Uložené procedury v PostgreSQL

Historie, současnost, budoucnost

Pavel Stěhule

http://www.pgsql.cz

14. 1. 2008

Uložené procedury
Zákaznický kód prováděný na databázovém serveru

Poznámky

- název technologie - **uložené procedury** (externí, SQL),
- název se používá i pro kód, který se v jiných prostředích označuje jako funkce (uložené funkce neexistují),
- veškeré uložené procedury v PostgreSQL jsou z pohledu klasických jazyků funkce,
- SQL procedury lze v PostgreSQL psát v jazycích SQL a PL/pgSQL,
- **z uložených procedur není přístup ke GUI**! Uložené procedury nemají nic společného s 4GL databázemi.

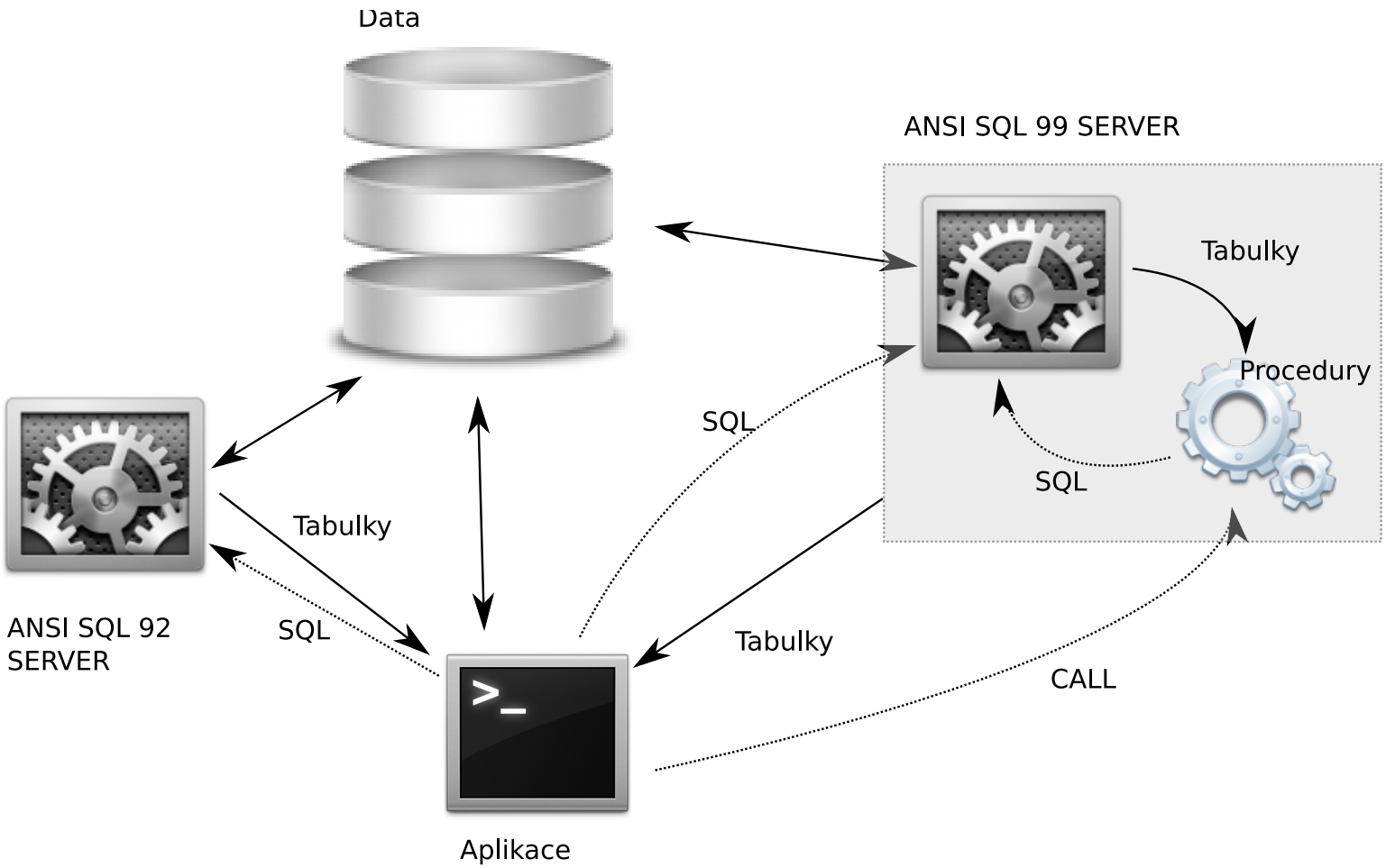
Uložené procedury
Zákaznický kód prováděný na databázovém serveru

```
postgres=# CREATE FUNCTION csum(integer, integer)
           RETURNS integer AS
           $$ SELECT $1 + $2; $$ LANGUAGE SQL;
CREATE FUNCTION

postgres=# SELECT csum(i,j)
              FROM (VALUES(10,20),
                          (30,40)) foo(i,j);
 csum
------
   30
   70
(2 rows)
```

