How to search a specific value in all tables (PostgreSQL)?

https://stackoverflow.com/questions/5350088/how-to-search-a-specific-value-in-all-tables-postgresql

How about dumping the contents of the database, then using `grep`?

```
$ pg_dump --data-only --inserts -U postgres your-db-name > a.tmp
$ grep United a.tmp
INSERT INTO countries VALUES ('US', 'United States');
INSERT INTO countries VALUES ('GB', 'United Kingdom');
```

The same utility, pg_dump, can include column names in the output.

Just change `--inserts` to `--column-inserts`. That way you can search for specific column

names, too. But if I were looking for column names, I'd probably dump the schema instead of the data.

```
$ pg_dump --data-only --column-inserts -U postgres your-db-name > a.tmp
$ grep country_code a.tmp
INSERT INTO countries (iso_country_code, iso_country_name)
VALUES ('US', 'United  States');
INSERT INTO countries (iso_country_code, iso_country_name)

VALUES ('GB', 'United Kingdom');
```

+1 free and simple. And if you want structure pg_dump can do that too.
Also if grep isn't your thing use what ever file content searching tool you want on
the dumped out structures and/or data.

If you want to grep text data (which is typically encoded in more recent versions of postgres),
you may need to `ALTER DATABASE your_db_name`
`SET bytea_output = 'escape';` on the database
(or a copy thereof) before dumping it. (I'm not seeing a way to specify this just for a
`pg_dump` command.)

can you explain in detail .. ? How to search string 'ABC' into all tables ?

Here's a pl/pgsql function that locates records where any column contains a specific value.

It takes as arguments the value to search in text format, an array of table names to search into

(defaults to all tables) and an array of schema names (`public` by default).

It returns a table structure with schema, name of table, name of column and pseudo-column

`ctid` (non-durable physical location of the row in the table, see System Columns)

```
CREATE OR REPLACE FUNCTION search_columns(
    needle text,
    haystack_tables name[] default '{}',
    haystack_schema name[] default '{public}'
)
RETURNS table(schemaname text, tablename text, columnname text, rowctid text)
AS $$
begin
  FOR schemaname,tablename,columnname IN
      SELECT c.table_schema,c.table_name,c.column_name
      FROM information_schema.columns c
      JOIN information_schema.tables t ON
        (t.table_name=c.table_name AND t.table_schema=c.table_schema)
      WHERE (c.table_name=ANY(haystack_tables) OR haystack_tables='{}')
        AND c.table_schema=ANY(haystack_schema)
        AND t.table_type='BASE TABLE'
  LOOP
    EXECUTE format('SELECT ctid FROM %I.%I WHERE cast(%I as text)=%L',
       schemaname,
       tablename,
       columnname,
       needle
    ) INTO rowctid;
    IF rowctid is not null THEN
      RETURN NEXT;
    END IF;
 END LOOP;
END;
$$ language plpgsql;
```

EDIT: this code is for PG 9.1 or newer.

Examples of use in a test database:

Search in all tables within public schema:

```
select * from search_columns('foobar');
 schemaname | tablename | columnname | rowctid
------------+-----------+------------+---------
 public     | s3        | usename    | (0,11)
 public     | s2        | relname    | (7,29)
 public     | w         | body       | (0,2)
(3 rows)
```

Search in a specific table:

```
 select * from search_columns('foobar','{w}');
 schemaname | tablename | columnname | rowctid
------------+-----------+------------+---------
 public     | w         | body       | (0,2)
(1 row)
```

Search in a subset of tables obtained from a select:

```
select * from search_columns('foobar', array(select table_name::name from
information_schema.tables where table_name like 's%'), array['public']);

 schemaname | tablename | columnname | rowctid
------------+-----------+------------+---------
 public     | s2        | relname    | (7,29)
 public     | s3        | usename    | (0,11)
(2 rows)
```

Get a result row with the corresponding base table and and ctid:

```
select * from public.w where ctid='(0,2)';
 title |  body  |         tsv
-------+--------+---------------------
 toto  | foobar | 'foobar':2 'toto':1
```

