A step towards SQL/MED DATALINK

Author: Gilles Darold

PgConf Asia September 2019

LzLabs | software defined mainframe®

SQL/MED

- SQL/MED => SQL Management of External Data
- Define methods to access non-SQL data in SQL
- Standardized by ISO/IEC 9075-9:2001 (completed in SQL:2003 and SQL:2008)

SQL/MED specification is divided in two parts

Part 1 : Foreign Data Wrapper

- Access other data sources represented as SQL tables in PostgreSQL
- Fully implemented in PostgreSQL
- List of FDW : https://wiki.postgresql.org/wiki/Foreign_data_wrappers

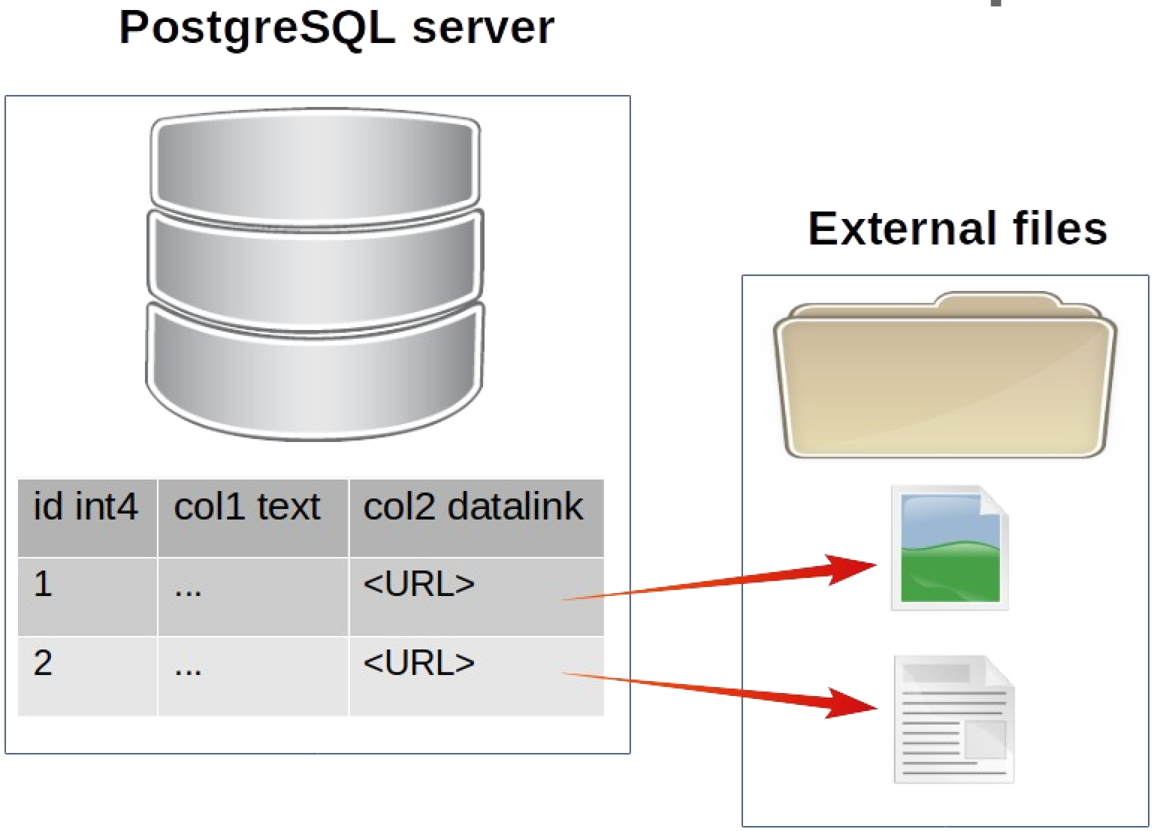
Part 2 : Datalinks

- Reference file that is not part of the SQL environment
- The file is assumed to be managed by some external file manager
- Column values are references to local or remote files
- There is no PostgreSQL implementation
- A plpgsql/plperlu prototype at https://github.com/lacanoid/datalink

What are Datalink exactly ?

- Files are referenced through a new DATALINK type.
- A special SQL type intended to store :
  - URL in database referencing a external file
  - Access control and behavior over external files
- File contents are not stored in the database but on file system

Datalink representation



id int4	col1 text	col2 datalink
1	...	<URL>
2	...	<URL>

Why are Datalink useful?

Applications that need:

- the power of SQL with ACID properties
- and working directly with files.

Support huge files, ex: video or audio streaming, without having to be stored in ByteA or Large Object columns.

Avoid filling Share Buffers, WALs and database with files that can be accessed directly by other programs, ex: images or PDF files through a Web server.

Benefit of external file caching, advanced file content search and indexing.

[…]

Purpose of Datalinks

Provide a mechanism to implement the

- referential integrity
- recovery
- and access control

of the files and the SQL-data associated with them.

Files can be modified using file system commands or via SQL

These mechanisms are collectively called the **Datalinker**.

Datalink implementations

- PostgreSQL has no implementation, but a wish list:
  - https://wiki.postgresql.org/wiki/DATALINK
- Oracle has external data support through DIRECTORY and BFILE.
- SQL Server no implementation, use varbinary => bytea.
- MySQL no implementation BINARY/VARBINARY => bytea.
- DB2 is the only DBMS that fully implements Datalink

Why so few implementation

- Technology from the past?
  - We have learned to work without DATALINK
  - Even DB2 have deprecated DATALINK?
- ACID properties ?
  - Complex to guarantee
  - Especially with filesystem access
- Then why a Datalink extension for PostgreSQL?
  - Build a proof of concept to test things
  - Look for a faster alternative for Bytea / Large Object
  - Add ACID properties to external file storage

Oracle implementation

Oracle implementation

Composed of two SQL objects:

- DIRECTORY
- BFILE

Oracle implementation - Directory

DIRECTORY : base object pointing to a operating system directory on the database server for reading and writing external files.

```
SQL> CREATE DIRECTORY testdir AS '/var/www/images';
```

Manage privilege to read/write to this directory :

```
SQL> GRANT read, write ON DIRECTORY testdir TO username;
```

Oracle implementation - BFILE

BFILE : data type used to store a locator (link) to an external file (stored outside of the database).

BFILEs are read-only and cannot be replicated to another system.

```
SQL> CREATE TABLE test_bfile (id NUMBER, file BFILE);
SQL> CREATE DIRECTORY testdir AS '/var/www/images';
SQL> INSERT INTO test_bfile VALUES (1, BFILENAME('TESTDIR', 'logo.png'));
```

The files must exist and be written using external application or created by an Oracle stored procedure.

BFILE for PostgreSQL

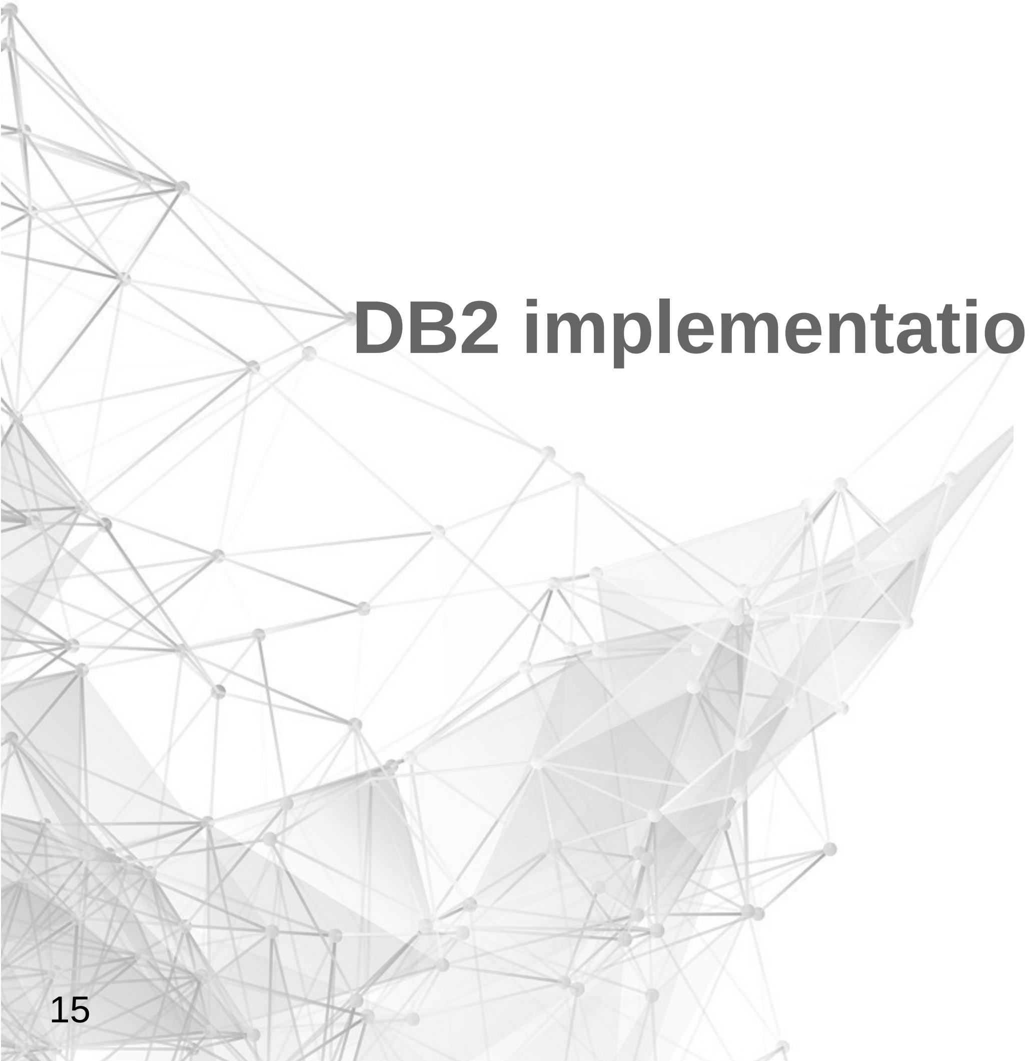
There is a PostgreSQL extension emulating BFILE: *external_file*.