Uložené procedury
Zákaznický kód prováděný na databázovém serveru

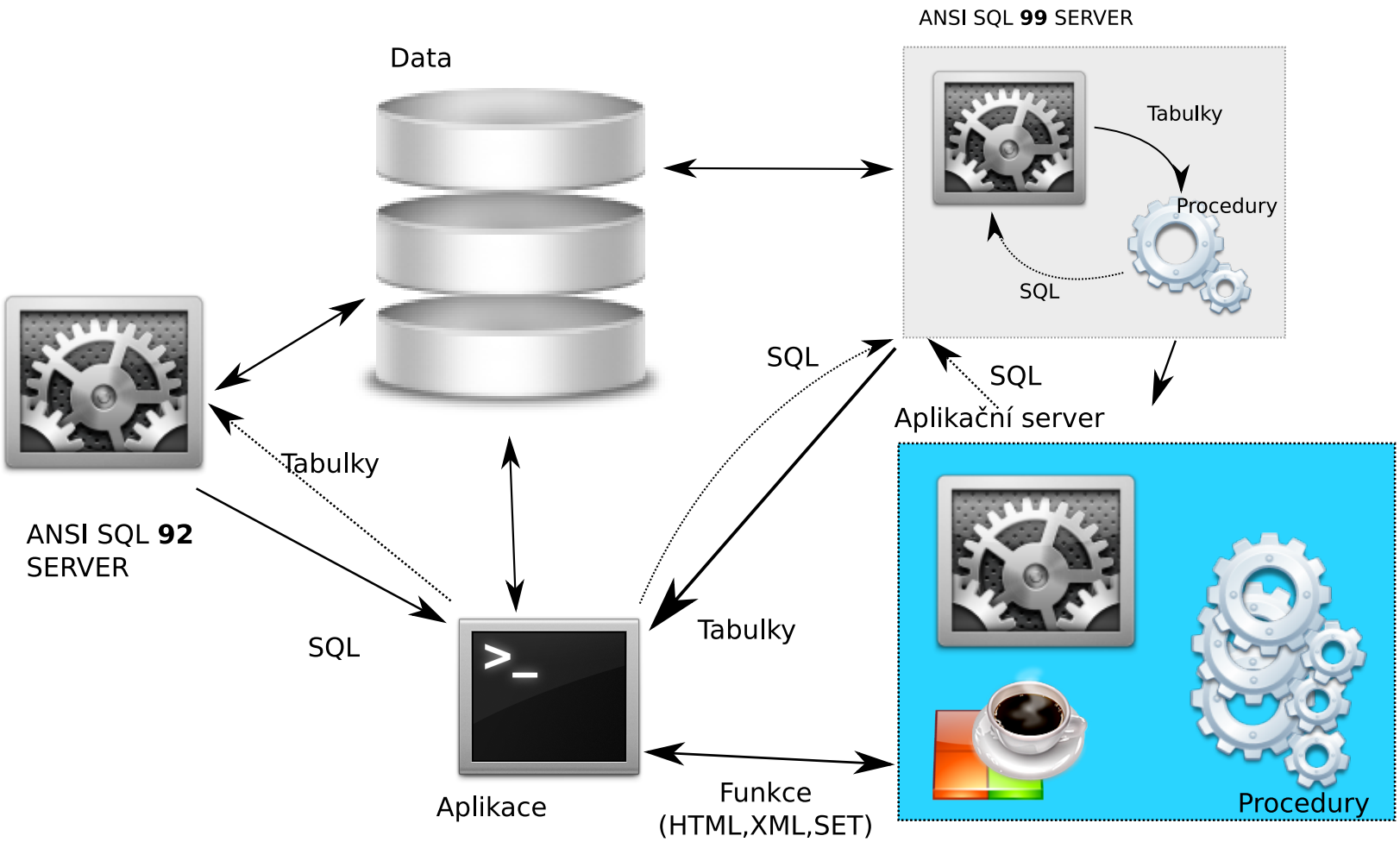
```
postgres=# CREATE FUNCTION cproc(integer)
           RETURNS void AS $$
           BEGIN
             CREATE TABLE footab(a integer);
             INSERT INTO footab
                SELECT i FROM generate_series(1, $1) g(i);
           END;
           $$ LANGUAGE plpgsql;
CREATE FUNCTION

postgres=# SELECT cproc(10);
 cproc
-------

(1 row)

postgres=#
```

