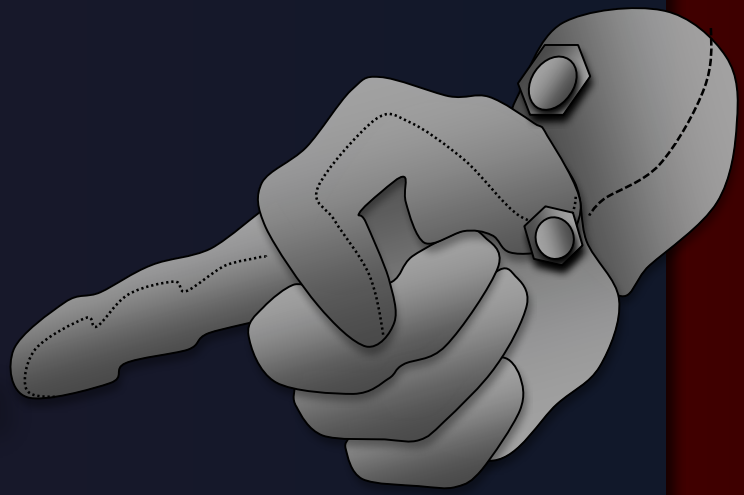
```
Procedure reward (bonus IN number) IS
BEGIN
  UPDATE Employees
     SET  Salary = Salary + bonus
     WHERE Name like '%Trump';
END
DECLARE mean number(7,2) :=  0;
BEGIN
   mean =  avgSalary( );
   reward(2*mean);
END
```

Passing Parameters to Functions & Procedures

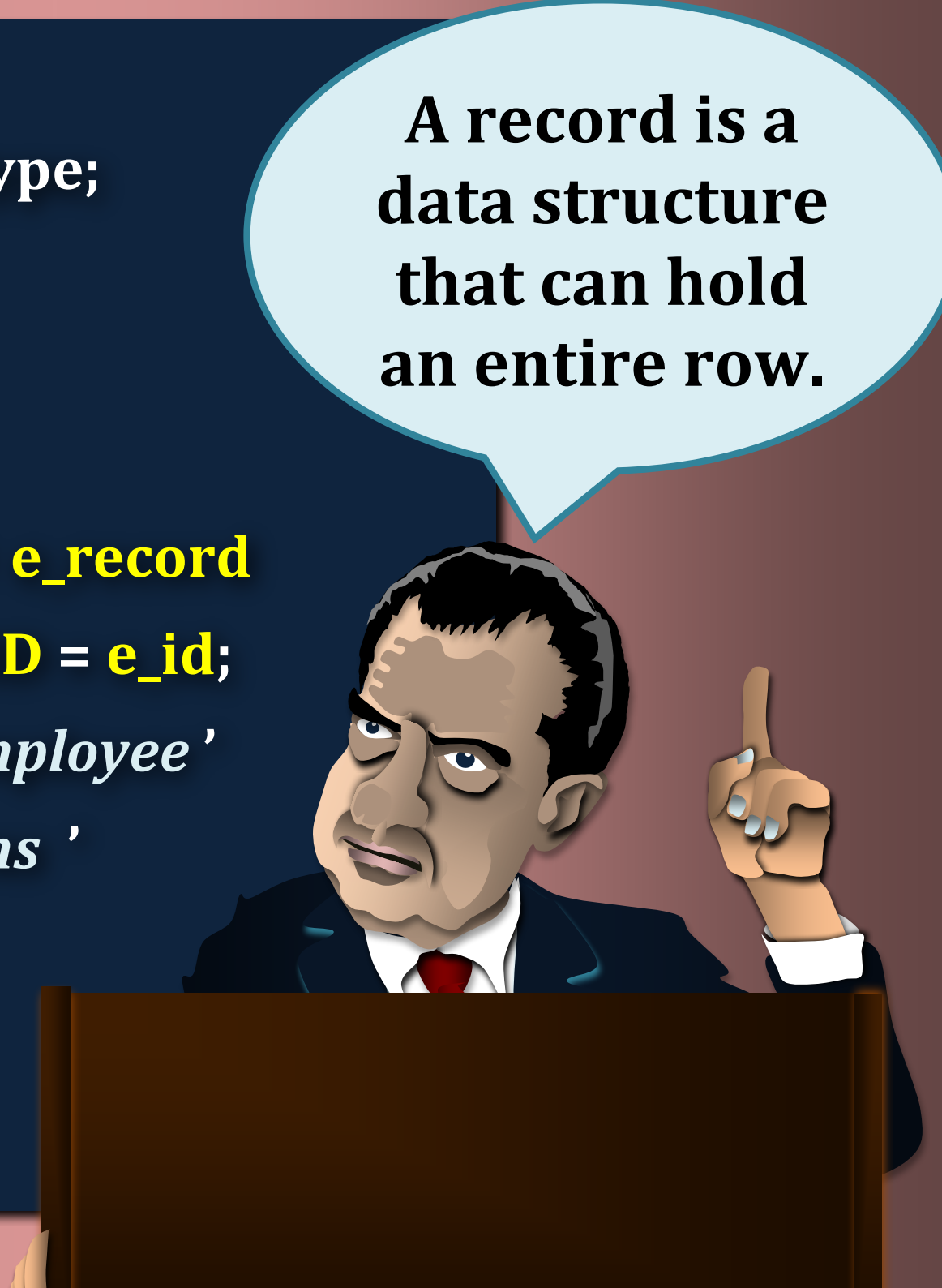
```
DECLARE
  e_name Employees.Name%type;
  e_salary Employees.Salary%type;
  Cursor e_cursor IS
   SELECT Name, Salary from Employees;
BEGIN
   OPEN e_cursor;
   -- Fetch one row after another