More information and example here:

- https://github.com/darold/external_file

Author: Dominique Legendre

Adds the same functionalities given by the Oracle's BFILE data type



DB2 implementation

DB2, the “defacto” standard

Only SGBD implementing SQL/MED DATALINK

DB2 Datalink implementation has two components:

- Data Links engine
  - resides on the DB server and part of the DB2 engine code
  - responsible for processing SQL requests involving DATALINK columns
- Data Links Manager

DB2, Datalink manager

The Data Links Manager consists of two components:

- Data Links File Manager (DLFM)
- Data Links File System Filter (DLFF)

DLFM applies referential integrity, access control, backup and recovery as specified in the DATALINK column definition.

DLFF is a database/filesystem layer that intercepts file system calls (open, rename, unlink) issued by the application. This ensures that any access request meets DBMS security and integrity requirements.

DB2 DATALINK type

The DATALINK values encode:

- the name of a Data Links server containing the file
- the filename in terms of a Uniform Resource Locator (URL)

The DATALINK value is robust in terms of integrity, access control, and recovery.

The only Data Links server names that you can specify in a DATALINK value are those that have been registered to a DB2 database.

Back to standard SQL

DATALINK: Example of use

```
CREATE TABLE persons (
        id integer,
        fullname text,
        picture DATALINK
);

INSERT INTO persons VALUES (1, 'Jon Doe', DLVALUE('file:///dir1/img1.jpg'));
```

The standard specifies that the referenced URL can only be of the format of

- FILE (file://) for local access
- URL (http://) for remote access

With the “http” scheme the Datalinker can not do anything except storing URLs. All other schemes are not supported.

DATALINK options

SQL/MED allows options to be specified for DATALINK columns.

Used to determined how strictly the SQL-server controls the file.

The possibilities range from :

- no control at all (the file does not even have to exist)
- to full control, (ex: removal of file linked by a datalink value)

```
CREATE TABLE images (id integer, filename text,
             picture datalink FILE LINK CONTROL INTEGRITY ALL);
```

DATALINK options 1/3

Link and Integrity Control

NO LINK CONTROL : no validation of the reference to existing file/URL (default).

FILE LINK CONTROL : Datalink value must reference an existing file/URL.

INTEGRITY ALL | SELECTIVE | NONE

- ALL : referenced files can only be renamed or deleted through SQL.
- SELECTIVE : referenced files can be renamed or deleted through SQL or through the file system.
- NONE : no integrity, implied for NO LINK CONTROL

DATALINK options 2/3

Unlinking and Recovery Behavior

ON UNLINK DELETE | RESTORE | NONE :

- DELETE : file is deleted from file system when deleted from database.
- RESTORE : file’s original permissions are restored when deleted from database.
- NONE : no change in file permissions when file reference is deleted from database.

RECOVERY YES / NO: applies recovery abilities (PITR) to referenced files.

DATALINK options 3/3

Access Permissions

READ PERMISSION FS | DB :

- FS : file system controls file read permission.
- DB : Database system controls file read permission.

WRITE PERMISSION FS | ADMIN | BLOCKED

- FS : File system controls file write permission.
- ADMIN : Writes to the file are managed by the database system.
- BLOCKED : modifying file “in place” is not authorized.

DATALINK type ISO Specification

The DataLink type can not be cast in an other data type.

If X or Y are a datalink, then comparison operator shall be either equals operator or not equals operator.

DATALINK X is equal to DATALINK Y if and only if :

- DLCOMMENT(X) = DLCOMMENT(Y)
- And DLLINKTYPE(X) = DLLINKTYPE(Y)
- And DLURLPATHONLY(X) = DLURLPATHONLY(Y)
- And DLURLSCHEME(X) = DLURLSCHEME(Y)
- And DLURLSERVER(X) = DLURLSERVER(Y).

Functions to extract information

- DLCOMMENT(datalink) → text
- DLLINKTYPE(datalink) → text enum(‘FILE’, ‘URL’)
- DLURLSCHEME(datalink) → text
- DLURLSERVER(datalink) → text

- DLURLCOMPLETE(datalink) → uri with a file access token
- DLURLCOMPLETEONLY(datalink) → uri

- DLURLPATH(datalink) → text with a file access token
- DLURLPATHONLY(datalink) → text

DATALINK restriction

- Default value is NULL
- DATALINK can not appears in:
  - < comparison predicate >
  - < general set function >
  - < group by clause >
  - < order by clause >
  - < unique constraint definition >
  - < referential constraint definition >
  - < DISTINCT clause >
  - <select list> of an operand of UNION, INTERSECT, and EXCEPT.
  - Columns used for matching when forming a <joined table>.

DATALINK reserved keywords

DATALINK adds some more reserved keywords:

```
| BLOCKED | CONTROL | DB | FILE | FS | INTEGRITY | LINK
| PERMISSION | RECOVERY | REQUIRING | RESTORE
| SELECTIVE | TOKEN | UNLINK | YES

| DATALINK | DLNEWCOPY | DLPREVIOUSCOPY
| DLURLCOMPLETE | DLURLCOMPLETEWRITE
| DLURLCOMPLETEONLY | DLURLPATH
| DLURLPATHWRITE | DLURLPATHONLY | DLURLSCHEME
| DLURLSERVER | DLVALUE
```

Consult DATALINK values

Like any other SQL data type:

```
SELECT picture FROM persons WHERE id = 1
```

returns a Datalink value expression, for DB2 it is:

```
FILE        file:///dir1/img1.jpg
```

The Datalink represented by the link type and the URL of the external file.

Modify DATALINK values

- For INSERT :
  - DLVALUE(url[,link_type][,comment]) → datalink
- For UPDATE :
  - DLNEWCOPY(url,token) → datalink
  - DLPREVIOUSCOPY(url,token) → datalink
  - DLVALUE(url[,link_type][,comment]) → datalink
  - DLREPLACECONTENT(url-target, url-source, comment) → datalink

A Datalink implementation for PostgreSQL

DATALINKER mechanism

Formally represented by a DATALINK extension composed of:

- A URI extension to normalize, verify and access URL parts.
- The uuid-ossp extension to generate token.
- A library of C functions for access to external files.
- A SQL extension definition file implementing the Datalink type and the SQL functions.
- A background worker acting like “autovacuum” for Datalink.

URI Base data type

Access to local or remote file, need to add an URI type

URI PostgreSQL extension :

- https://github.com/darold/uri
- Store any kind of well formed URL
- Based on uriparser and liburi to validate URI
- Use libcurl to verify remote URL and get header information
- Support indexing and contains operator (@>)

Not only a basic type for URL, there is advanced features and constraints.

Not part of the Datalink extension to be used and evolved independently.

Using URI data type

There is some constraints in using this type:

- The Uri must be well formed, throw an error otherwise.
- URIs are normalized (canonic representation) according to section 6.2.2 of RFC3986, including:
  - Adjusting percent-encoded characters.
  - Removal of redundant components from the path
    - *“/a/b/c/../d/../../e”* will be normalized to *“/a/e”*
- Turn scheme and host part into lowercase.
- Do not expect saving the exact same input provided by the user or application.

Manipulate URI 1/4

Per RFC 3986 extracting URI parts from a URI column.

- **uri_get_scheme(uri)** returns protocol part of uri as text
- **uri_get_auth(uri)** returns user part of uri as text
- **uri_get_host(uri)** returns host part of uri as text
- **uri_get_port(uri)** returns port part of uri as text
- **uri_get_portnum(uri)** returns port part uri as integer
- **uri_get_path(uri)** returns path part of uri as text
- **uri_get_query(uri)** returns query part of uri as text
- **uri_get_fragment(uri)** returns fragment part of uri as text

Manipulate URI 2/4

More function to manipulate URI data.

- **uri_is_absolute(uri)**, **uri_is_absolute_path(uri)** returns true if uri is absolute or if uri path is absolute
- **uri_localpath_exists(uri)**, **uri_remotepath_exists(uri)** returns true if uri exists as a regular local path (not symlink) or as a remote url.
- **uri_path_exists(uri)** returns true if uri exists as a local regular path (not symlink) or remote url (local/remote is autodetected).
- **uri_path_content_type(uri)** returns the content_type of the url (autodetect local/remote) using libmagic/file and HTTP content-type.
- **uri_path_size(uri)** returns the size of a local path (not symlink) or remote url (autodetect local/remote).
- **uri_escape(text)**, **uri_unescape(text)** returns the encoded or decoded URL of the given string.

Manipulate URI 3/4

A function widely used in the Datalink extension to secure paths:

- **uri_rebase_url(uri, uri)** returns an uri of a path rebased on an uri base

Examples:

```
test=# SELECT uri_rebase_url('/pgcluu_logo.png', 'http://pgcluu.darold.net/');
              uri_rebase_url
------------------------------------------
 http://pgcluu.darold.net/pgcluu_logo.png

test=# SELECT uri_rebase_url('video/vid1.mp4', 'file:///base_directory1/');
           uri_rebase_url
-------------------------------------
 file:///base_directory1/video/vid1.mp4
```