Dvouvrstvá architektura
S využitím ANSI SQL Serveru



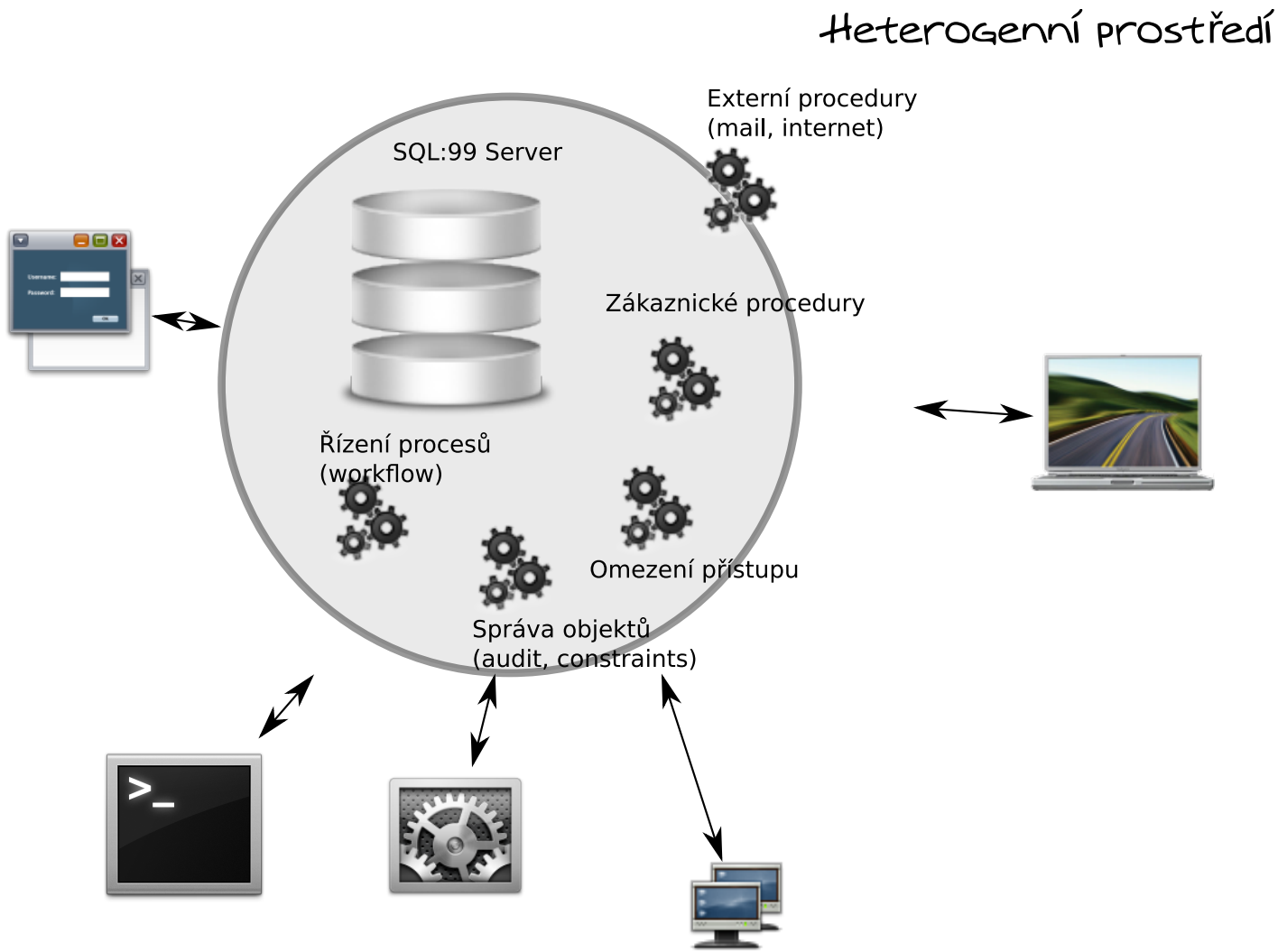
Třívrstvá a vícevrstvá architektura

