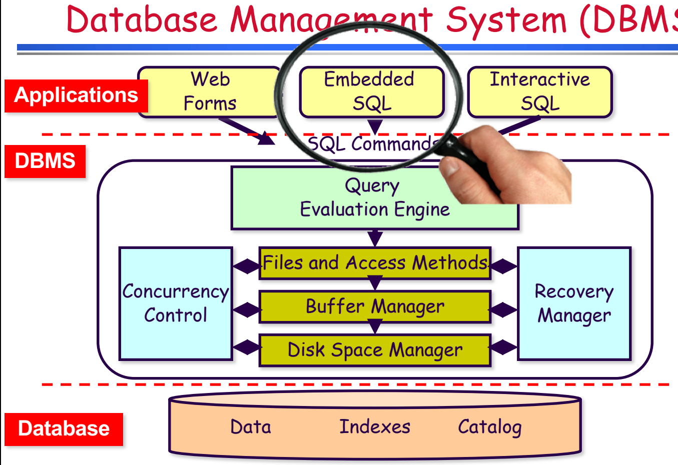
Database Programming at Large

Stored Procedures and Embedded SQL

Database Management System (DBMS)



Database Programming

- Objective:
  - To access a database from an **application** program (as opposed to interactive interfaces)
- Why?
  - An interactive interface is convenient but not sufficient
    - A majority of database operations are made thru application programs (increasingly thru web applications)

Database Programming Approaches

- Embedded commands:
  - Database commands are **embedded** in a general-purpose programming language
- Library of database functions:
  - Available to the host language for database calls; known as an **API** (Application Program Interface)
  - *e.g., JDBC, ODBC, PHP, Python*
- A brand new, full-fledged language
  - **PL/SQL: P**rocedural **L**anguage extensions to SQL
  - e.g., Postgres PL/pgSQL, Oracle PL/SQL,

Approach 3: SQL/PL

- Functions/procedures can be written in SQL itself, or in an external programming language
- Functions are very useful with specialized data types
  - E.g. functions to check if polygons overlap, or to compare images for similarity
- Some databases support **table-valued functions**, which can return a relation as a result
- SQL3 also supports a rich set of imperative constructs
  - Loops, if-then-else, case, assignment + exception handling
  - Similar to CSH script language
- Many DBMS have proprietary procedural extensions to SQL that differ from SQL3.

ANSI SQL Functions

authors (author, title, author_order)

- Definition of a Function

```
create or replace function author_count (name varchar(20))
    return integer
a_count integer;              -- local variable declaration
begin
    select count(author) into a_count    -- into is a tuple assignment operator
    from authors
    where authors.title=name;
    return a_count;
end;
/
```

- '/': Executes a PL/SQL block
- Invocation ?

```
SELECT title, author_count(title)
FROM books4
WHERE author_count(title)> 1;
```

PL/pgSQL Function

- Create a function statement

```
CREATE [OR REPLACE] FUNCTION func_name(…) RETURNS r_type AS
 $$
 [ DECLARE
   declarations ]
 BEGIN
     statements
 END;
 $$ LANGUAGE plpgsql;
```

- `LANGUAGE plpgsql` can either appear before the top $$ or after the bottom $$
- Drop a function statement

```
DROP FUNCTION [IF EXISTS] func_name() [CASCADE|RESTRICT];
```

PL/pgSQL Example Function

```
create or replace function author_count (name varchar(20))
        returns integer as
    $$
    declare
    a_count integer;              -- local variable declaration
    begin
        select count(author) into a_count
         from authors
        where authors.title=name;
        return a_count;
    end;
    $$ LANGUAGE plpgsql;
```

Trigger example in Postgres

```
CREATE TRIGGER Name_Trim
     BEFORE INSERT
     ON Student
     FOR EACH ROW
        WHEN (NEW.Name IS NOT NULL)
     EXECUTE FUNCTION trim_spaces_name();
```

PL/pgSQL Trigger Function

```
CREATE OR REPLACE FUNCTION trim_spaces_name()
   RETURNS trigger AS
$$
BEGIN
 NEW.name = LTRIM(NEW.name);
 RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';
```

More on triggers in Postgres

- CREATE [CONSTRAINT] TRIGGER *trig_name*

```
   time event
   ON table_name
   [ NOT DEFERRABLE | [ DEFERRABLE ]
     { INITIALLY IMMEDIATE | INITIALLY DEFERRED } ]
   [ FOR EACH { ROW | STATEMENT } ]
   [ WHEN ( condition ) ]
   EXECUTE {FUNCTION | PROCEDURE} func_name ();
```

- Constraint triggers must be AFTER ROW triggers.
- SET CONSTRAINTS trig_name < Evaluation Mode>

ANSI SQL Procedures

- Definition of a procedure:

```
create or replace procedure author_count_proc (in title varchar(20),
                                               out a_count integer )
  begin
    select count(author) into a_count
     from authors
     where authors.title = title;
 end;
 /
```

- Parameters Options: **IN, OUT, INOUT**
  - **Oracle syntax:** (title **in** varchar(20), a_count **out** integer)

PostgreSQL Stored Procedures

```
CREATE [OR REPLACE] PROCEDURE name(parameters)
LANGUAGE language_name
AS $$
  stored_procedure_body;
$$;
```

- Parameters Options: **IN, INOUT**, or **VARIADIC**
  - If omitted, the default is **IN**
  - There is no **OUT**
  - **VARIADIC** is array parameter
- language_name: SQL or PLpgSQL (or plpgsql)
- If you want to end a procedure earlier, you can use the **RETURN** statement with no expression as follows: RETURN;

Stored Procedure (Parameters by position)

```
CREATE OR REPLACE PROCEDURE transfer(INT, INT, DEC)
LANGUAGE plpgsql
AS $$
BEGIN
  -- subtracting the amount from the sender's account
  UPDATE accounts
  SET balance = balance - $3
  WHERE id = $1;
  -- adding the amount to the receiver's account
  UPDATE accounts
  SET balance = balance + $3
  WHERE id = $2;
END; -- or COMMIT;
$$;
```

ANSI/PGSQL Procedures: Invocation

- Procedures can be invoked either within a trigger, an SQL procedure, or from embedded SQL, using the **Call** statement.
- E.g., from an SQL procedure block

```
declare a_count integer;
begin
  call author_count_proc (`Database Systems', a_count);
  call transfer (101, 102, 300.50);
end;
```

- SQL3 allows name **overloading** for function and procedures, as long as the number or types of arguments is different.