Manipulate URI 4/4

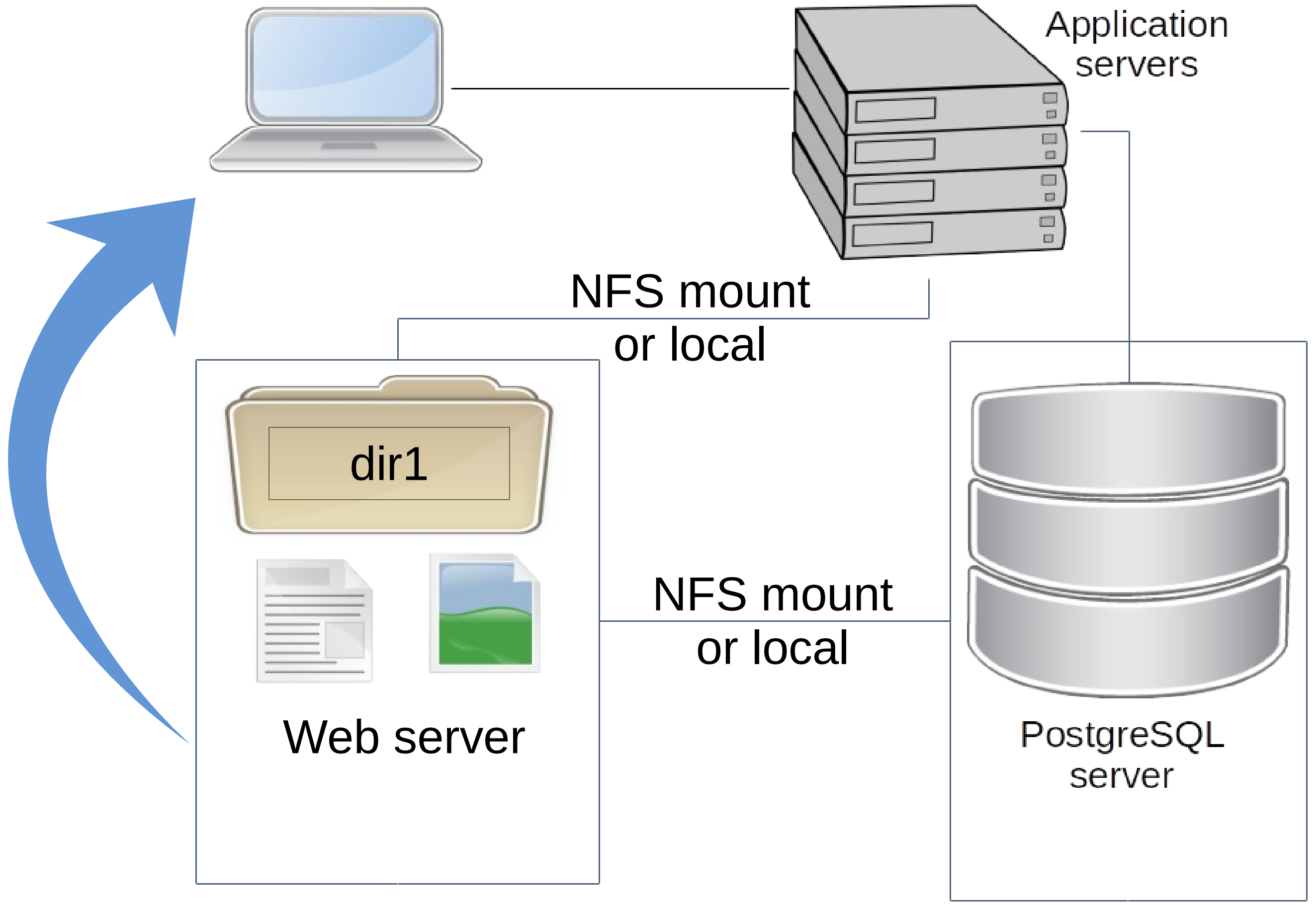
An other function widely used in the Datalink extention:

- **uri_get_relative_path(uri, uri)** returns a path relative to its base

```
test=# SELECT
        uri_get_relative_path('file:///data/pg_dl/dir1/images/logo.png',
                                          'file:///data/pg_dl/dir1/');
    uri_get_relative_path
  -----------------------------
     images/logo.png
```

Used to store only the smaller significant part of the path to save space.

Architecture of the PostgreSQL DATALINK extension


Application servers
NFS mount or local
dir1
Web server
NFS mount or local
PostgreSQL server

DATALINK data type

The Datalink value expression is represented by:

{baseid, url, comment, token, previous_token, unlink_privileges}

Defined as a composite data type in this POC:

```
CREATE TYPE datalink AS
(
    dl_base integer,        -- Id of the base directory
    dl_path uri,            -- Url of the external file
    dl_comment text,        -- A comment
    dl_token uuid,          -- Current active token
    dl_prev_token uuid,     -- Previous active token
    ~~dl_unlink_privileges  -- Privileges to be restored on unlink~~
);
```

DATALINK bases

- DB2 has PREFIX (the mount point) stored in table DFM_PRFX.
- Oracle has DIRECTORY stored in table ALL_DIRECTORIES.
- The DATALINK extension has Bases Directories:

```
CREATE TABLE pg_datalink_bases
(
        dirid serial ,                          -- a unique number as reference
        dirname text PRIMARY KEY ,   -- the unique name of the directory
        base uri NOT NULL                   -- the URI of the base directory
);
```

There is two entries in this table representing the default bases.

DATALINK default bases

- FILE : used to store any local path not associated to a dedicated base.
- URL : used to store any remote URL not associated to a dedicated base.

```
test=# SELECT * FROM pg_datalink_bases WHERE dirid <= 0;
 dirid | dirname |                base
-------+---------+---------------------------------------
    -1 | FILE    | file:///var/lib/datalink/pg_external_files
     0 | URL     | http://
```

Default FILE base directory at runtime: /var/lib/datalink/pg_external_files/
GUC: *datalink.dl_base_directory*

The URL base has no host part, any URL can be stored with this base.

DATALINK adding bases

Create a new base directory, only superuser can do that:

```
    INSERT INTO pg_datalink_bases (dirname, base)
                 VALUES (‘MyBase1’, ‘file:///var/www/mysite1/’);
```

- No verification if the path exists.
- postgres user must have read/write permission on the directory if files will be managed by SQL.
- Sub-directories can be used on new bases.

DATALINK duplicate bases

- Duplicate base allowed:

```
INSERT INTO pg_datalink_bases (dirname, base, recovery)
        VALUES (‘file_with_backup’, ‘file://path/to/dir2/’, ‘t’);

INSERT INTO pg_datalink_bases (dirname, base, recovery)
        VALUES (‘file_without_backup’, ‘file://path/to/dir2/’, ‘f’);
```

Allow different behaviors for external files in the same directory.

This is an extension to the standard where the same behavior is used for all Datalink in the same column.

DATALINK options/attributes

Columns with Datalink type have special attributes:

```
CREATE TABLE person (
        fullname varchar(128) NOT NULL ,
        picture datalink LINKTYPE URL FILE LINK CONTROL INTEGRITY ALL
        READ PERMISSION FS WRITE PERMISSION BLOCKED
        RECOVERY YES ON UNLINK RESTORE
);
```

No hook on the parser: not possible to add this syntax from an extension.

These attributes definition are reported into the *pg_datalink_bases* table which allow to specify these options for a given directory base.

A column can have multiple attributes definitions <> ISO SQL/MED

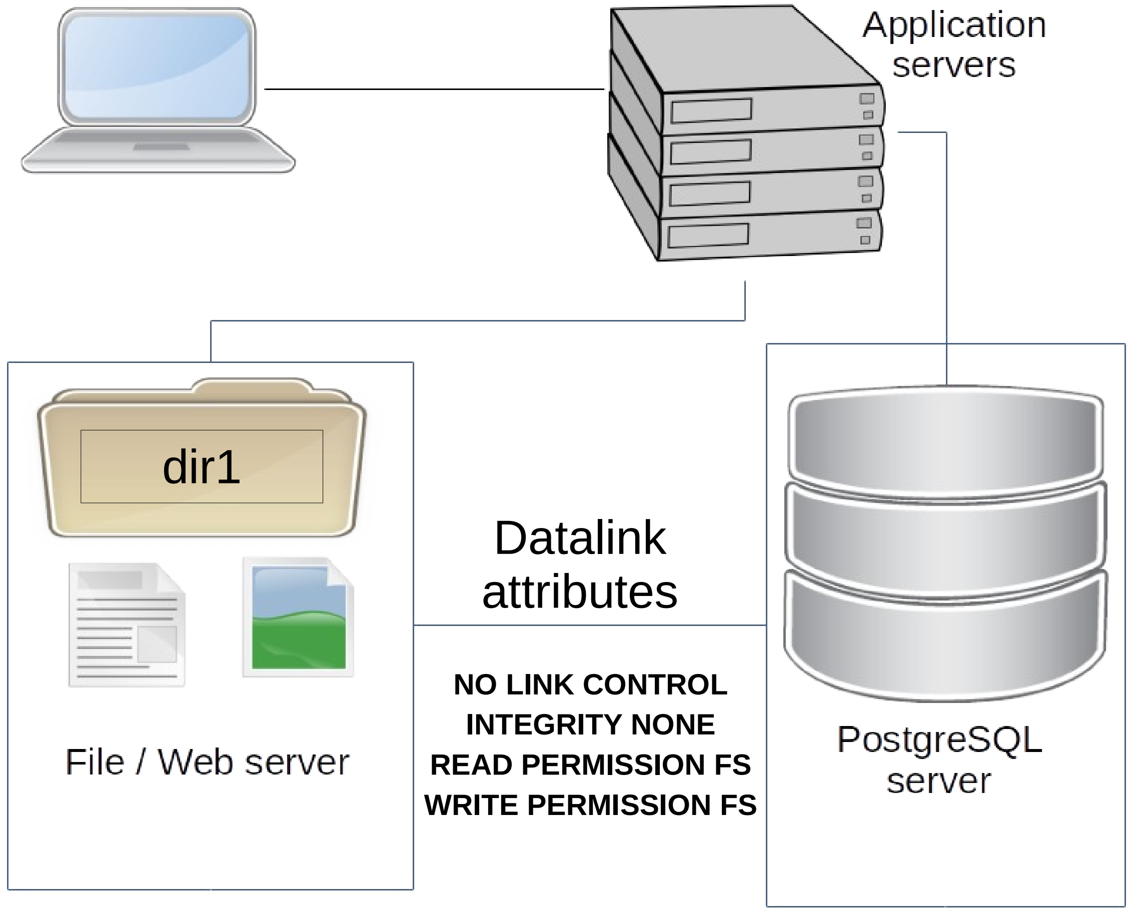
pg_datalink_bases attributes

```
CREATE TABLE pg_datalink_bases (
        dirid serial ,
        dirname text PRIMARY KEY ,
        base uri NOT NULL,
        linkcontrol boolean DEFAULT false,
        integrity boolean DEFAULT false,
        readperm boolean DEFAULT false,
        writeperm boolean DEFAULT false,
        writeblocked boolean DEFAULT true,
        writetoken boolean DEFAULT true,
        recovery boolean DEFAULT false,
        onunlink text DEFAULT 'NONE'
                        CHECK (onunlink IN ('NONE', 'RESTORE', 'DELETE')) );
```

The correct use and association of all attributes is verified by trigger.

DATALINK extension behaviors following attributes definitions

Default Datalink attributes



- Missing external file ok
- Delete/rename files FS
- Read permission given by the FS
- Write permission given by the FS

FS = File System

NO LINK CONTROL

The file referenced by the URL may not existing.

This option implies that :

- the integrity control option is NONE,
- the read permission option is FS,
- the write permission option is FS,
- the recovery option is NO,
- the unlink option is NONE,

Specify different values for these options is not permitted. The correct use and association of all attributes is verified by trigger.

Probably shall be better to use a simple URI data type instead?

