Processing Data Inside PostgreSQL

BRUCE MOMJIAN,
ENTERPRISEDB

February, 2009

Abstract

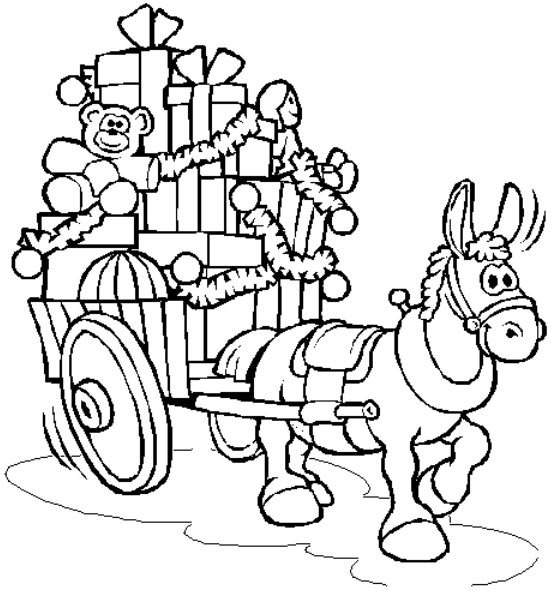
There are indisputable advantages of doing data processing in the database rather than in each application. This presentation explores the ability to push data processing into the database using SQL, functions, triggers, and the object-relational features of POSTGRESQL.

 http://momjian.us/presentations

Pre-SQL Data Access

No one wants to return to this era:

- Complex cross-table access
- Single index
- No optimizer
- Simple WHERE processing
- No aggregation



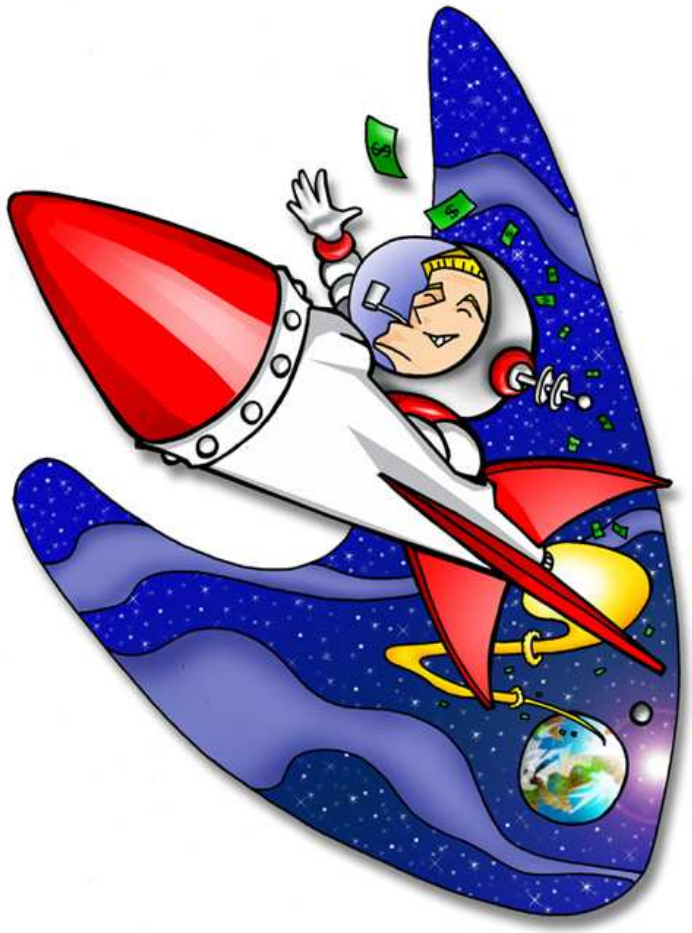
SQL Data Access

You probably take these for granted:

- Easy cross-table access, with optimizer assistance
- Complex WHERE processing
- Transaction Control
- Concurrency
- Portable language (SQL)

Post-Ingres

Welcome to the next generation of data storage.



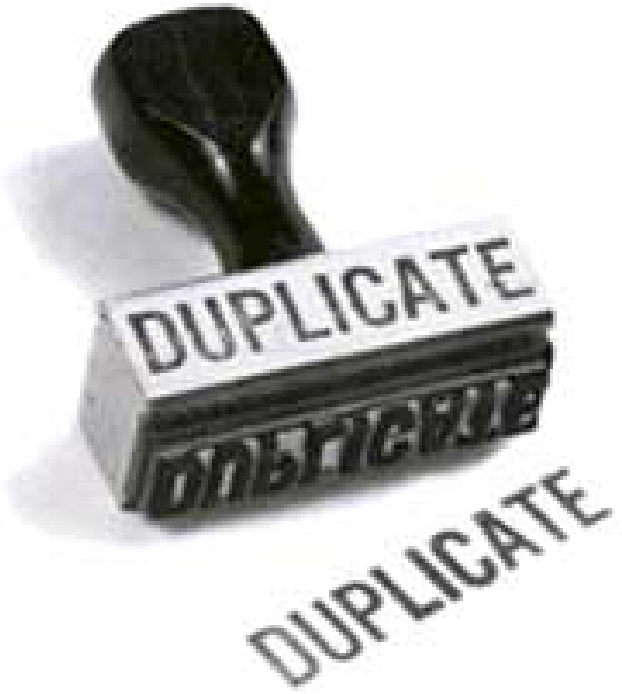
Contents

1. SQL

Make full use of the SQL tools available.

