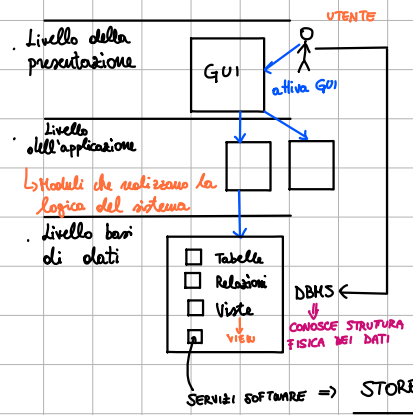
ACCESSO DA SOFTWARE



RIGHE BLU: giusto path applicativo

=> STORED PROCEDURE: procedura che mettiamo dentro il DBMS

1) Progettare servizi riutilizzabili
2) Lascio che DBMS faccia lavoro di OTTIMIZZAZIONE

TIPI DI ACCESSO:

1) STORED PROCEDURE => INTERNO => PL/pgSQL
2) ACCESSO ESTERNO (JDBC) => JAVA

ACCESSO INTERNO

- Utilizziamo PL/pgSQL => programma procedurale da PostgreSQL
- Linguaggio TURING-COMPLETO => stesso potere computazionale di una Macchina di Turing

SQL non è TURING-COMPLETO

SINTASSI:

```
CREATE FUNCTION function_name (arg1 arg_type, ..., argn arg_type)
RETURNS {return_type | TABLE(col1 col_type, ..., coln col_type) | TRIGGER} AS
```

se VOID: la funzione è una PROCEDURA

```
$$
DECLARE
    <dichiarazioni di variabili>;
BEGIN
    <istruzioni>
END;
$$ LANGUAGE plpgsql
```

può essere anche un altro simbolo

ESEMPIO

Fattura

SOGGETTO	IMPONIBILE	ALIQUOTA
A	1000	22

```
CREATE TABLE fattura (soggetto VARCHAR(100), imponibile REAL,
aliquota INTEGER);
INSERT INTO Fattura VALUES ('a', 1000.00, 22), ('a', 30.00, 4), ('b',
10.000, 10);
```

Supponiamo adesso di voler calcolare, dato un soggetto, una tabella con una riga che indichi tale soggetto ed il totale che esso dovrà versare di tasse. Lo possiamo fare attraverso la seguente stored procedure:

```
CREATE OR REPLACE FUNCTION totale_fattura(sogg VARCHAR)
RETURNS TABLE (sgt VARCHAR, tot REAL) AS
$$
  DECLARE
    r record; totale REAL;
  BEGIN
    totale := 0;
    FOR r IN SELECT * FROM Fattura WHERE soggetto=sogg LOOP
      totale := totale + r.imponibile*(100+r.aliquota)/100;
    END LOOP;
    CREATE TEMPORARY TABLE return_table (sgt VARCHAR, tot REAL);
    INSERT INTO return_table VALUES (sogg, totale);
    RETURN QUERY SELECT * FROM return_table;
  END;
$$ LANGUAGE plpgsql;
```

VARIABILI LOCALI

tipo generico per TUPLA (tipo tupla)

RITORNA UNA TABELLA (TEMPORANEA) → 2 attributi: . SGT . TOT

solo RETURN return_table : ERRORE

TRIGGER

Voglio contare quante volte qualcuno controlla le mie fatture e anche il totale

CONTROLLO

sog	totale
X	100
Y	200

=> aggiornamento tabella deve essere vicino ai dati => DBMS

Un TRIGGER dice: quando succede un certo evento su una tabella, chiama una STORED PROCEDURE/FUNZIONE

SINTASSI:

```
CREATE TRIGGER trigger_name [BEFORE | AFTER | INSTEAD OF]
[INSERT | DELETE | UPDATE] ON table_name
    [FOR EACH ROW | FOR EACH STATEMENT] EXECUTE PROCEDURE procedure_name
```

(BEFORE | AFTER | INSTEAD OF: prima | dopo | invece dell'evento)

(INSERT | DELETE | UPDATE: => L'evento che attiva il trigger)

(FOR EACH ROW: riga-tupla — Attivato per ogni tupla coinvolta nella modifica)

(FOR EACH STATEMENT: Attivato una volta per ciascuna operazione di modifica)

(procedure_name: deve avere RETURN TRIGGER)

- All'interno del blocco di istruzioni di una funzione definita per un trigger (cioè il cui tipo è TRIGGER), sono disponibili alcune variabili speciali di tipo record tra cui:
  - NEW: rappresenta la nuova tupla per operazioni INSERT/UPDATE nei trigger FOR EACH ROW. Viene settato a NULL nei trigger FOR EACH STATEMENT oppure per operazioni di DELETE.
  - OLD: rappresenta la vecchia tupla per operazioni UPDATE/DELETE nei trigger FOR EACH ROW. Viene settato a NULL nei trigger FOR EACH STATEMENT oppure per operazioni di INSERT.

ESEMPIO

Lavoratore

CODICE	NOME	COGNOME	PROGETTO	SINDACATO

PROGETTO

CODICE	USURA	RESPONSABILE

Quando inserisco nuovo progetto, lo dò a ogni lavoratore che non appartiene a un progetto

```
CREATE OR REPLACE FUNCTION assegna_progetto() RETURNS
TRIGGER AS
$$
  BEGIN
    UPDATE lavoratore
    SET inprogetto=NEW.codice WHERE inprogetto IS NULL;
    RETURN NEW;
  END;
$$ language plpgsql;

CREATE TRIGGER trigger_progetto AFTER INSERT ON progetto
  FOR EACH ROW EXECUTE PROCEDURE assegna_progetto();
```