S využitím ANSI SQL Serveru a aplikačního serveru



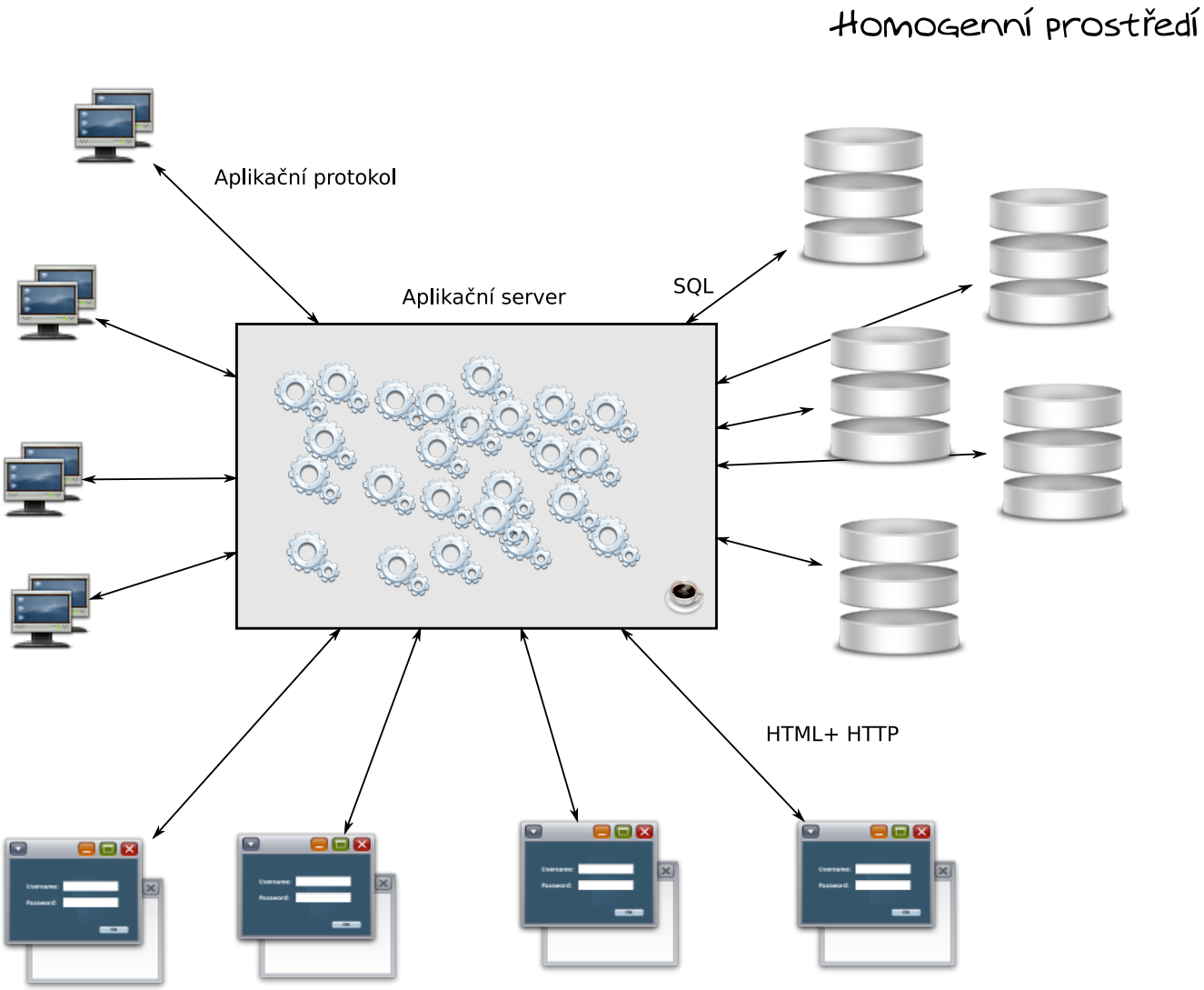
Vliv použití uložených procedur na prostředí IS