2. Functions and Triggers

Put your programs
in the database.



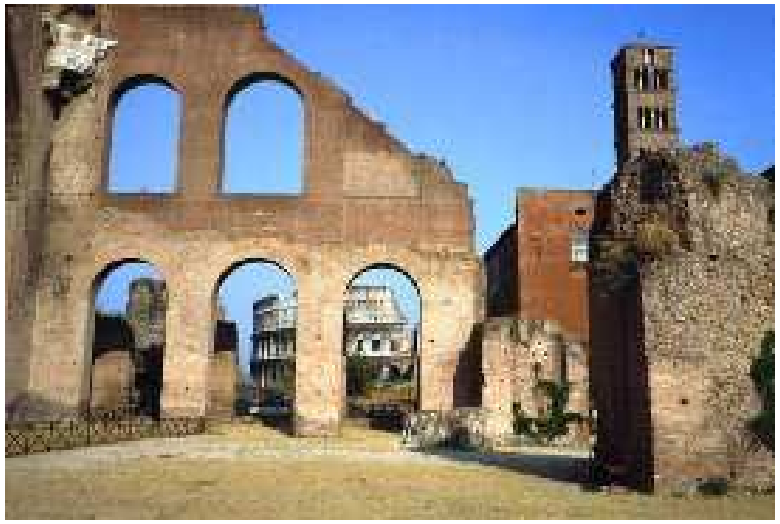
3. Customizing Database Features

Change the
database features.

1. SQL

Table Constraints

Table creation requires concentration.



Unique Test in an Application

```
BEGIN;
LOCK tab;
SELECT ... WHERE col = key;
if not found
    INSERT (or UPDATE)
COMMIT;
```

UNIQUE Constraint

```
CREATE TABLE tab
(
    col ... UNIQUE
);
CREATE TABLE customer (id INTEGER UNIQUE);
```

Preventing NULLs

```
if (col != NULL)
    INSERT/UPDATE;
```

NOT NULL Constraint

```
CREATE TABLE tab
(
    col ... NOT NULL
);
CREATE TABLE customer (name TEXT NOT NULL);
```

Primary Key Constraint

- UNIQUE
- NOT NULL

```
CREATE TABLE customer (id INTEGER PRIMARY KEY);
```

Ensuring Table Linkage
Foreign —> Primary

```
BEGIN;
SELECT *
FROM primary
WHERE key = col
FOR UPDATE;
if found
    INSERT (or UPDATE) INTO foreign;
COMMIT;
```

Ensuring Table Linkage
Primary —> Foreign

```
BEGIN;
SELECT *
FROM foreign
WHERE col = key
FOR UPDATE;
if found
    ?
UPDATE/DELETE primary;
COMMIT;
```

Ensuring Table Linkage Example

```
CREATE TABLE statename (
                    code  CHAR(2) PRIMARY KEY,
                    name  VARCHAR(30)
);

CREATE TABLE customer
(
                          customer_id INTEGER,
                          name        VARCHAR(30),
                          telephone   VARCHAR(20),
                          street      VARCHAR(40),
                          city        VARCHAR(25),
                          state       CHAR(2) REFERENCES statename,
                          zipcode     CHAR(10),
                          country     VARCHAR(20)
);
```

Ensuring Table Linkage
Larger Example

```
CREATE TABLE customer
(
                        customer_id INTEGER PRIMARY KEY,
                        name        VARCHAR(30),
                        telephone   VARCHAR(20),
                        street      VARCHAR(40),
                        city        VARCHAR(25),
                        state       CHAR(2),
                        zipcode     CHAR(10),
                        country     VARCHAR(20)
);

CREATE TABLE employee
(
                        employee_id INTEGER PRIMARY KEY,
                        name        VARCHAR(30),
                        hire_date   DATE
);

CREATE TABLE part (
```

```
                    part_id     INTEGER PRIMARY KEY,
                    name        VARCHAR(30),
                    cost        NUMERIC(8,2),
                    weight      FLOAT
);

CREATE TABLE salesorder (
                         order_id      INTEGER,
                         customer_id   INTEGER REFERENCES customer,
                         employee_id   INTEGER REFERENCES employee,
                         part_id       INTEGER REFERENCES part,
                         order_date    DATE,
                         ship_date     DATE,
                         payment       NUMERIC(8,2)
);
```