READ / WRITE PERMISSION FS

The file system gives authorization to the application to read/write files.

PostgreSQL access to external files => *postgres* superuser only.

The file system doesn’t know the connected DB user.

In this first version of the Datalink extension FS managed privileges are not supported. All access to files are done through the *postgres* user.

Require to implement a layer responsible of checking the authorization on DB or FS sides and intercepting the open/read/write/rename/unlink call to files.

READ / WRITE PERMISSION DB

External files on FS must have the R/W permission for the *postgres* user.

Access by other users, for example the *www-data* used by a Web Server, can be controlled by using group privilege.

The base directory must be created like that:

```
cd /var/lib/datalink/pg_external_files/
mkdir dir1/
chown postgres.www-data dir1/
chmod u=rwX,g=rwsX,o= dir1/
```

INTEGRITY versus R/W PERMISSION

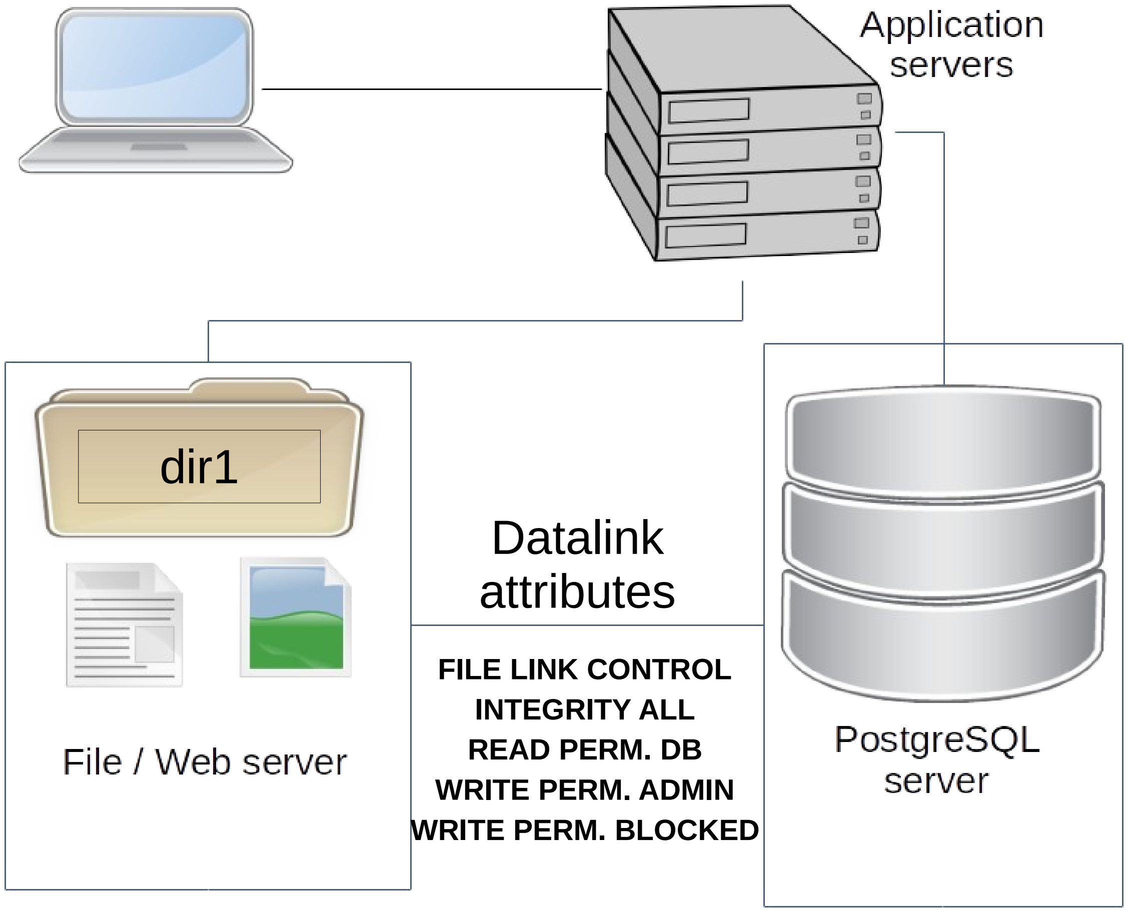
INTEGRITY SELECTIVE:

- the read permission option is FS,
- the write permission option is FS,

INTEGRITY ALL (DB):

- the read permission option is DB or FS,
- the write permission option is DB (ADMIN) or FS,
- access to files require or not a TOKEN FOR UPDATE,
- direct modification to files is forced to BLOCKED

Full control by the Datalink extension



- External files must exists
- Delete/rename files SQL only
- Read permission by SQL
- Write permission by SQL

WRITE PERMISSION BLOCKED

Direct write access to files referenced by Datalink is not available.

Modifications must be done this way:

- Copy the file,
- Modify the copy,
- Update the Datalink to reference the copy,

With WRITE PERMISSION ADMIN attribute is forced to BLOCKED.

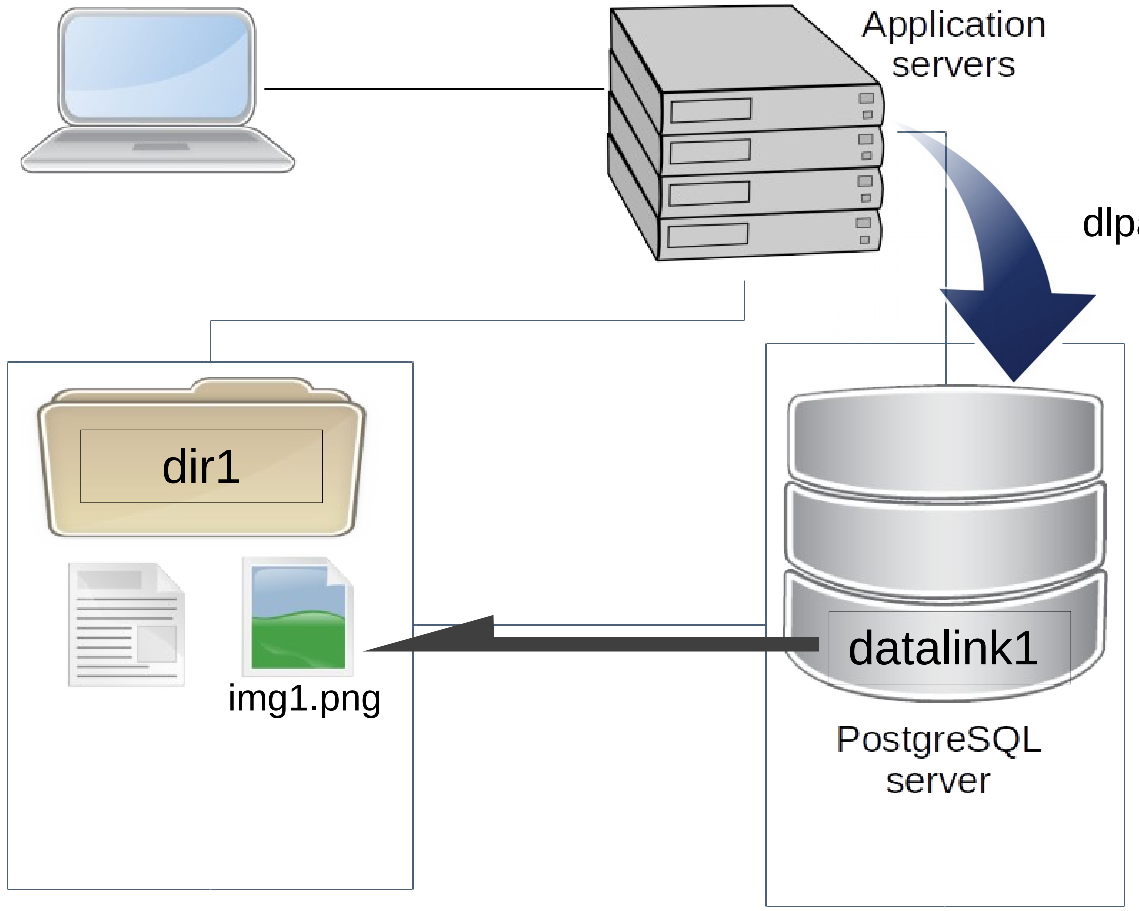
CONCURRENCY and PITR

You definitively need attributes:

- INTEGRITY ALL
- WRITE PERMISSION ADMIN REQUIRING TOKEN FOR UPDATE
- RECOVERY YES.

