HISTORICAL REFERENCE & DATA INTEGRITY IN E-GOVERNANCE PERSPECTIVE

HELLO!

I am Shaji Albert

I am here to discuss about my experience
with database design during the implementation
of our e-Governance projects

Kerala

I am from Kerala, one of the Southern states of India, blessed with its scenic beauty and progressive minded people.

Kerala has always showed its affinity towards the free software. ICFOSS formed in 2008, is the outcome of a decade of work by Free Software enthusiasts, advocates, developers and supporters in the state of Kerala and outside.

Stationery Department

Stationery Department is the centralised purchasing and distributing agency of Kerala Government. Every year this department spends crores of Rupees for the purchase of office & Computer consumables.
The department has a rich history and it claims its origin from the period of **Sri Swathi Thirunal Rama Varma** who ruled Thiruvithamkoor from 1829-1846

“

The Government has adopted the **Free and Open source technologies** as one of the basic guiding principles and shall strive for the promotion and adoption of I.T Policy-2017 18 the same.

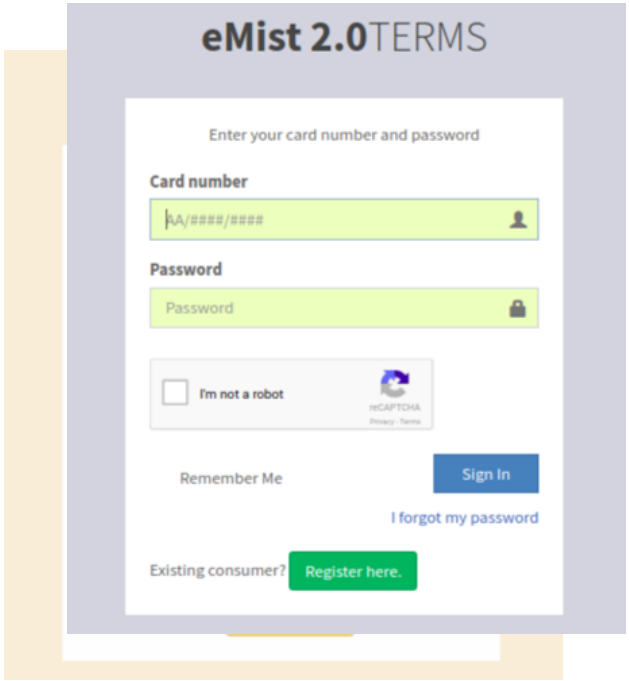
The Government shall make it mandatory for all software solutions made through public funding to adopt free and Open source technologies. Government shall support and promote “Volunteer Computing’ projects. Activities to achieve the objectives of IT policy 2007 in these matter shall be taken up

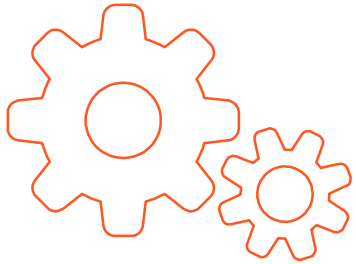
Kerala Government IT Policy 2017 Page 17

FOSS

- In Kerala its mandatory to use Free Open Source Softwares in all the software solutions made using public fund.
- Gov departments made significant progress in FOSS usage. All new softwares are developed using FOSS.
- Migrations of proprietary softwares to FOSS is under progress.
- Most of these softwares are using Postgres as back end

eMist 2.0 (TERMS)





eMist 2.0 (TERMS)

Stationery department is under the process of reconstructing the existing resource management application in latest technologies with simplified procedures after GPR.

Once the new application is implemented, all the consumer offices will be able to login to the system and request office consumables via online.

Prologue

Here, I am about to present few data storage and change tracking conundrums we have encountered during the designing of the software and the proposed solutions for those.

Two kind of tables

The tables in a relational database can be classified into two categories

1. Master data tables
2. Transaction data tables

Master data tables

These are the tables that hold master records such as Office names, Item names etc..

These table will have seldom insert operations, but frequent change requests.

Challenge

History of every change made in these master data tables need to be recorded.

It is not who last updated a record or not when that update is made .

But the complete change history of a record.

Solution

1.

Duplicate the tables

public schema	history schema
m_table1	his_table1
m_table2	his_table2
m_table3	his_table3

These tables will have an additional column named **operation_flag** which is an **ENUM ('i', 'u', 'd')**