Ensuring Table Linkage
Prevent Change to Primary

```
BEGIN;
SELECT ...
FROM foreign
WHERE col = key
FOR UPDATE;
IF found
    ABORT;
UPDATE/DELETE primary;
COMMIT;
```

Ensuring Table Linkage
REFERENCES Constraint
NO ACTION/RESTRICT (default)

```
CREATE TABLE foreign
(
    col ... REFERENCES primary (col)
            ON UPDATE NO ACTION -- not required
            ON DELETE NO ACTION -- not required
);
```

Ensuring Table Linkage
Cascade Change to Primary

```
BEGIN;
SELECT ...
FROM foreign
WHERE col = key
FOR UPDATE;
IF found
    UPDATE/DELETE foreign;
UPDATE/DELETE primary;
COMMIT;
```

Ensuring Table Linkage
REFERENCES Constraint
CASCADE

```
CREATE TABLE foreign
(
    col ... REFERENCES primary (col)
            ON UPDATE CASCADE
            ON DELETE CASCADE
);
```

Ensuring Table Linkage
Set Foreign to NULL on Change to Primary

```
BEGIN;
SELECT ...
FROM foreign
WHERE col = key
FOR UPDATE;
IF found
    UPDATE foreign SET col = NULL;
UPDATE/DELETE primary;
COMMIT;
```

Ensuring Table Linkage
REFERENCES Constraint
SET NULL

```
CREATE TABLE foreign
(
    col ... REFERENCES primary (col)
            ON UPDATE SET NULL
            ON DELETE SET NULL
);
```

Ensuring Table Linkage
Set Foreign to DEFAULT on Change to Primary

```
BEGIN;
SELECT ...
FROM foreign
WHERE col = key
FOR UPDATE;
IF found
    UPDATE foreign SET col = DEFAULT;
UPDATE/DELETE primary;
COMMIT;
```

Ensuring Table Linkage
REFERENCES Constraint
SET DEFAULT

```
CREATE TABLE foreign
(
    col ... REFERENCES primary (col)
            ON UPDATE SET DEFAULT
            ON DELETE SET DEFAULT
);

CREATE TABLE order (cust_id INTEGER REFERENCES customer (id));
```

Controlling Data

```
if col > 0 ...
   (col = 2 OR col = 7) ...
   length(col) < 10 ...
   INSERT/UPDATE tab;
```

CHECK Constraint

```
CREATE TABLE tab
(
    col ... CHECK (col > 0 ...
);

CREATE TABLE customer (age INTEGER CHECK (age >= 0));
```

Check Constraint Example

```
CREATE TABLE friend2 (
            firstname VARCHAR(15),
            lastname  VARCHAR(20),
            city      VARCHAR(15),
            state     CHAR(2)      CHECK (length(trim(state)) = 2),
            age       INTEGER      CHECK (age >= 0),
            gender    CHAR(1)      CHECK (gender IN ('M','F')),
            last_met  DATE         CHECK (last_met BETWEEN '1950-01-01'
                                          AND CURRENT_DATE),
            CHECK (upper(trim(firstname)) != 'ED' OR
                   upper(trim(lastname)) != 'RIVERS')
);

INSERT INTO friend2
VALUES ('Ed', 'Rivers', 'Wibbleville', 'J', -35, 'S', '1931-09-23');

ERROR:  ExecAppend: rejected due to CHECK constraint friend2_last_met
```

Default Column Values

```
if col not specified
   col = DEFAULT;
INSERT/UPDATE tab;
```

DEFAULT Constraint

```
CREATE TABLE tab
(
    quantity ... DEFAULT 1
);

CREATE TABLE customer (created timestamp DEFAULT CURRENT_TIMESTAMP);
```

Auto-numbering Column

```
CREATE TABLE counter (curr INTEGER);
INSERT INTO counter VALUES (1);
...
BEGIN;
val = SELECT curr FROM counter FOR UPDATE;
UPDATE counter SET curr = curr + 1;
COMMIT;
INSERT INTO tab VALUES (... val ...);
```