To test again a regular expression instead of strict equality, like grep, this:

```
SELECT ctid FROM %I.%I WHERE cast(%I as text)=%L
```

may be changed to:

```
SELECT ctid FROM %I.%I WHERE cast(%I as text) ~ %L
```

ERROR: syntax error at or near "default" LINE 3: haystack_tables name[] default '{}'
(Using PostgreSQL 8.2.17 and cannot upgrade) – Henno May 11 '14 at 8:54

@Henno: yes it requires PG-9.1. Edited now to make that explicit. To use it with older versions, you'll have to adapt it. – Daniel Vérité May 11 '14 at 12:27

Unfortunately I don't feel my skills + time is sufficient for that...
I tried googling for postgresql function default argument but I'm not even sure if the problem is name[] or the default keyword. – Henno May 11 '14 at 17:32

@Henno: I'm sure it's *default* and the 2nd problem will be *format()* which appeared in 9.1. What you may do is post a question linking to this one and asking how to convert that to 8.2. Also maybe you don't need these *haystack** parameters if you need to search in every schema/every table. – Daniel Vérité May 11 '14 at 18:16

The only tool I know which can do that is: SQL Workbench/J: http://www.sql-workbench.net/

A Java/JDBC based tool which offers a special (proprietary) SQL "command" to search through all (or just selected) tables in a database:

http://www.sql-workbench.net/manual/wb-commands.html#command-search-data
http://www.sql-workbench.net/wbgrepdata_png.html

Do you know if is it possible to search the name of a specific column instead of a specific data ?
And if someone think it could help. Here is @Daniel Vérité's function, with another param that

accept names of columns that can be used in search. This way it decrease the time of processing.

At least in my test it reduced a lot.

```
CREATE OR REPLACE FUNCTION search_columns(
    needle text,
    haystack_columns name[] default '{}',
    haystack_tables name[] default '{}',
    haystack_schema name[] default '{public}'
)
RETURNS table(schemaname text, tablename text, columnname text, rowctid text)
AS $$
begin
  FOR schemaname,tablename,columnname IN
      SELECT c.table_schema,c.table_name,c.column_name
      FROM information_schema.columns c
      JOIN information_schema.tables t ON
        (t.table_name=c.table_name AND t.table_schema=c.table_schema)
      WHERE (c.table_name=ANY(haystack_tables) OR haystack_tables='{}')
        AND c.table_schema=ANY(haystack_schema)
        AND (c.column_name=ANY(haystack_columns) OR haystack_columns='{}')
```

```
        AND t.table_type='BASE TABLE'
  LOOP
    EXECUTE format('SELECT ctid FROM %I.%I WHERE cast(%I as text)=%L',
       schemaname,
       tablename,
       columnname,
       needle
    ) INTO rowctid;
    IF rowctid is not null THEN
      RETURN NEXT;
    END IF;
 END LOOP;
END;
$$ language plpgsql;
```

Bellow is an example of usage of the search_function created above.

```
SELECT * FROM search_columns('86192700'
    , array(SELECT DISTINCT a.column_name::name FROM information_schema.columns AS a
            INNER JOIN information_schema.tables as b ON (b.table_catalog =
a.table_catalog AND b.table_schema = a.table_schema AND b.table_name = a.table_name)
        WHERE
            a.column_name iLIKE '%cep%'
            AND b.table_type = 'BASE TABLE'
            AND b.table_schema = 'public'
    )

    , array(SELECT b.table_name::name FROM information_schema.columns AS a
            INNER JOIN information_schema.tables as b ON (b.table_catalog =
a.table_catalog AND b.table_schema = a.table_schema AND b.table_name = a.table_name)
        WHERE
            a.column_name iLIKE '%cep%'
            AND b.table_type = 'BASE TABLE'
            AND b.table_schema = 'public')
);
```

Here's @Daniel Vérité's function with progress reporting functionality. It reports progress

in three ways:

1. by RAISE NOTICE;
2. by decreasing value of supplied {progress_seq} sequence from
3. {total number of colums to search in} down to 0;
4. by writing the progress along with found tables into text file, located in

5. c:\windows\temp\{progress_seq}.txt.