Implement triggers

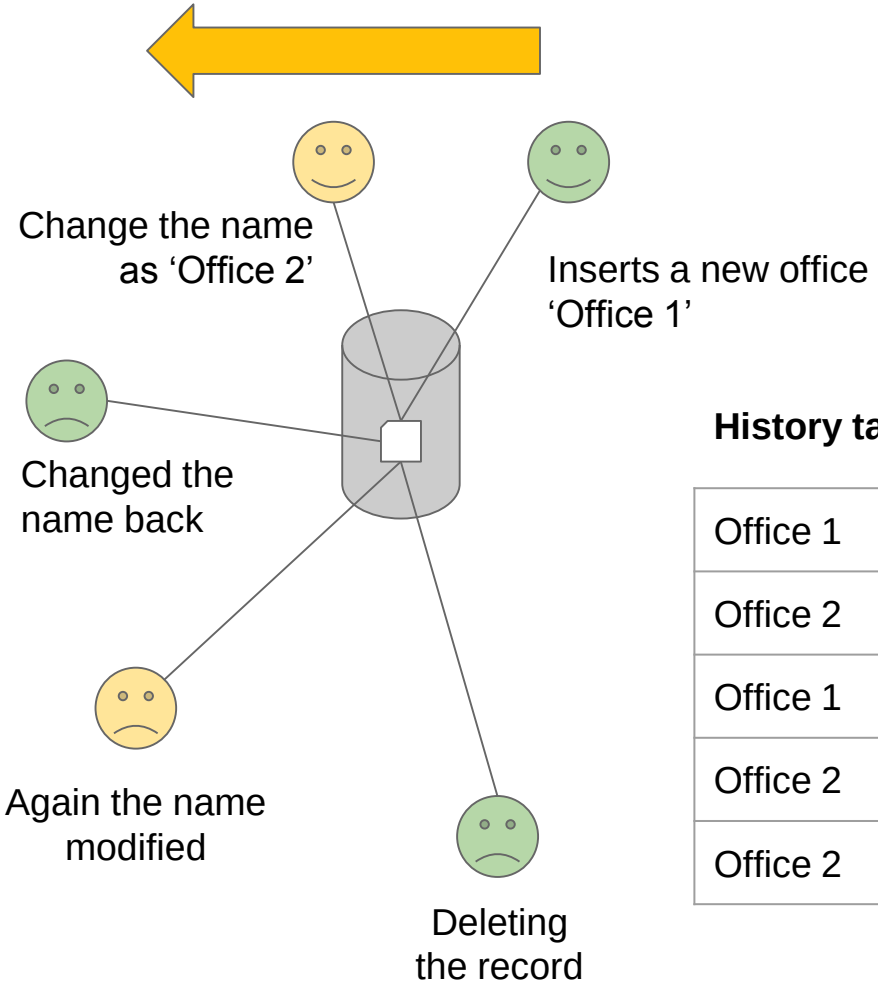
2.

m_table1

```
CREATE TRIGGER .. AFTER INSERT ..
.. <Copy new record to backup table with
operation_flag i>

CREATE TRIGGER .. AFTER UPDATE ..
.. <Copy updated record to backup table with
operation_flag u>

CREATE TRIGGER .. AFTER DELETE ..
.. <Copy deleted record to backup table with
operation_flag d>
```

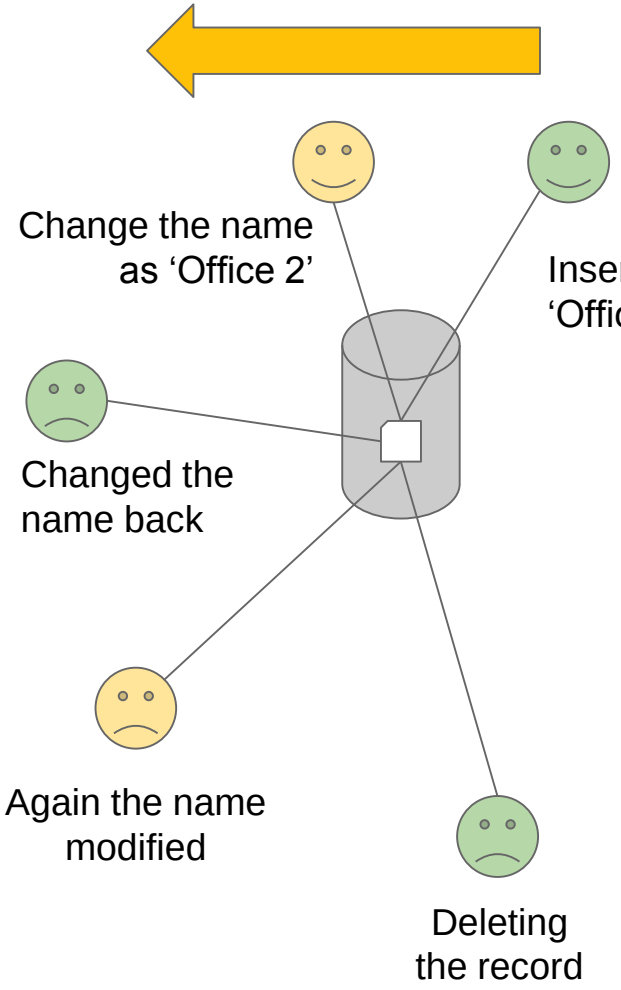


Ram

Kris

History table

Office 1	12:30 PM	Ram	i
Office 2	12:45 PM	Kris	u
Office 1	12:50 PM	Ram	u
Office 2	12:55 PM	Kris	u
Office 2	01:00 PM	Kris	d