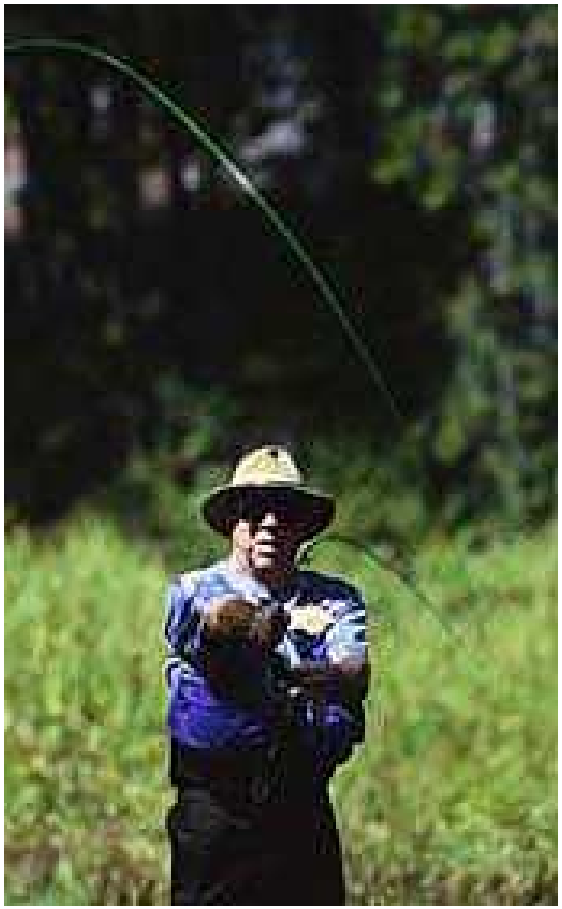
SERIAL/Sequence

```
CREATE TABLE tab
(
    col SERIAL
);

CREATE TABLE tab
(
    col INTEGER DEFAULT nextval('tab_col_seq')
);

CREATE TABLE customer (id SERIAL);

CREATE SEQUENCE customer_id_seq;
CREATE TABLE customer (id INTEGER DEFAULT nextval('customer_id_seq'));
```

Constraint Macros
DOMAIN

```
CREATE DOMAIN phone AS
    CHAR(12) CHECK (VALUE ~ '^[0-9]{3}-[0-9]{3}-[0-9]{4}$');
CREATE TABLE company ( ... phnum phone, ...);
```

Using SELECT's Features



ANSI Outer Joins - LEFT OUTER

```
SELECT *
FROM tab1, tab2
WHERE tab1.col = tab2.col
UNION
SELECT *
FROM tab1
WHERE col NOT IN
(
    SELECT tab2.col
    FROM tab2
);

SELECT *
FROM tab1 LEFT JOIN tab2 ON tab1.col = tab2.col;
```

ANSI Outer Joins - RIGHT OUTER

```
SELECT *
FROM tab1, tab2
WHERE tab1.col = tab2.col
UNION
SELECT *
FROM tab2
WHERE col NOT IN
(
    SELECT tab1.col
    FROM tab1
);

SELECT *
FROM tab1 RIGHT JOIN tab2 ON tab1.col = tab2.col;
```

ANSI Outer Joins - FULL OUTER

```
SELECT *
FROM tab1, tab2
WHERE tab1.col = tab2.col
UNION
SELECT *
FROM tab1
WHERE col NOT IN
(
    SELECT tab2.col
    FROM tab2
)
UNION
SELECT *
FROM tab2
WHERE col NOT IN
(
    SELECT tab1.col
    FROM tab1
);

SELECT *
FROM tab1 FULL JOIN tab2 ON tab1.col = tab2.col;
```

ANSI Outer Join Example

```
SELECT *
FROM customer LEFT JOIN order ON customer.id = order.cust_id;
```

Aggregates
SUM()

```
total = 0
FOREACH val IN set
    total = total + val;
END FOREACH
SELECT SUM(val) FROM tab;
```

Aggregates
MAX()

```
max = MIN_VAL;
FOREACH val IN set
    if (val > max)
        max = val;
END FOREACH

SELECT MAX(val) FROM tab;

SELECT MAX(cost) FROM part;
```

Aggregates
GROUP BY SUM()

```
qsort(set)

save = '';
total = 0;
FOREACH val IN set
    if val != save and save != ''
    {
        print save, total;
        save = val;
        total = 0;
    }
    total = total + amt;
END FOREACH
if save != ''
    print save, total;

SELECT val, SUM(amt) FROM tab GROUP BY val;
```

Aggregates
GROUP BY MAX()

```
save = '';
max = MIN_VAL;
FOREACH val IN set
    if val != save and save != ''
    {
        print save, max;
        save = val;
        max = MIN_VAL;
    }
    if (amt > max)
        max = amt;
END FOREACH
if save != ''
    print save, max;

SELECT val, MAX(amt) FROM tab GROUP BY val;
```

Aggregates
GROUP BY Examples

```
SELECT part, COUNT(*)
FROM order
ORDER BY part;

SELECT cust_id, SUM(due)
FROM order
GROUP BY cust_id
ORDER BY 2 DESC;
```

Merging SELECTs
UNION

```
SELECT * INTO TEMP out FROM ...
INSERT INTO TEMP out SELECT ...
INSERT INTO TEMP out SELECT ...
SELECT DISTINCT ...

SELECT *
UNION
SELECT *
UNION
SELECT *;
```