   -- Do work with each row in
   CLOSE e_cursor;
END
```



```
Cursor e_cursor IS
   SELECT Name, Salary from Employees;
BEGIN
  OPEN e_cursor;
  << fetch_loop >>
  Fetch e_cursor into e_name, e_salary;
     Exit When e_cursor%notfound;
  dbms_output.put_line(e_name || ' earns ' || e_salary );
  END LOOP fetch_loop;
  CLOSE e_cursor;
END
```

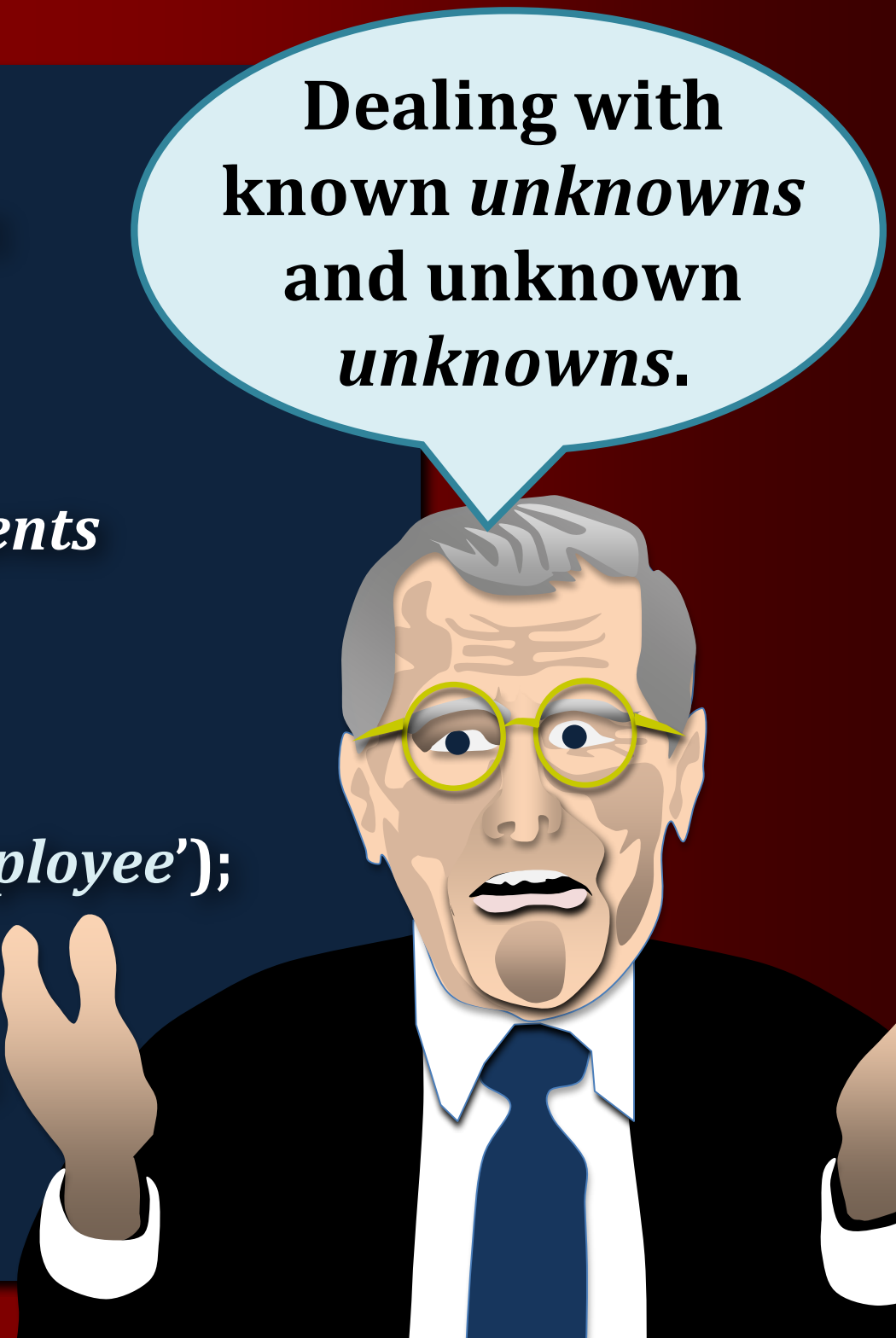


```
DECLARE
 e_record Employees%rowtype;
BEGIN
 << id_loop >>
 FOR e_id in  1...1000 LOOP
    Select Name, Salary INTO  e_record
    From Employees Where  ID = e_id;