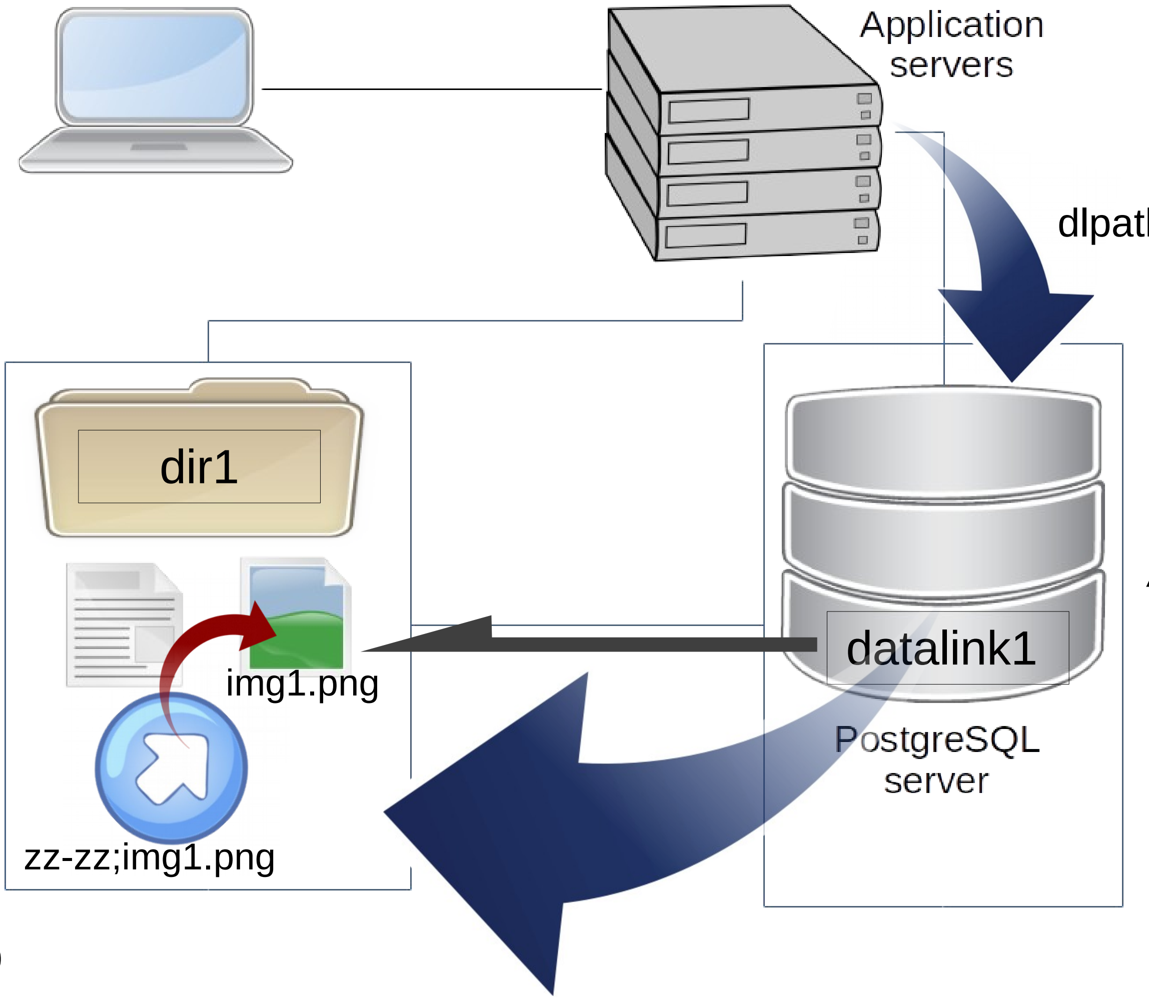
READING with concurrency

INTEGRITY ALL



1) The application ask to the DB a path with a token to read the file referenced in datalink1 record

INTEGRITY ALL



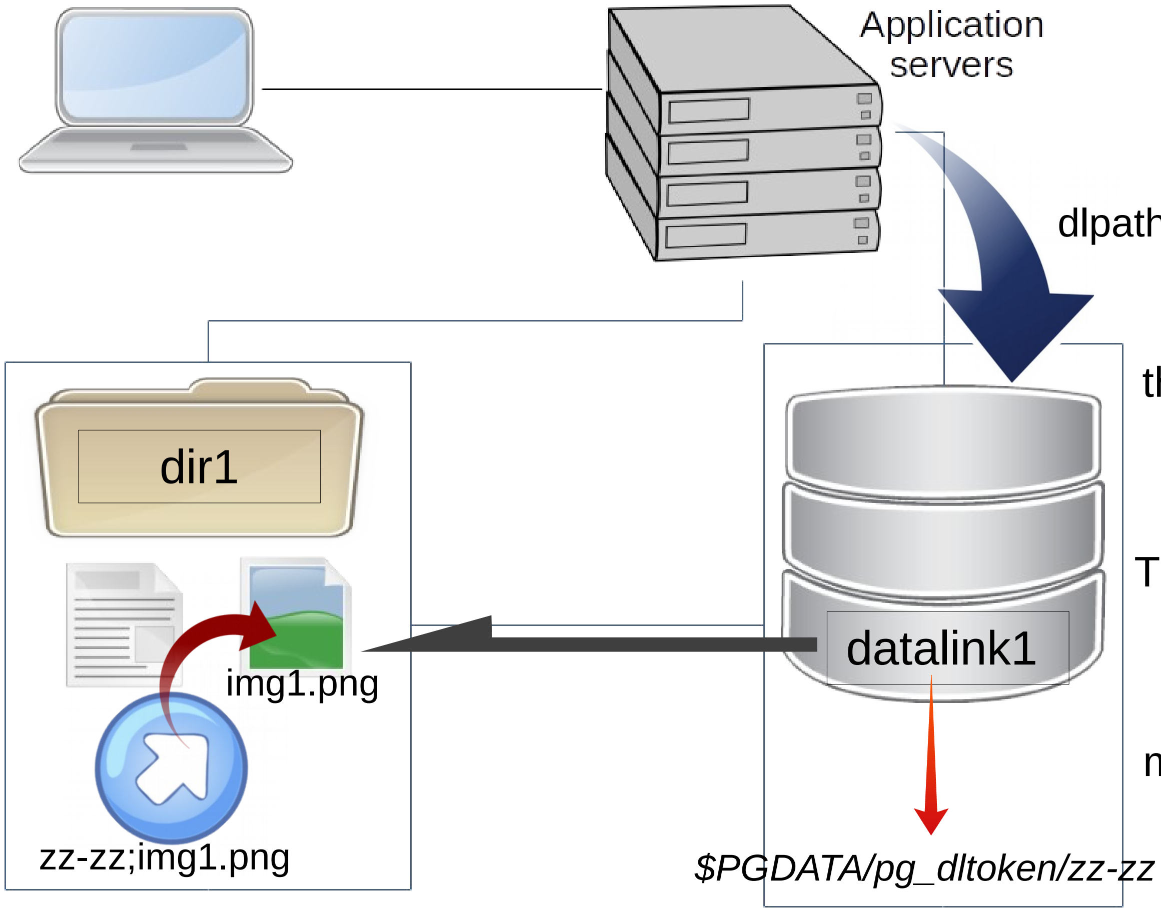
2.1) The DLPATHCOMPLETE() function create a symlink to the current version of the file with a new token in the link name:

/dir1/zz-zz;img1.png → /dir1/img1.png

The token is generated using uuid_generate_v4().

The token prefixes the filename with a semicolon as separator.

TOKEN EXPIRATION



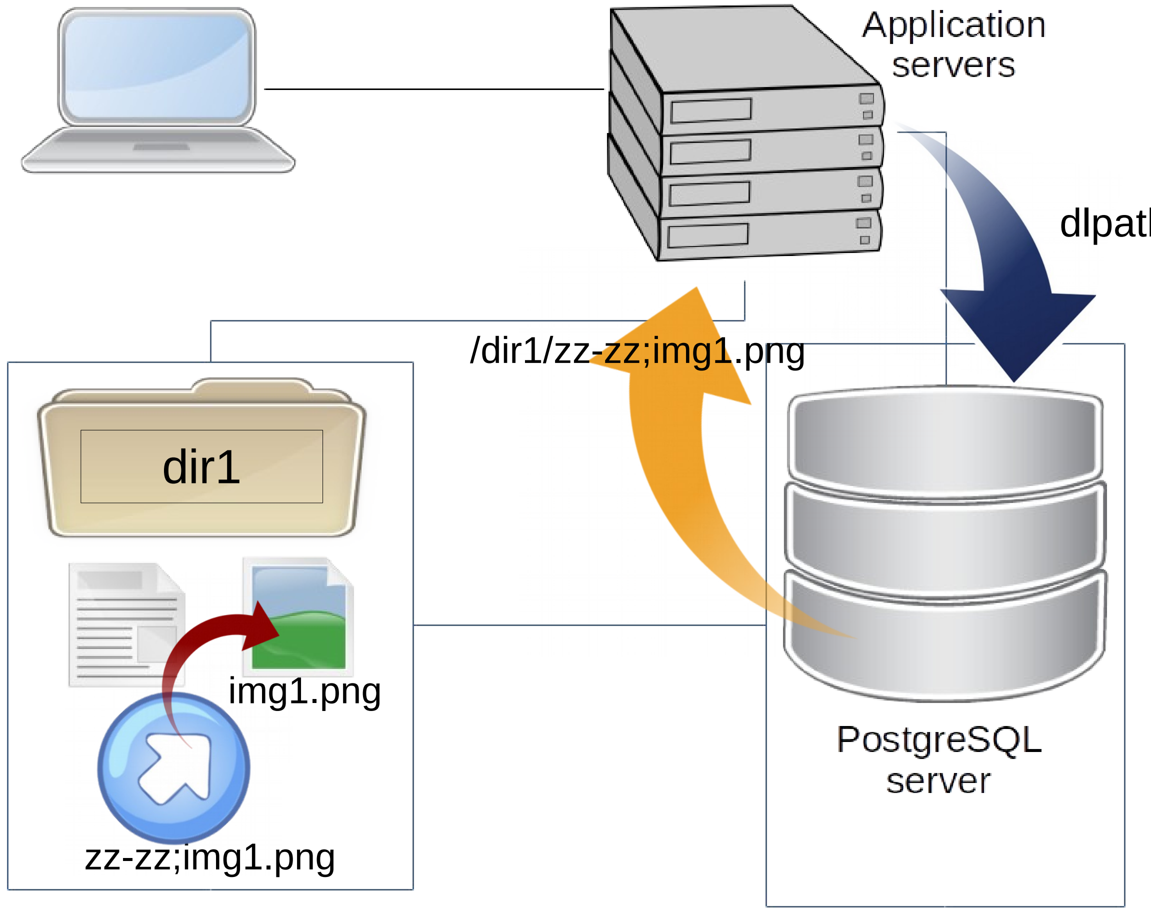
2.2) After creating the symlink the function creates a token file:

$PGDATA/pg_dltoken/zz-zz

The expiration time is taken from the creation date of the file.

The file contains the access mode: R, the TXID and the path to the file.