Joining SELECTs
INTERSECT

```
SELECT * INTO TEMP out;
DELETE FROM out WHERE out.* NOT IN (SELECT ...);
DELETE FROM out WHERE out.* NOT IN (SELECT ...);

SELECT *
INTERSECT
SELECT *
INTERSECT
SELECT *;
```

Subtracting SELECTs
EXCEPT

```
SELECT * INTO TEMP out;
DELETE FROM out WHERE out.* IN (SELECT ...);
DELETE FROM out WHERE out.* IN (SELECT ...);

SELECT *
EXCEPT
SELECT *
EXCEPT
SELECT *;
```

Controlling Rows Returned
LIMIT/OFFSET

```
DECLARE limdemo CURSOR FOR SELECT ...
FOR i = 1 to 5
    FETCH IN limdemo
END FOR

SELECT *
LIMIT 5;

DECLARE limdemo CURSOR FOR SELECT ...
MOVE 20 IN limdemo
FOR i = 1 to 5
    FETCH IN limdemo;
END FOR

SELECT *
OFFSET 20 LIMIT 5;
```

Controlling Rows Returned
LIMIT/OFFSET Example

```
SELECT order_id, balance
FROM order
ORDER BY balance DESC
LIMIT 10;
```

Locking SELECT Rows FOR UPDATE

```
BEGIN;
LOCK tab;
SELECT * FROM CUSTOMER WHERE id = 4452;
UPDATE customer SET balance = 0 WHERE id = 4452;
COMMIT;

BEGIN;
SELECT *
FROM customer
WHERE id = 4452
FOR UPDATE;
...
UPDATE customer
SET balance = 0
```

```
WHERE id = 4452;
COMMIT;
```

Temporary Tables

```
CREATE TABLE tab (...);
...
DROP TABLE tab;

CREATE TEMP TABLE tab (...);

SELECT *
INTO TEMPORARY hold
FROM tab1, tab2, tab3
WHERE ...
```

Automatically Modify SELECT VIEW - One Column

```
SELECT col4
FROM tab;

CREATE VIEW view1 AS
SELECT col4
FROM tab;

SELECT * FROM view1;
```

Automatically Modify SELECT VIEW - One Row

```
SELECT *
FROM tab
WHERE col = 'ISDN';

CREATE VIEW view2 AS
SELECT *
FROM tab
WHERE col = 'ISDN';

SELECT * FROM view2;
```

Automatically Modify SELECT VIEW - One Field

```
SELECT col4
FROM tab
WHERE col = 'ISDN';

CREATE VIEW view3 AS
SELECT col4
FROM tab
WHERE col = 'ISDN';

SELECT * FROM view3;
```

Automatically Modify INSERT/UPDATE/DELETE Rules

```
INSERT INTO tab1 VALUES (...);
INSERT INTO tab2 VALUES (...);

CREATE RULE insert_tab1 AS ON INSERT TO tab1 DO
INSERT INTO tab2 VALUES (...);

INSERT INTO tab1 VALUES (...);
```

Automatically Modify INSERT/UPDATE/DELETE Rules Example

```
CREATE TABLE service_request
(
    customer_id INTEGER,
    description text,
   cre_user text DEFAULT CURRENT_USER,
   cre_timestamp timestamp DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE service_request_log
(
    customer_id INTEGER,
    description text,
    mod_type char(1),
    mod_user text DEFAULT CURRENT_USER,
   mod_timestamp timestamp DEFAULT CURRENT_TIMESTAMP
);
```

Rules Example - Rule Definition

```
CREATE RULE service_request_update AS -- UPDATE rule
ON UPDATE TO service_request
DO
    INSERT INTO service_request_log (customer_id, description, mod_type)
    VALUES (old.customer_id, old.description, 'U');

CREATE RULE service_request_delete AS -- DELETE rule
ON DELETE TO service_request
DO
    INSERT INTO service_request_log (customer_id, description, mod_type)
    VALUES (old.customer_id, old.description, 'D');
```

Multi-User Consistency

- Atomic Changes
- Atomic Visibility
- Atomic Consistency
- Reliability

User 1	User 2	Descrip
BEGIN WORK		User 1 starts a t
UPDATE acct SET balance = balance - 100 WHERE acctno = 53224		remove 100 fro
UPDATE acct SET balance = balance + 100 WHERE acctno = 94913		add 100 to an a
SELECT * FROM acct		sees both chang
	SELECT * FROM acct	sees no changes
COMMIT WORK		
	SELECT * FROM acct	sees both chang