Heterogenní prostředí - Otevřená databáze



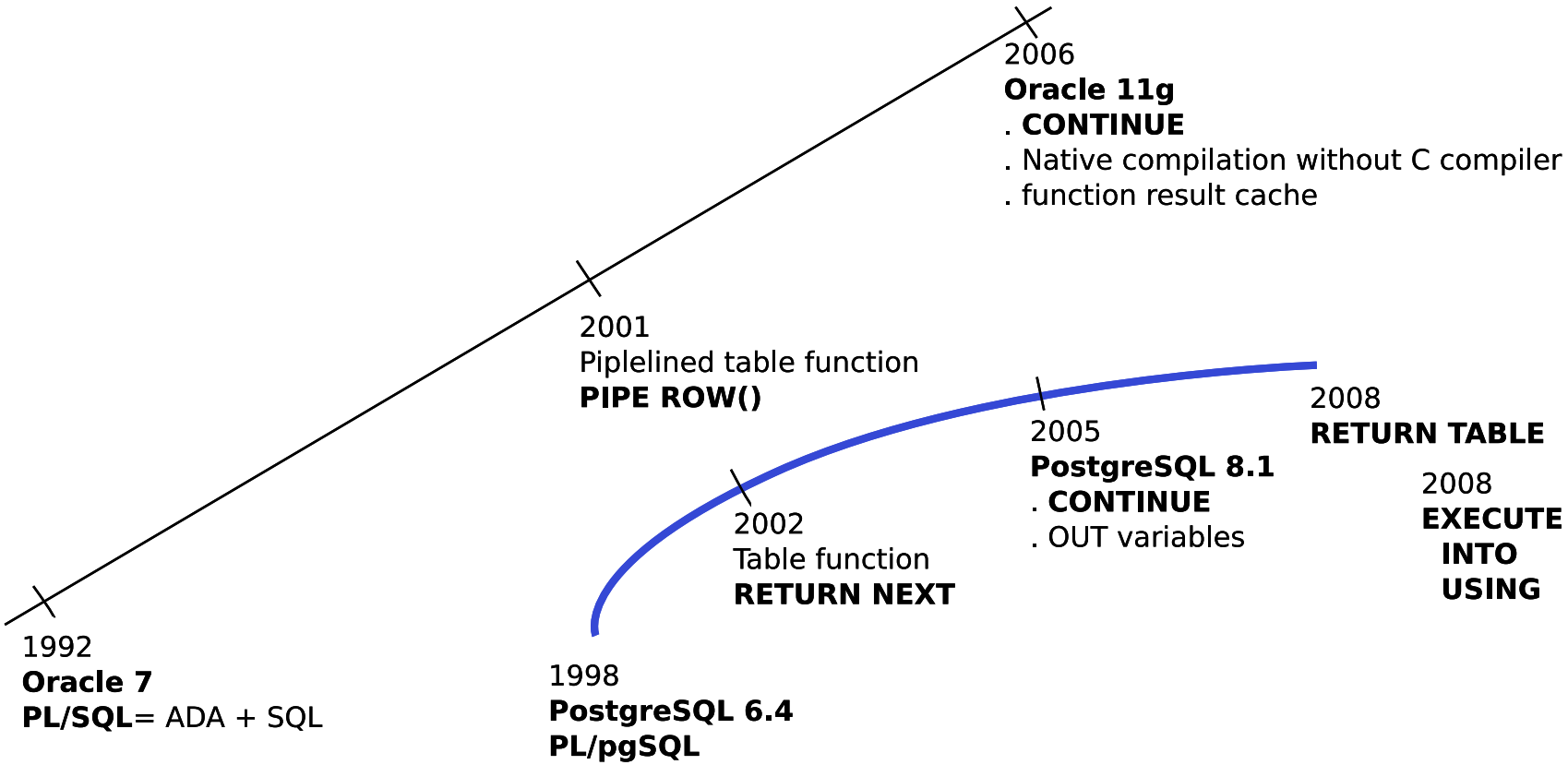
Vliv použití aplikačního serveru na prostředí IS