INTEGRITY ALL

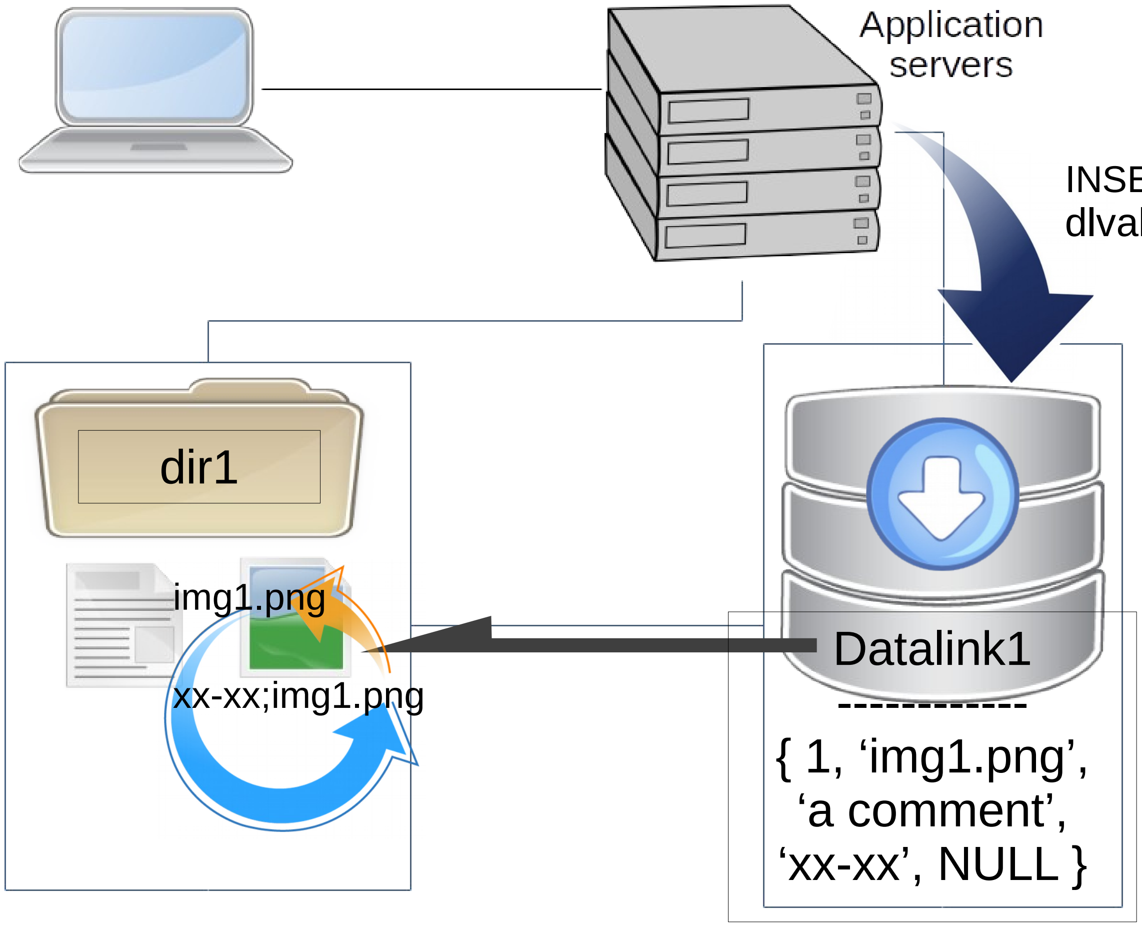


3) The DLPATHCOMPLETE() function return the path of the symlink with the token to the application.

The application can access the file until the token expires.

WRITING with concurrency

REQUIRING TOKEN FOR UPDATE

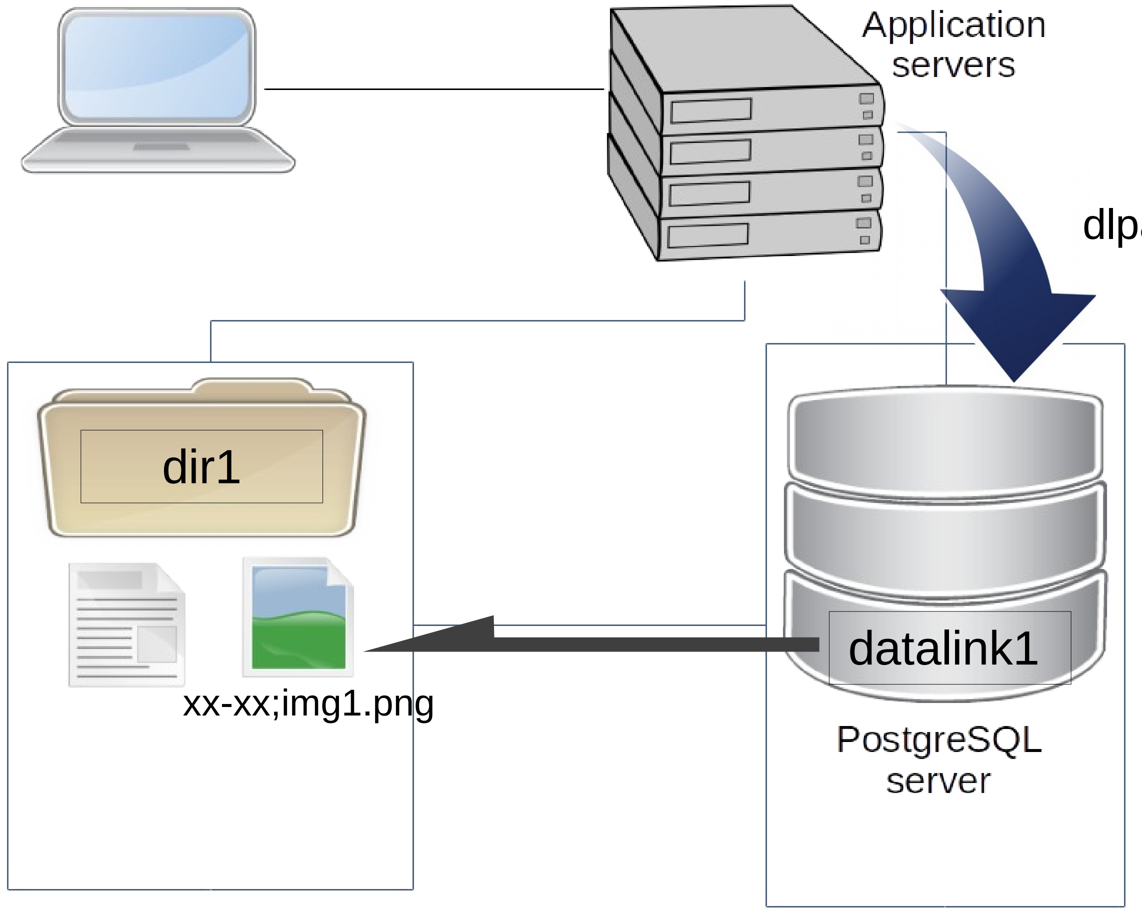

Application servers
INSERT +
dlvalue('img1.png', 'dir1', 'a comment')
dir1
img1.png
xx-xx;img1.png
Datalink1

{ 1, 'img1.png', 'a comment', 'xx-xx', NULL }

5) the application link an existing external file by using DLVALUE() with the file path associated to a base directory.

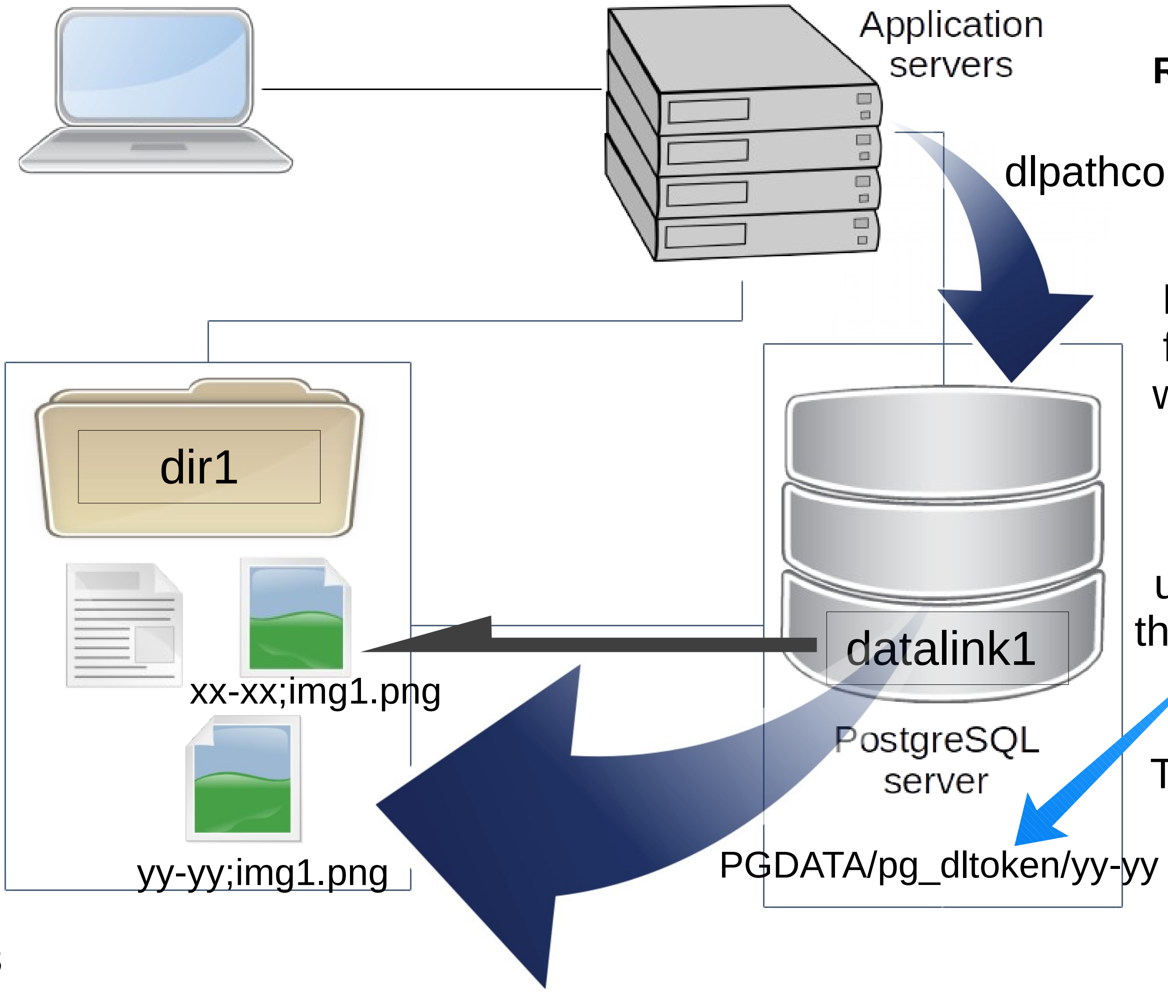
The DLVALUE() function rename the file with a new token and store the information in the Datalink data type.