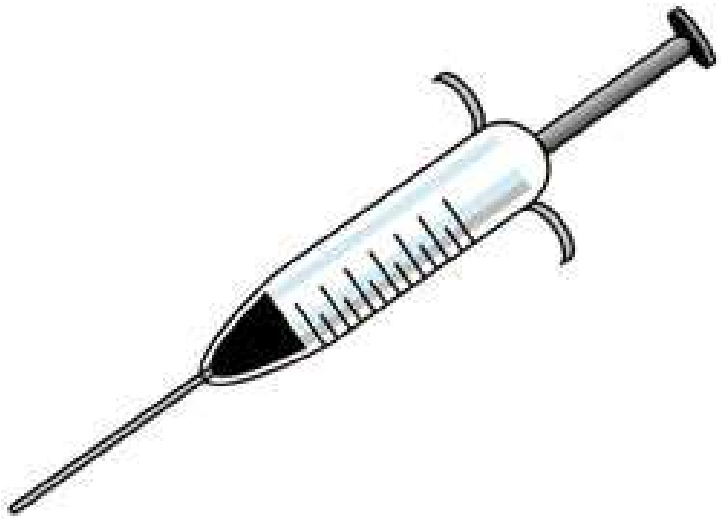
Notification
LISTEN/NOTIFY

```
signal()/kill()

LISTEN myevent;
NOTIFY myevent;
```

Application Walk-through

Gborg, http://gborg.postgresql.org/

Placing Code Into the Database: Server-side Functions



Single-Parameter Built-In Functions/Operator

```
SELECT factorial(10);
factorial
-----------
3628800
(1 row)

SELECT 10!;
?column?
----------
3628800
(1 row)
```

Two-Parameter Built-in Function/Operator

```
SELECT date_mi('2003-05-20'::date, '2001-10-13'::date);
date_mi
---------
584
(1 row)

SELECT '2003-05-20'::date - '2001-10-13'::date;
?column?
----------
584
(1 row)

psql \df
psql \do
```

Custom Server-Side Functions

- Create function
- Call function, manually or automatically

Compute Sales Tax

```
total = cost * 1.06;
INSERT ... VALUES ( ... total ... );

INSERT ... VALUES ( ... cost * 1.06, ... );

CREATE FUNCTION total(float)
RETURNS float
AS 'SELECT $1 * 1.06;'
LANGUAGE 'sql';

INSERT ... VALUES ( ... total(cost) ... )
```

Convert Fahrenheit to Centigrade

```
cent = (faren - 32.0) * 5.0 / 9.0
INSERT ... VALUES ( ... cent ... )

INSERT ... VALUES ( ... (faren - 32.0) * 5.0 / 9.0, ... )

CREATE FUNCTION ftoc(float)
RETURNS float
AS 'SELECT ($1 - 32.0) * 5.0 / 9.0;'
LANGUAGE 'sql';

INSERT ... VALUES ( ... ftoc(faren) ... )
```

Compute Shipping Cost

```
if cost < 2
    shipping = 3.00
else if cost < 4
    shipping = 5.00
else shipping = 6.00

INSERT ... VALUES ( ... cost + shipping ... );
```

Shipping Cost Function

```
CREATE FUNCTION shipping(numeric)
RETURNS numeric
AS 'SELECT CASE
        WHEN $1 < 2 THEN CAST(3.00 AS numeric(8,2))
        WHEN $1 >= 2 AND $1 < 4 THEN CAST(5.00 AS numeric(8,2))
        WHEN $1 >= 4 THEN CAST(6.00 AS numeric(8,2))
END;'
LANGUAGE 'sql';

INSERT ... VALUES ( ... cost + shipping(cost) ... );
```

String Processing — PL/pgSQL

```
CREATE FUNCTION spread(text)
RETURNS text
AS $$
DECLARE
    str text;
    ret text;
    i integer;
    len integer;
BEGIN
    str := upper($1);
    ret := ''; -- start with zero length
    i := 1;
    len := length(str);
    WHILE i <= len LOOP
        ret := ret || substr(str, i, 1) || ' ';
        i := i + 1;
    END LOOP;
    RETURN ret;
```

```
END;
$$
LANGUAGE 'plpgsql';

SELECT spread('Major Financial Report');
                    spread
-----------------------------------------------
 M A J O R   F I N A N C I A L   R E P O R T
(1 row)
```

State Name Lookup
SQL Language Function

```
SELECT name
FROM statename
WHERE code = 'AL';

CREATE FUNCTION getstatename(text)
RETURNS text
AS 'SELECT name
    FROM statename
    WHERE code = $1;'
LANGUAGE 'sql';

SELECT getstatename('AL');
```