Homogenní prostředí - Uzavřená databáze



Vývoj PL/SQL a PL/pgSQL

Přiblížení se a vzdalování



Vztah PL/pgSQL a PL/SQL
Vztah PostgreSQL a Oracle

Poznámky

- jedním z cílů byla kompatibilita s Oraclem (i na úrovni protokolu),
- PL/pgSQL v jádře v roce 1998 (rok před ANSI SQL:99 - SQL/PSM),
- aktuální cíl - **shoda s ANSI SQL**,
- rozdíly:
  - chybějící podpora packages,
  - jiná syntaxe tabulkových funkcí (PIPE ROW x RETURN NEXT),
  - **specifický způsob předávání hodnot OUT parametrů**,
  - výkonostní rozdíly - PL/pgSQL používá i pro jednoduché operace základních datových typů (integer, varchar) SQL jádro,
  - chybí podpora procedur - v Oracle možnost řízení transakcí,
  - chybí kolekce, chybí konstrukce CASE, chybí GOTO,
  - preference Perlu před Javou

Vztah PL/pgSQL a PL/pgPSM

stejné jádro, jiný parser

Poznámky

- cílem je vytvořit **referenční implementaci** SQL/PSM v PostgreSQL,
- co lze implementovat z ANSI, co je třeba implementovat z DB2, MySQL,
- seznámit uživatele a vývojáře s jazykem SQL/PSM,
- stejné, různé:
  - stejný způsob zpracování a interpretace,
  - podobný styl jazyka (opět inspirací je ADA, Modula, ...),
  - zachycení vyjímek a **varování**, a jejich ošetření handlery,
  - proměnné kompozitních typů jsou **ukazatelé**, je třeba volat kontruktor,
  - bohatší diagnostika, přístup k zásobníku obsahující diagnostická data,
  - širší nabídka konstrukcí, chybí celočíselný FOR, iterace skrz dotaz (FOR) je jednodušší a mocnější.

PL/pgPSM
Exception and warning handlers

```
CREATE OR REPLACE FUNCTION fx(b int)
RETURNS int as
$$
  BEGIN
    DECLARE c int;
    DECLARE CONTINUE HANDLER FOR NOT FOUND PRINT 'not found';
    SET (b,c) = (SELECT a,a FROM foo);
    RETURN b+c;
  END;
$$ LANGUAGE plpgpsm;
```

Poznámka

Kromě CONTINUE handleru existuje ještě varianta EXIT a UNDO, které způsobí přerušení provádění procedury.

PL/pgPSM

Proměnné složených typů jsou reference

```
BEGIN -- plpgpsm
  DECLARE c mytype;
  SET c.a = 20; -- nelze, v tuto chvili c IS NULL
  SET c = (NULL,NULL)::mytype; -- nekompatibilni s ANSI :( chybi k
  SET c.a = 20;
  IF (c IS NULL) THEN ... -- lze

BEGIN -- plpgsql
  DECLARE c mytype;
  SET c.a = 20; -- lze
  IF (c IS NULL) THEN ... -- nelze, nema smysl c je hodnota
```

PL/pgPSM

Bohatší diagnostika, přístup k zásobníku diagnostických dat

```
BEGIN
  DECLARE frk CONDITION;
  DECLARE x, y, z text;
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
      GET STACKED DIAGNOSTICS x = CONDITION_IDENTIFIER,
         y = RETURNED_SQLSTATE;
      PRINT 'handler >', x,',', y,',', z;
      GET DIAGNOSTICS x = CONDITION_IDENTIFIER, y = RETURNED_SQLSTATE,
         z = MESSAGE_TEXT;
      PRINT 'handler >', x,',', y,',', z;
      RESIGNAL SET PG_MESSAGE_DETAIL = 'any detail';
    END;
  PRINT 'start';
  SIGNAL frk;
  SIGNAL SQLSTATE '33333' SET MESSAGE_TEXT = 'err message';
  PRINT 'finish';
END;
```

PL/pgPSM
Cyklus FOR

```
[ label ':' ]
FOR [ namespace AS ]
    [ cursor's name  CURSOR FOR ]  sql query
DO
   sql/psm statements list
END FOR [ label ]
```

```
CREATE OR REPLACE FUNCTION foo()
RETURNS void AS $$
FOR
   SELECT i FROM generate_series(1, 10) g(i)
DO
  PRINT i * 2;
END FOR;
$$ LANGUAGE plpgpsm;
```

Poznámky

Microsoft nikdy nepředpokládal širší použití uložených procedur - fy. Microsoft byla jednou z prvních propagujících architekturu využívající aplikační servery (v jejich případě založené na používání Visual Basicu):

- Úsporný jazyk - syntaxe vychází pravděpodobně z C,
- minimální shoda s ANSI SQL bez náznaků přibližování,
- nové vlastnosti povětšinou nemají oporu v ANSI SQL,
- stále chybí podpora polí,
- aktuálně intenzivní propagace externích procedur založených na .NET platformě (opět bez opory ke standardu),
- chybí podpora packages,
- nikdy nevznikly komerční knihovny.