REQUIRING TOKEN FOR UPDATE



1) The application ask to the DB an authorization to modify the file referenced in “datalink1” record

REQUIRING TOKEN FOR UPDATE

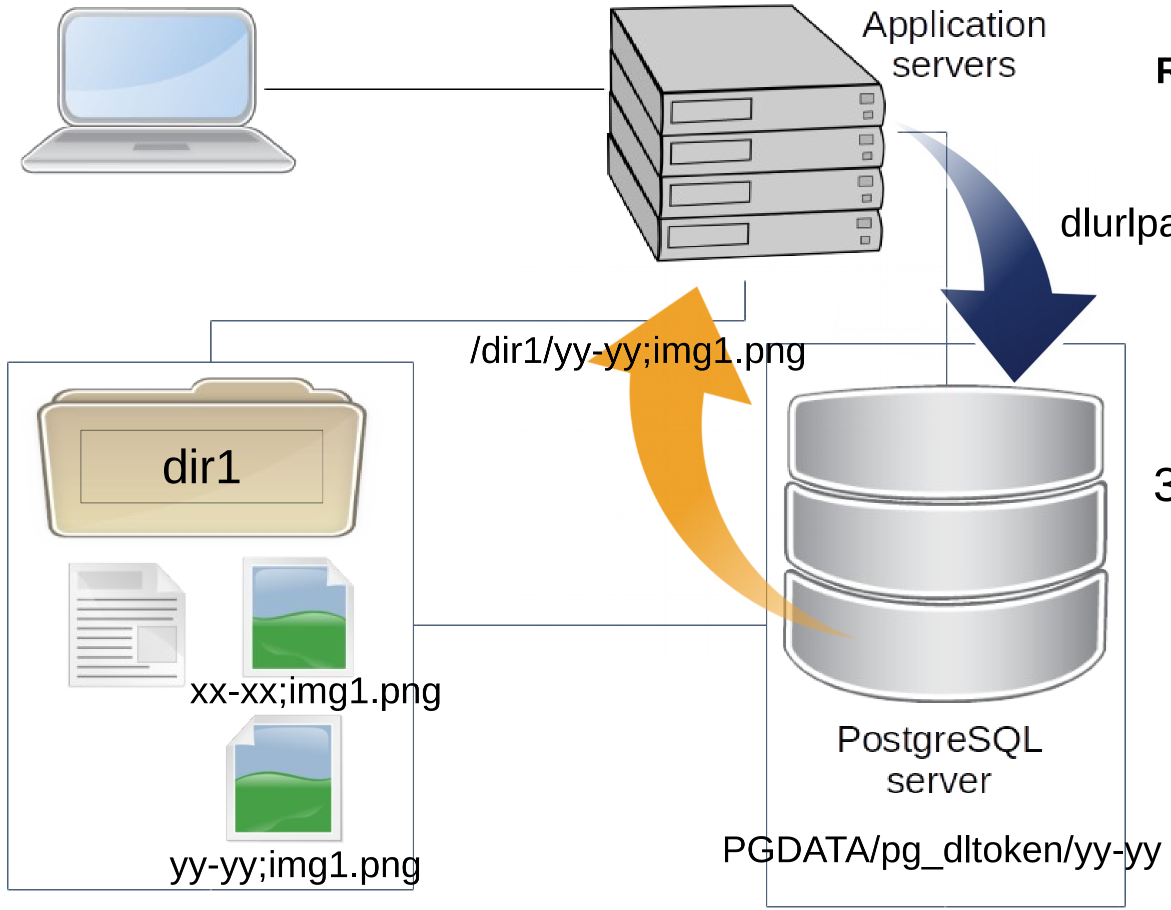


2) The DLPATHCOMPLETEWRITE() function lock and copy the file with a new token in destination filename.

The token is generated using uuid_generate_v4(). It prefixes the filename with a semicolon as separator.

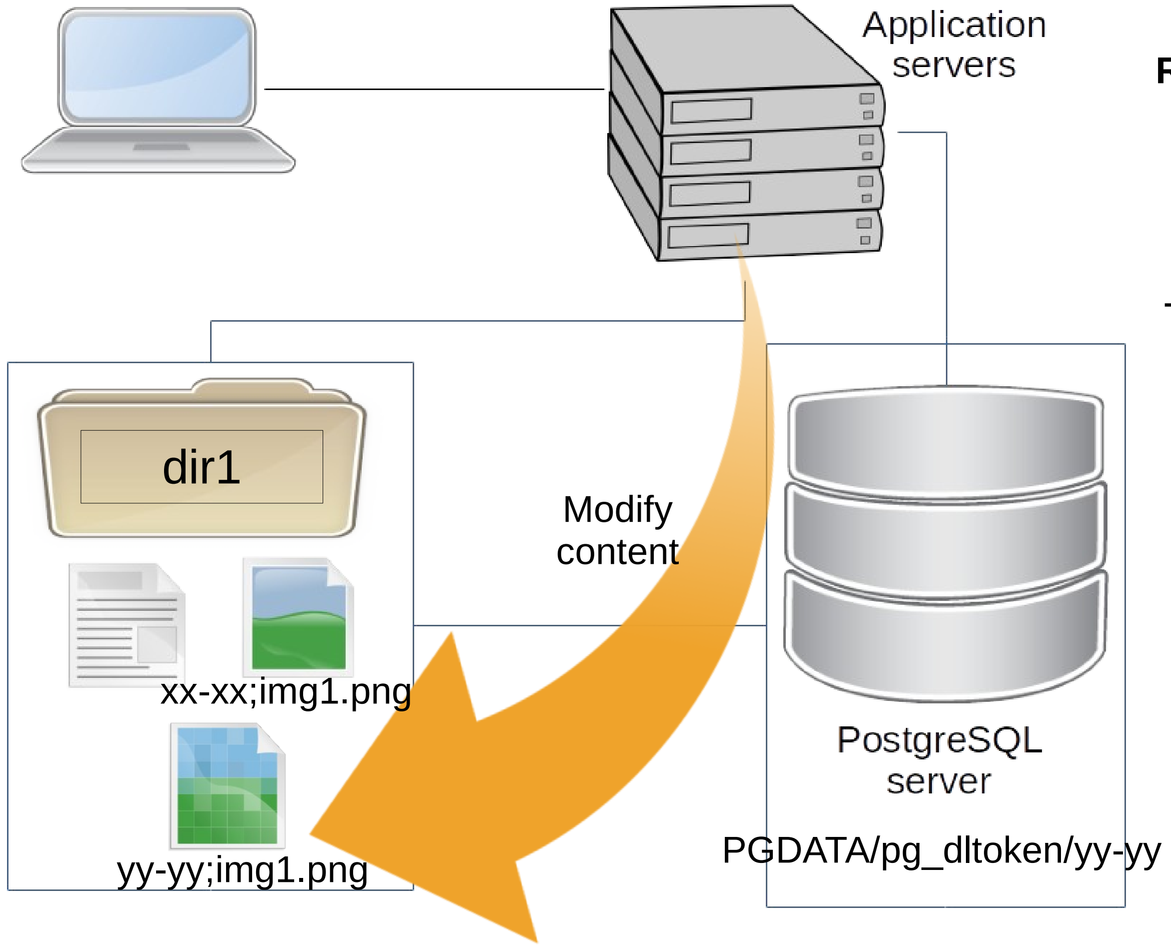
The application can access the file until the token expires.

REQUIRING TOKEN FOR UPDATE



3) The DLURLPATHWRITE() function return the path of the copy with the token to the application.

“/path/token;filename”

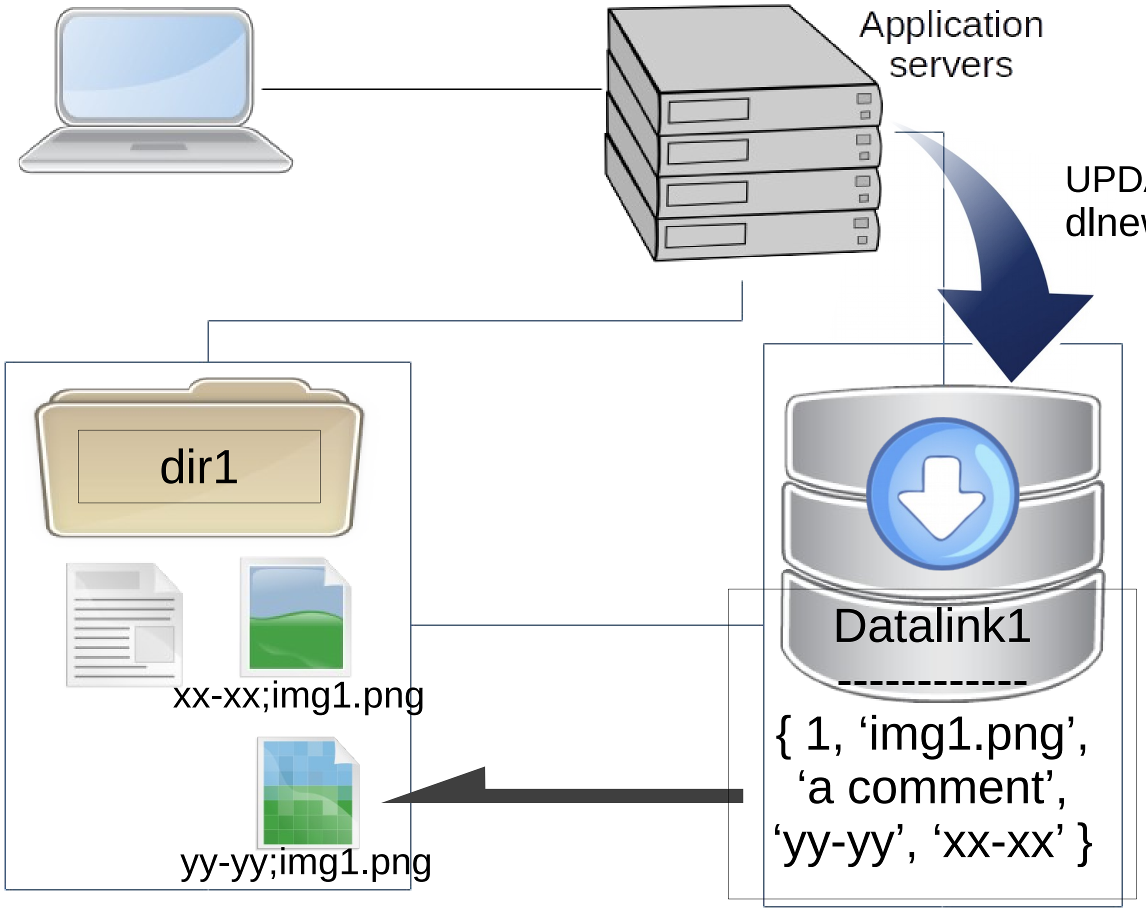


REQUIRING TOKEN FOR UPDATE

4) The application modify the content of the copy.

