Simple Versioning of Database Entries

Peter Seiderer

October 8, 2001

Abstract

For many internet applications user editable data is stored in databases. Therefore it is an advantage to provide some sort of version control to the stored information as provided by e.g. CVS, ClearCase or SourceSafe on a per file basis. In this article a simple way of providing part of the version control functionality for database information is described. For space saving purpose a simple algorithm for delta compressing is used.

1 Introduction

In the process of programming the source code is often stored in a Version Control System like CVS, ClearCase of SourceSafe, so you can trace back the development path of the source code, or in case of an mistake (e.g. a freshly introduced bug) to have the possibility to fall back to an earlier version. In many internet applications where user editable information is stored in a database some sort of version control would be of great advantage.

In the next chapter a simple solution for this problem is described. The solution uses a shadow table to store the older versions of database table entries.

In the third chapter a simple algorithm for delta compressing (computing the edited parts of a text) is described. This algorithm is then used in chapter four to introduce a space saving approach for the version control system.

2 First approach

In figure 2 an example of a database table is shown. The first row labeled `id` is the primary key, the second row labled `data1` is some integer value and the third row labeld `data2` is of type varchar. Such a (empty) table is easily created by the statement[1]:

```
CREATE TABLE table1
  (id SERIAL, data1 INT, data2 VARCHAR);
```

Figure 1: Example database table table1

id	data1	data2
1	314156	“some text”
2	22	“Hello World”
3	12	“not important”

A first proach to provide a simple form of version control for the database table table1 is to create a shadow table via the command

```
CREATE TABLE shadow_table1
  (id INT, data1 INT, data2 INT, version INT);
```

where old versions of the entries are automatically stored by a PL/SQL function which is triggered by an update on table entry.

Therefore first the PS/SQL function is created which stores the old entry plus an additional version number in the shadow table:

```
CREATE FUNCTION func_table1_vc ()
```

[1]For all database examples in this text postgres version 7.1.3 was used [pos]

```
    RETURNS OPAQUE AS '
  DECLARE
    v integer;
  BEGIN
    SELECT INTO v version FROM shadow_table1
      WHERE id = OLD.id
      ORDER BY version DESC LIMIT 1;
    IF v ISNULL THEN
      v := 0;
    ELSE
      v :=  v + 1;
    END IF;
    INSERT INTO shadow_table1
      VALUES (OLD.id,OLD.data1,OLD.data2,v);
    RETURN NEW;
  END;
' LANGUAGE 'plpgsql';
```

The PL/SQL function is called by the following trigger on a update:

```
CREATE TRIGGER trig_table1_vc
  AFTER UPDATE ON table1
  FOR EACH ROW EXECUTE
  PROCEDURE func_table1_vc();
```

So if the text of row two of the example is updated to “Hello Reader” the shadow table would look like the following:

Figure 2: Example database tables table1 and shadow_table1

id	data1	data2
1	314156	“some text”
2	22	“Hello Reader”
3	12	“not important”

id	data1	data2	version
2	22	“Hello World”	0

Note that by the use of the shadow table to store the whole version information no change of the original table is necessary and no adjustment of running applications needs to be done.

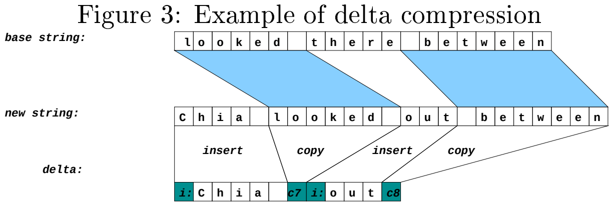
3 Delta compression

For large amounts of text where only minor changes where introduced, e.g. text editing by a user, this in chapter 2 shown approach is very inefficient in terms of space. It would be much better only to store the differences between the new version and the previous version. This is sometimes called delta compression in the literature.

As an example take the sentence “`looked there between`” which is changed to “`Chia looked out between`”[2].

Instead of storing the whole new sentence it would be sufficient to store only the edited parts “`Chia `” and “`out`” and the information which parts are copied from the original string as shown in figure 3.