    dbms_output.put_line('Employee'
      || e_record.Name || ' earns '
      || e_record.Salary);
  END LOOP  id_loop;
END
```



```
DECLARE
 e_record Employees%rowtype;
BEGIN
 -- Do work
 -- Load rows with Select statements
 -- Process each row in turn

 WHEN no_data_found THEN
    dbms_output.put_line('No Employee');
 WHEN others THEN
  dbms_output.put_line('Error');
END
```

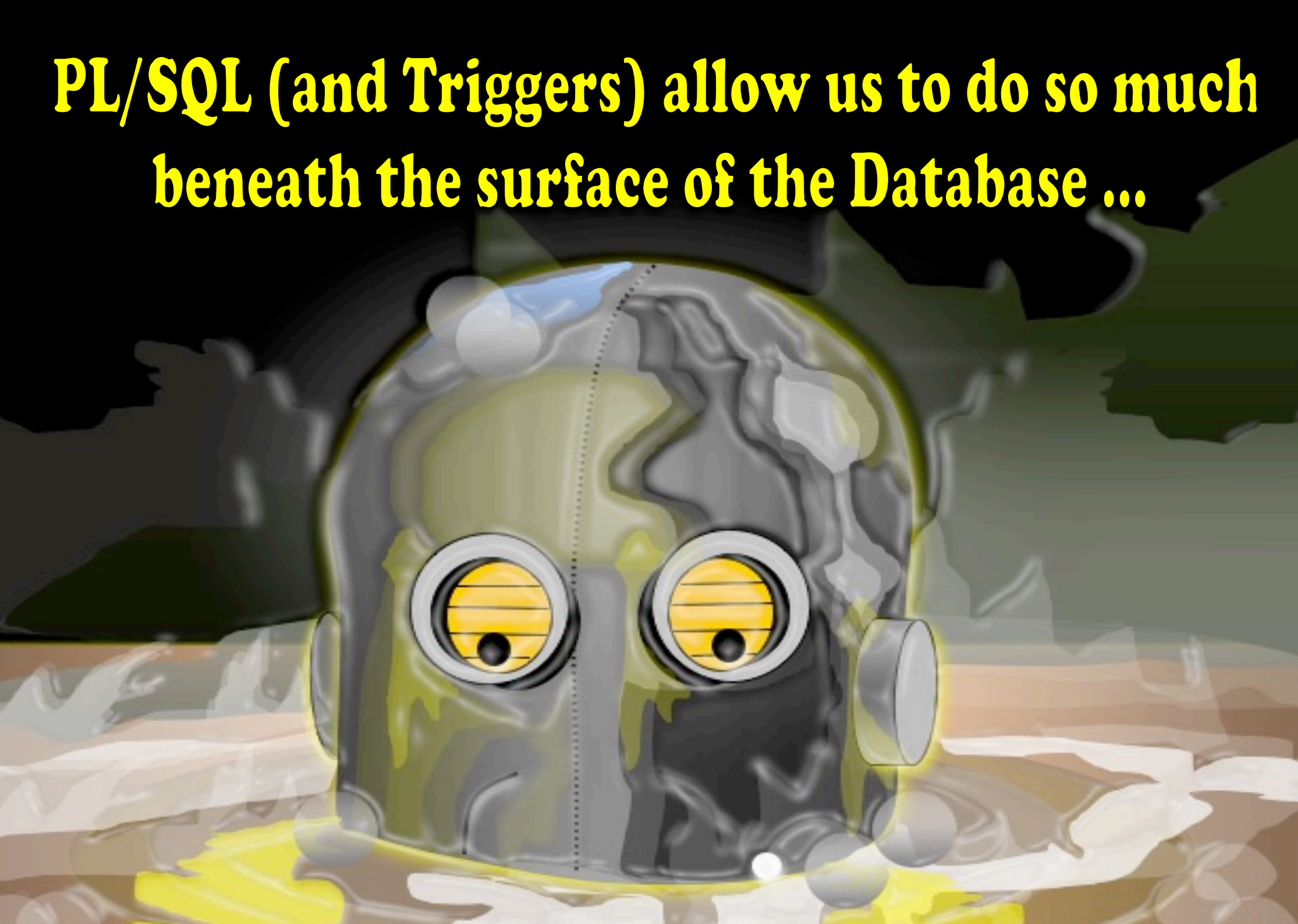


```
Create or Replace Trigger
      show_salary_changes
Before Delete or Insert or Update
      on Employees
For Each Row
      When (New.ID > 0)
DECLARE  sal_delta number;
BEGIN
 sal_delta := :New.Salary - :Old.Salary;
 dbms_output.put_line('Employee: '
     || :Old.Name || ' change ' || sal_delta);
END
```

A "trigger" is a routine that is activated by a DB change.

```
Create or Replace Trigger
      Insert_employee
Instead Of   Insert
      on Employees
For Each Row
      When (New.Salary > 100000)


BEGIN
  dbms_output.put_line('Excess salary: '
     || :New.Name || ' is too high at '
     || :New.Salary );
END
```


PL/SQL (and Triggers) allow us to do so much
beneath the surface of the Database ...