State Name Lookup From String PL/pgSQL Language Function

```
CREATE FUNCTION getstatecode(text)
RETURNS text
AS $$
DECLARE
    state_str statename.name%TYPE;
    statename_rec record;
    i integer;
    len integer;
    matches record;
    search_str text;
BEGIN
     state_str := initcap($1); -- capitalization match column
     len := length(trim($1));
     i := 2;
     SELECT INTO statename_rec * -- first try for an exact match
     FROM statename
     WHERE name = state_str;
     IF FOUND
     THEN RETURN statename_rec.code;
     END IF;
```

```
    WHILE i <= len LOOP -- test 2,4,6,... chars for match
        search_str = trim(substr(state_str, 1, i)) || '%';
        SELECT INTO matches COUNT(*)
        FROM statename
        WHERE name LIKE search_str;
        IF matches.count = 0 -- no matches, failure
        THEN RETURN NULL;
        END IF;
        IF matches.count = 1 -- exactly one match, return it
        THEN
            SELECT INTO statename_rec *
            FROM statename
            WHERE name LIKE search_str;
            IF FOUND
            THEN RETURN statename_rec.code;
            END IF;
        END IF;
        i := i + 2; -- >1 match, try 2 more chars
    END LOOP;
    RETURN '';
END;
$$
LANGUAGE 'plpgsql';

SELECT getstatecode('Alabama');
SELECT getstatecode('ALAB');
```

```
SELECT getstatecode('Al');
SELECT getstatecode('Ail');
```

State Name Maintenance

```
CREATE FUNCTION change_statename(char(2), char(30))
RETURNS boolean
AS $$
DECLARE
     state_code ALIAS FOR $1;
     state_name ALIAS FOR $2;
     statename_rec RECORD;
BEGIN
     IF length(state_code) = 0 -- no state code, failure
     THEN RETURN 'f';
     ELSE
         IF length(state_name) != 0 -- is INSERT or UPDATE?
         THEN
             SELECT INTO statename_rec *
             FROM statename
             WHERE code = state_code;
             IF NOT FOUND -- is state not in table?
             THEN    INSERT INTO statename
                     VALUES (state_code, state_name);
             ELSE    UPDATE statename
                     SET name = state_name
```

```
                    WHERE code = state_code;
            END IF;
            RETURN 't';
        ELSE -- is DELETE
            SELECT INTO statename_rec *
            FROM statename
            WHERE code = state_code;
            IF FOUND
            THEN    DELETE FROM statename
                    WHERE code = state_code;
                    RETURN 't';
            ELSE RETURN 'f';
            END IF;
        END IF;
    END IF;
END;
$$
LANGUAGE 'plpgsql';

SELECT change_statename('AL','Alabama');
SELECT change_statename('AL','Bermuda');
SELECT change_statename('AL','');
SELECT change_statename('AL',''); -- row was already deleted
```

SELECT Inside FROM

```
SELECT *
FROM (SELECT * FROM tab) AS tab;

SELECT *
FROM (  SELECT 1,2,3,4,5 UNION
        SELECT 6,7,8,9,10 UNION
       SELECT 11,12,13,14,15) AS tab15;
col| col| col| col| col
---+----+----+----+----
 1 |  2 |  3 |  4 |  5
 6 |  7 |  8 |  9 | 10
11 | 12 | 13 | 14 | 15
(3 rows)
```

Function Returning Multiple Values

```
CREATE TABLE int5(x1 INTEGER, x2 INTEGER, x3 INTEGER, x4 INTE-
GER, x5 INTEGER);
CREATE FUNCTION func5() RETURNS SETOF int5 AS
'SELECT 1,2,3,4,5;'
LANGUAGE SQL;

SELECT * FROM func5();
 x1 | x2 | x3 | x4 | x5
----+----+----+----+----
  1 |  2 |  3 |  4 |  5
(1 row)
```

Function Returning a Table Result

```
CREATE OR REPLACE FUNCTION func15() RETURNS SETOF int5 AS
'   SELECT 1,2,3,4,5 UNION
    SELECT 6,7,8,9,10 UNION
    SELECT 11,12,13,14,15;'
LANGUAGE SQL;

SELECT * FROM func15();
 x1 | x2 | x3 | x4 | x5
----+----+----+----+----
  1 |  2 |  3 |  4 |  5
  6 |  7 |  8 |  9 | 10
 11 | 12 | 13 | 14 | 15
(3 rows)
```

Automatic Function Calls
Trigger

- `BEFORE/AFTER ROW`
- `INSERT/UPDATE/DELETE`
- `OLD/NEW`

Trigger on Statename

```
CREATE FUNCTION trigger_insert_update_statename()
RETURNS trigger
AS $$
BEGIN
    IF new.code !~ '^[A-Za-z][A-Za-z]$'
    THEN RAISE EXCEPTION 'State code must be two alphabetic characters.';
    END IF;
    IF new.name !~ '^[A-Za-z ]*$'
    THEN RAISE EXCEPTION 'State name must be only alphabetic characters.';
    END IF;
    IF length(trim(new.name)) < 3
    THEN RAISE EXCEPTION 'State name must longer than two characters.';
    END IF;
    new.code = upper(new.code); -- uppercase statename.code
    new.name = initcap(new.name); -- capitalize statename.name
    RETURN new;
END;
$$
```

```
LANGUAGE 'plpgsql';
```