EXECUTE INTO USING

Rychlejší a bezpečnější vyhodnocování dynamických dotazů

PL/pgSQL - PostgreSQL 8.4 (2008)

- syntaxe podobná Oracle,
- redukuje počet potenciálně nebezpečných parametrů,
- přehlednější a kratší zápis,
- odpadá převod binárních hodnot na text.

```
EXECUTE
        'SELECT * FROM '
           || CASE tblname
                  WHEN 'tab1' THEN 'tab1'
                  WHEN 'tab2' THEN 'tab2'
                  ELSE '"some is wrong"' END
           || ' WHERE c1 = $1 AND c2 = $2' -- parametry
    USING unsecure_parameter1, unsecure_parameter2;
```

PL/pgPSM - PostgreSQL 8.4 (2008)

- syntaxe odpovídá standardu SQL/PSM,
- vůči standardu chybějící podpora procedur a objektů,
- vůči PL/pgSQL bohatší jazyk, úplnější podpora vyjímek (příkazy **SIGNAL** a **RESIGNAL**),
- vůči PL/pgSQL jednodušší zápis, tělo funkce tvoří příkaz, nikoliv blok.

```
CREATE OR REPLACE FUNCTION get_sum(IN a integer, OUT b integer) AS
$$
  SET b = (SELECT sum(f.a)
             FROM Foo f
            WHERE f.a > get_sum.a);
$$ LANGUAGE plpgpsm;
```

Uložené procedury
multirecordsets, stacked recordset, unbounded selects

Skutečné procedury - PostgreSQL 8.5 (2009)

- spouštějí se příkazem **CALL** (nelze používat v kombinaci s ost. příkazy),
- struktura tabulky vrácené příkazem SELECT je známá před vyhodnocením příkazu, struktura tabulky vrácené příkazem CALL není determinována zápisem - může se určit až při vyhodnocení na základě dat.

```
CREATE OR REPLACE procedure crosstab(...) AS $$
DECLARE
  cols_expr varchar; xtab_expr varchar;
BEGIN
  EXECUTE  'SELECT array_to_string(ARRAY(SELECT ''SUM( ...
    INTO cols_expr;
  xtab_expr := 'SELECT '|| dimy_name ||', ' || cols_expr ...
  EXECUTE xtab_expr;
END;
$$ LANGUAGE plpgsql;
```

Uložené procedury

Multirecordsets, stacked recordset, unbounded selects

```
postgres=# CALL crosstab('gender', 'FROM employees', 'shop',
 'FROM employees e INNER JOIN shops s ON e.shop_id = s.id','salary');
  shop    |   m   |  f   | total
----------+-------+------+-------
New York  |     0 | 5600 |  5600
Zurich    |  4500 | 4700 |  9200
London    | 10300 |    0 | 10300
(3 rows)

CALL 0
postgres=# CALL crosstab('shop', 'FROM shops', 'gender',
  'FROM employees e INNER JOIN shops s ON e.shop_id = s.id','salary');
gender | London | New York | Zurich | total
-------+--------+----------+--------+-------
m      |  10300 |        0 |   4500 | 14800
f      |      0 |     5600 |   4700 | 10300
(2 rows)

CALL 0
```

Integrace SQL/PSM a SQL

Podmíněné provádění SQL příkazů, session proměnné

Výhody integrace - PostgreSQL 8.6 (2010)

- jednodušší a praktičtější než anonymní bloky,
- přesnější diagnostika syntaktických chyb - vše je v jednom parseru,
- jednodušší zápis - kód jako SQL příkaz, nikoliv jako řetězec.

```
DECLARE ga integer; -- session promenna
SET ga = (SELECT ....);
SELECT ... WHERE col = ga;

IF NOT EXISTS(SELECT *    -- podminene provedeni SQL skriptu
                FROM information_schema.tables
               WHERE table_name = '...') THEN
  CREATE TABLE ....
ELSE
  ...
END IF;
```

Uložené procedury v PostgreSQL

Historie, současnost, budoucnost

Pavel Stěhule

http://www.pgsql.cz

14. 1. 2008