Figure 3: Example of delta compression



For a detailed description on how to find such a delta see [Bur96] or [BCD95].

4 Space saving approach

The database procedure triggered by an update described in chapter 2 stores a whole copy of the corresponding database row. In this chapter a more advanced procedure is described which only stores the delta for text entries as described in chapter 3. Therefore two functions are used: *pg_diff* and *pg_patch* which are realized as C functions and incorporated via the postgres C interface:

```
CREATE FUNCTION pg_diff(text, text)
  RETURNS bytea AS 'libpg_ci_diff.so'
  LANGUAGE 'c' WITH (isStrict);

CREATE FUNCTION pg_patch(text, bytea)
```

[2] This sentence is the beginning of chapter 34 from [Gib97].

```
  RETURNS text AS 'libpg_ci_diff.so'
  LANGUAGE 'c' WITH (isStrict);
```

.

The new update triggered procedure is changed now to the following:

```
CREATE FUNCTION func_table1_vc ()
    RETURNS OPAQUE AS '
  DECLARE
    v integer;
  BEGIN
    SELECT INTO v version FROM shadow_table1
      WHERE id = OLD.id
      ORDER BY version DESC LIMIT 1;
    IF v ISNULL THEN
      v := 0;
    ELSE
      v :=  v + 1;
    END IF;
    INSERT INTO shadow_table1
      VALUES (OLD.id,OLD.data1,
       pg_diff(NEW.data2,OLD.data2),v);
    RETURN NEW;
  END;
' LANGUAGE 'plpgsql';
```

With this configuration the update command from the previous example plus a additional update of the text to "Hello Reader!" will lead to the following entries in the database tables table1 and shadow_table1:

Figure 4: Example database tables table1 and shadow_table1 (diff)

id	data1	data2
1	314156	"some text"
2	22	"Hello Reader!"
3	12	"not important"

id	data1	data2	version
2	22	"c:0:6:i:5:World"	0
2	23	"c:0:12"	1

The entry "c:0:6:i:5:World" means copy the first 6 characters from "Hello Reader" and then insert the 5 character long string "World".

This is not a space saving example but imagine a some kbytes long text and a user only changing some minor typos.

The only thing remaining now is a comfortable view on the shadow_table, so not only to show the delta files but the whole different versions. This is done by the following:

```
CREATE FUNCTION patch_data1 (integer, integer)
     RETURNS VARCHAR AS '
   DECLARE
     i ALIAS FOR $1;
     v ALIAS FOR $2;
     tmp varchar;
     delta RECORD;
   BEGIN
     SELECT INTO tmp data1 FROM test
      WHERE index = i;
     FOR delta IN SELECT data2 FROM shadow_test
       WHERE index = i
       AND version >= v ORDER BY version DESC LOOP
         tmp := pg_patch(tmp, delta.data2);
     END LOOP;
     RETURN tmp;
   END;
' LANGUAGE 'plpgsql';

CREATE VIEW view_shadow_table1 AS
    SELECT id AS id,
     data1 as data1
     patch_data2(index, version) AS data2
     version AS version FROM shadow_table1;
```

A `SELECT * FROM view_shadow_table1;` will lead to the following output:

Figure 5: Example database view view_shadow_table1

id	data1	data2	version
2	22	"Hello Reader"	0
2	22	"Hello Reader!"	1

5 Summary and Future Work

A simple and noninvasive method of database entry versioning was shown. To simplify the usage for a whole database a script for automatic generation of the shadow tables and the corresponding views and triggers would be helpful.

References

[BCD95] David T. Barnard, Gwen Clarke, and Nicolas Duncan. Tree-to-tree correction for document trees. Technical Report 95-372, Department of Computing and Information Science Queens's University, January 1995. citeseer.nj.nec.com/barnard95treetotree.html.

[Bur96] Randal C. Burns. Differential compression: A generalized solution for binary files. Master's thesis, University of California, Santa Cruz, December 1996.

[Gib97] William Gibson. *IDORU*. The Berkley Publishing Group, 1997.

[pos] http://www.postgresql.org.