Install Trigger On Statename

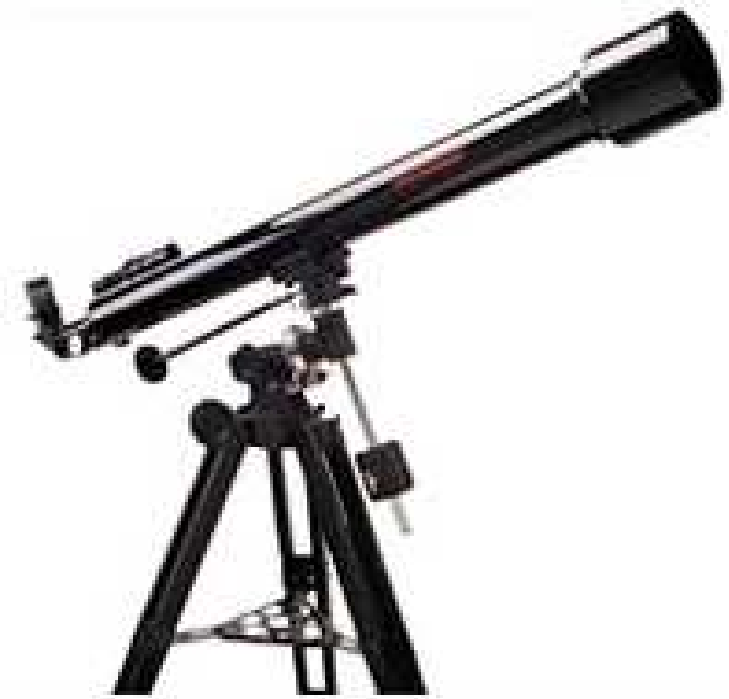
```
CREATE TRIGGER trigger_statename
BEFORE INSERT OR UPDATE
ON statename
FOR EACH ROW
EXECUTE PROCEDURE trigger_insert_update_statename();

INSERT INTO statename VALUES ('a', 'alabama');
INSERT INTO statename VALUES ('al', 'alabama2');
INSERT INTO statename VALUES ('al', 'al');
INSERT INTO statename VALUES ('al', 'alabama');
```

Function Languages

- SQL
- PL/pgSQL
- PL/TCL
- PL/Python
- PL/Perl
- PL/sh
- C

Function Examples

- /contrib/earthdistance
- /contrib/fuzzystringmatch
- /contrib/pgcrypto



Adding New Data and Indexing Features

Creation

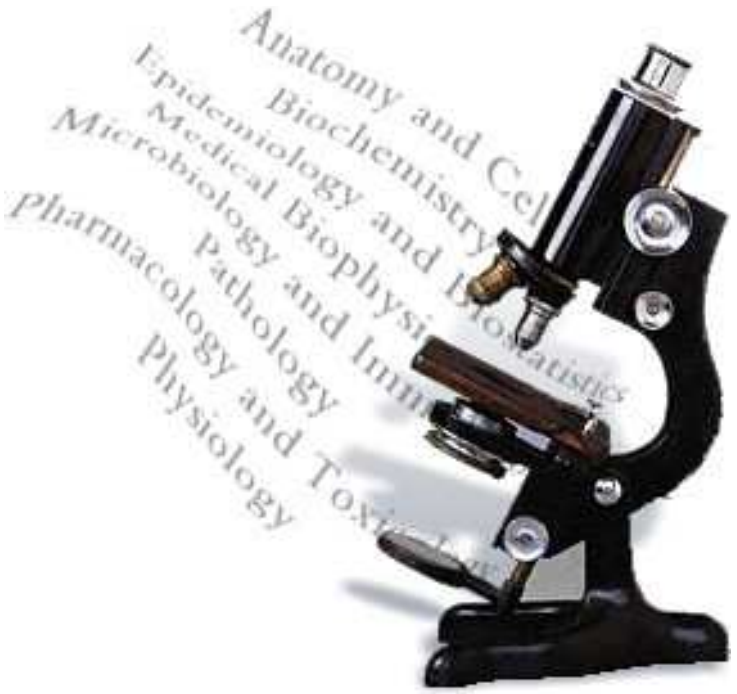
- CREATE FUNCTIONS in C
- **CREATE TYPE**
- **CREATE OPERATOR**
- **CREATE OPERATOR CLASS (index type)**

Create New Data Type
With Operator and Index Support

- Write input/output functions
- Register input/output functions with CREATE FUNCTION
- Register type with CREATE TYPE
- Write comparison functions
- Register comparison functions with CREATE FUNCTION
- Register comparison functions with CREATE OPERATOR
- Register operator class for indexes with CREATE OPERATOR CLASS

Create New Data Type Examples

- /contrib/chkpass
- /contrib/isn
- /contrib/cube
- /contrib/ltree
- /src/backend/utils/adt



Conclusion