To access the file with a ';' inside the name, the path must quoted by the application.

REQUIRING TOKEN FOR UPDATE

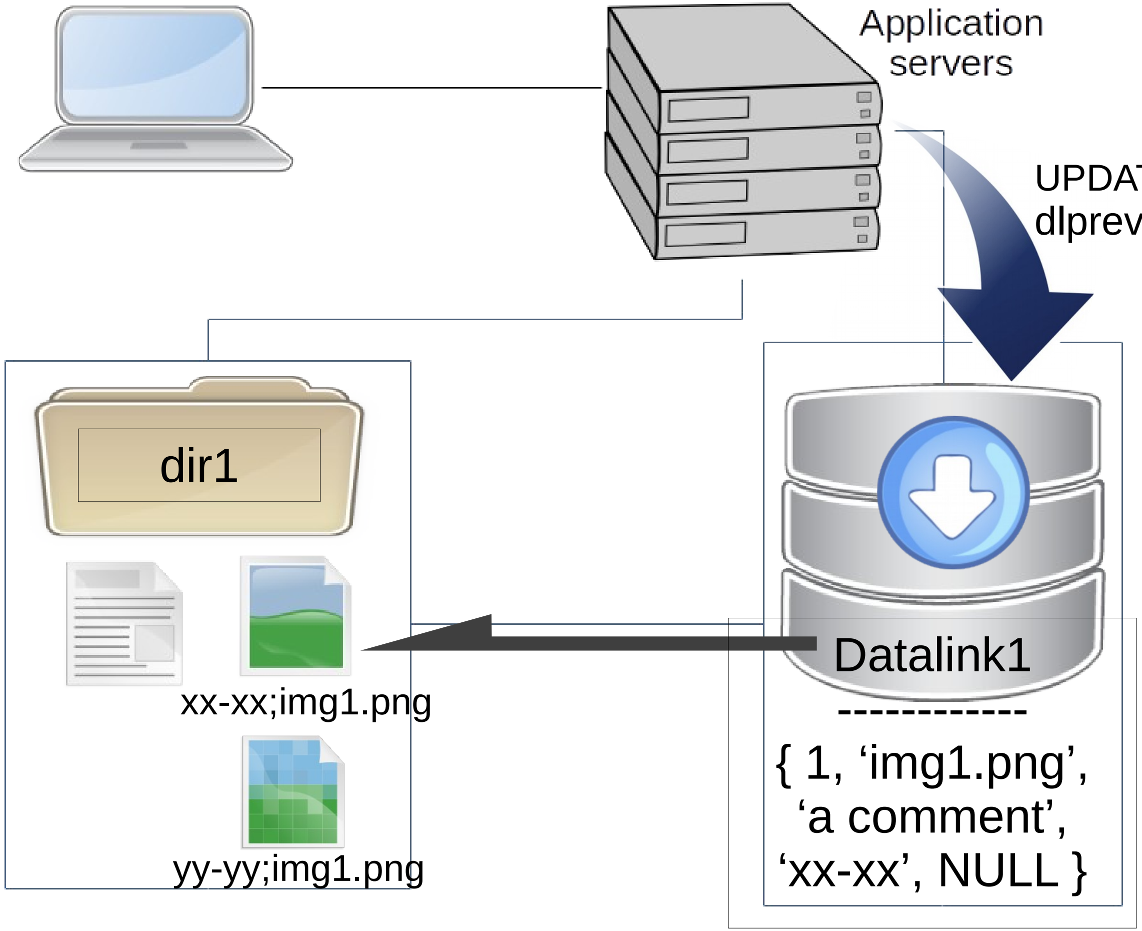

Application servers
UPDATE +
dlnewcopy('yy-yy;img1.png', true)
dir1
xx-xx;img1.png
yy-yy;img1.png
Datalink1

{ 1, 'img1.png', 'a comment', 'yy-yy', 'xx-xx' }

5) the application indicate to the DB that the copy is ready to be linked and that the URI has a token inside.

The DLNEWCOPY() function update the Datalink with the new token and store token of previous file into the dl_prevtoken field.

Restore to previous state



5) the application indicate to the DB that the previous copy shall be restored due to rollback.

The DLPREVIOUSCOPY() function just replace current token to xx-xx and set dl_prevtoken to NULL

Undo changes

After call to DLNEWCOPY() the Datalink returned looks like:

Datalink returned: { 1, ‘img1.png’, ‘a comment’, ‘yy-yy’, ‘xx-xx’ }

After call to DLPREVIOUSCOPY() the Datalink returned looks like:

Datalink returned: { 1, ‘img1.png’, ‘a comment’, ‘xx-xx’, NULL }

The dl_prev_token column could be changed into an uuid[N] to keep track of the last versions of a file and be able to undo up to N versions.

All versions of a file are preserved on disk, the “*dl_max_copies*” GUC value shall be set to remove automatically older versions.

Datalink Token

A token expires when:

- delta time with token file creation time > GUC *dl_token_expiry* (default 60 seconds)

- the transaction that creates the token has been committed or rollbacked

A token file contains:

- The access mode (read or write)
- The transaction id that creates the token
- The path to the external file

Concurrency with Datalink

Copy on Write

Write:

- A file is never modified in-place.
- A session modifying a file works on its own copy.
- Writers don’t block readers, writers don’t block writers

Read:

- Read access is done on the current version of a linked file.
- Readers don’t block readers, readers don’t block writers.

All access to external files must use a token in the URL/path.

Attribute NOT REQUIRING TOKEN FOR UPDATE does not allow concurrency and should be avoid.

Datalink is registered as: {1, ‘file1.txt’, ‘a comment’, ‘aa-aa-aa’, NULL}

Session 1: read file aa-aa-aa;file1.txt

BEGIN;
SELECT DLPATHCOMPLETE(datalink) …
Create a symlink:
    bb-bb;file1.txt → aa-aa-aa;file1.txt

The application open and read the file through link:

    /path/to/pg_datalink/bb-bb;file1.txt

Then close the file.

END;

Session 2: modify file aa-aa-aa;file1.txt

BEGIN;
SELECT DLPATHCOMPLETEWRITE(datalink) …
Create a copy of the current version:
    cp aa-aa-aa;file1.txt cc-cc-cc;file1.txt

The application works on file cc-cc-cc;file1.txt

SELECT DLNEWCOPY(datalink) …
Register the new copy by setting:
    previous token as aa-aa-aa
    current token to cc-cc-cc
COMMIT;

Datalink value: {1, ‘file1.txt’, ‘a comment’, ‘cc-cc-cc’, ‘aa-aa-aa’}

Datalink concurrency drawback

Each time a read access is requested a new symlink is created and not deleted.

Each time a file is modified a new file is created, in case of rollback it is not removed.

We can’t rely on the application to clean obsolete files, especially in case of application crash.

We need a process that will be able to detect obsolete READ symlink and file that must be removed after a ROLLBACK.

Datalink Garbage Cleaner

The Datalink Garbage Cleaner is a background worker that periodically:

- Read token: look for creation time of the token file. If greater than "*dl_token_expiry*" seconds:
  - eliminate obsolete symlinks with token created for reading
  - and remove the token file
- Write token: look for creation time of the token file. If greater than "*dl_token_expiry*" seconds:
  - read the TXID stored into the token file. Check if it's a rollbacked transaction then remove the file.
  - and remove the token file if the transaction in not in state "in progress".

Deleting a Datalink

A datalink is deleted when:

- a row with a datalink column is deleted
- a datalink column is set to NULL
- a datalink URI is set to an empty string

What happens to the files?

- ON UNLINK NONE: nothing, only possible is NO LINK CONTROL
- ON UNLINK DELETE: the file is automatically removed from disk
- ON UNLINK RESTORE: the file is restored to the state before it was linked (FS attributes)

The extension do it by triggers. At this time FS attributes are not restored, the file is just renamed to its original name without token.

DURABILITY

All versions of files are preserved if not explicitly deleted or *dl_max_copies* > 0.

Files need to be archived once they are linked to be sure that they can be restored at any point in time. Requires that RECOVERY YES is set.

The archiving must be asynchronous to not block applications.

In case of transaction ROLLBACK files must not be saved.

When DLVALUE()/DLNEWCOPY()/DLREPLACECONTENT() functions are called, path to files are registered into table *pg_datalink_archives*.

An external archiving daemon fsync the copies and archive the files found in the queue.

RECOVERY / PITR

What’s happen if we want to restore at a given time?

- PostgreSQL have PITR inside but files are external…

- Just restore your PostgreSQL cluster as usual.

    datalink record will have the token at time of the recovery

If not all copies have been preserved you need to restore the corresponding files → “Datalink reconciler”.

Files that have been created after and not linked anymore must be removed → “Datalink reconciler”.

Datalink reconciler process

A program provided by the extension to:

- Scan all database to get the list of Datalink columns.
- If not present, restore the file corresponding to the token value from the archive.
- Check the timestamp of the file corresponding to the token restored
  - and remove any corresponding files that have been created after this timestamp.

REPLICATION

All replica can mount the shared disk and control the files through SQL once they become primary server after a failover.

All replica can see immediately the file once it has been linked if they mount the same shared disk.

As we use copy on write it can work flawlessly with multi-master replication.

File replication can be done using any file system replication solution, like drbd for example (https://www.drbd.org).

Improvements / Todo list

- Write an archiver daemon.
- Write the Datalink reconciler.
- Allow to read/write files with different users than postgres.
- Allow controlling files at file system side (FS layer).
- Add support to read/write of remote URL.

Datalink Extension

WIP available at

- https://github.com/darold/datalink

Please star the project so that I can monitor the interest for the Datakink feature for PostgreSQL.

Rights and Attributions

Peter Eisenstraut conference at PgCon 2009 :
https://www.pgcon.org/2009/schedule/events/142.en.html

The DB2 Data Links information are taken from:
Managing Files Using DB2:
http://www.redbooks.ibm.com/abstracts/sg246280.html

Jim Melton:
https://sigmodrecord.org/publications/sigmodRecord/0209/jimmelton.pdf

https://en.wikipedia.org/wiki/SQL/MED

https://wiki.postgresql.org/wiki/DATALINK

Thanks ! Any questions?

http://github.com/darold/datalink

Gilles Darold < gilles@darold.net >