_

```
CREATE OR REPLACE FUNCTION search_columns(
    needle text,
    haystack_tables name[] default '{}',
    haystack_schema name[] default '{public}',
```

```
    progress_seq text default NULL
)
RETURNS table(schemaname text, tablename text, columnname text, rowctid text)
AS $$
DECLARE
currenttable text;
columnscount integer;
foundintables text[];
foundincolumns text[];
begin
currenttable='';
columnscount = (SELECT count(1)
      FROM information_schema.columns c
      JOIN information_schema.tables t ON
        (t.table_name=c.table_name AND t.table_schema=c.table_schema)
      WHERE (c.table_name=ANY(haystack_tables) OR haystack_tables='{}')
        AND c.table_schema=ANY(haystack_schema)
        AND t.table_type='BASE TABLE')::integer;
PERFORM setval(progress_seq::regclass, columnscount);

  FOR schemaname,tablename,columnname IN
      SELECT c.table_schema,c.table_name,c.column_name
      FROM information_schema.columns c
      JOIN information_schema.tables t ON
        (t.table_name=c.table_name AND t.table_schema=c.table_schema)
      WHERE (c.table_name=ANY(haystack_tables) OR haystack_tables='{}')
        AND c.table_schema=ANY(haystack_schema)
        AND t.table_type='BASE TABLE'
  LOOP
    EXECUTE format('SELECT ctid FROM %I.%I WHERE cast(%I as text)=%L',
       schemaname,
       tablename,
       columnname,
       needle
    ) INTO rowctid;
    IF rowctid is not null THEN
      RETURN NEXT;
      foundintables = foundintables || tablename;
      foundincolumns = foundincolumns || columnname;
      RAISE NOTICE 'FOUND! %, %, %, %', schemaname,tablename,columnname, rowctid;
    END IF;
         IF (progress_seq IS NOT NULL) THEN
        PERFORM nextval(progress_seq::regclass);
    END IF;
    IF(currenttable<>tablename) THEN
    currenttable=tablename;
     IF (progress_seq IS NOT NULL) THEN
        RAISE NOTICE 'Columns left to look in: %; looking in table: %',
currval(progress_seq::regclass), tablename;
        EXECUTE 'COPY (SELECT unnest(string_to_array(''Current table
(column ' || columnscount-currval(progress_seq::regclass) || ' of ' ||
columnscount || '): ' || tablename || '\n\nFound in tables/columns:\n' ||
COALESCE(
        (SELECT string_agg(c1 || '/' || c2, '\n') FROM
(SELECT unnest(foundintables) AS c1,unnest(foundincolumns) AS c2) AS t1)
        , '') || ''','''\n''))) TO ''c:\WINDOWS\temp\' || progress_seq || '.txt''';
    END IF;
    END IF;
 END LOOP;
```

```
END;
$$ language plpgsql;
```

Without storing a new procedure you can use a code block and execute to obtain a table of occurences.

You can filter results by schema, table or column name.

```
DO $$
DECLARE
  value int := 0;
  sql text := 'The constructed select statement';
  rec1 record;
  rec2 record;
BEGIN
  DROP TABLE IF EXISTS _x;
  CREATE TEMPORARY TABLE _x (
    schema_name text,
    table_name text,
    column_name text,
    found text
  );
  FOR rec1 IN
        SELECT table_schema, table_name, column_name
        FROM information_schema.columns
        WHERE table_name <> '_x'
                AND UPPER(column_name) LIKE UPPER('%%')
                AND table_schema <> 'pg_catalog'
                AND table_schema <> 'information_schema'
                AND data_type IN ('character varying', 'text',
'character', 'char', 'varchar')
        LOOP
    sql := concat('SELECT ', rec1."column_name", ' AS "found" FROM ',
rec1."table_schema" , '.',rec1."table_name" , ' WHERE
UPPER(',rec1."column_name" , ') LIKE UPPER(''','%%' , ''')');
    RAISE NOTICE '%', sql;
    BEGIN
        FOR rec2 IN EXECUTE sql LOOP
            RAISE NOTICE '%', sql;
            INSERT INTO _x VALUES (rec1."table_schema", rec1."table_name",
rec1."column_name", rec2."found");
        END LOOP;
    EXCEPTION WHEN OTHERS THEN
    END;
  END LOOP;
  END; $$;

SELECT * FROM _x;
```