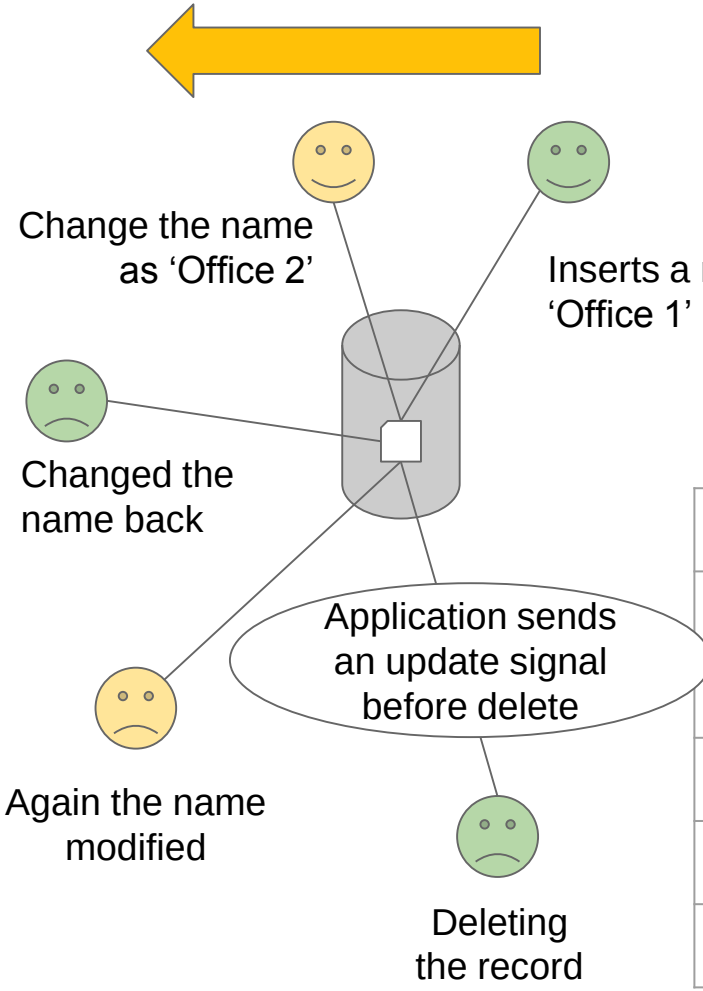


Ram

Kris

History table

Office 1	12:30 PM	Ram	i
Office 2	12:45 PM	Kris	u
Office 1	12:50 PM	Ram	u
Office 2	12:55 PM	Kris	u
Office 2	01:00 PM	Kris	d



Ram

Kris

History table

Office 1	12:30 PM	Ram	i
Office 2	12:45 PM	Kris	u
Office 1	12:50 PM	Ram	u
Office 2	12:55 PM	Kris	u
Office 2	01:00 PM	Ram	u
Office 2	01:00 PM	Ram	d

3.

Final result

By excluding the delete trigger and with application fine tuning we were able to achieve better result

Office 1	12:30 PM	Ram	i
Office 2	12:45 PM	Kris	u
Office 1	12:50 PM	Ram	u
Office 2	12:55 PM	Kris	u
Office 2	01:00 PM	Ram	d

12:30 PM
Today

Ram inserted a new office

12:45 PM
Today

Kris updated an office

View changes

12:50 PM
Today

Ram updated an office

View changes

12:55 PM
Today

Kris updated an office

View changes

12:50 PM
Today

Ram deleted an office

View changes

Application interface

Transaction tables

These tables are to record transactions such as correspondence, payment, transfer of stocks.

Unlike master tables update operations in Transaction tables are very rare.

Usually corrections in these tables are made by a new adjustment entry.

Why edit is restricted?

Each entries in a transaction table is dependent on previous entries.

Eg:

1. Issue of stock items that is credited as an entry in the same table.
2. Decisions of the officers based on the recommendations of their subordinates.

Edit or delete operations in a transaction table might change the ‘whole story’.

Case 1:
Malicious Edit

Original tuples	**Later one tuple is modified**
 **Subordinate:** I recommend the purchase of 5 new cars.	 **Subordinate:** I recommend the purchase of ~~5~~ **2** new cars.
 **Officer:** Purchase 3 new cars.	 **Officer:** Purchase 3 new cars.
Observation The decision is reasonable and the officer is wise.	**Observation** The decision is not reasonable. The officer is spendthrift

Case 2: Malicious Delete

Original tuples

Subordinate:
I recommend the purchase of 5 new cars.

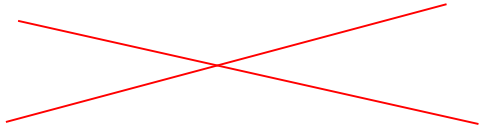
Superintendent:
As the department is in financial crisis, we can use the old cars.

Officer:
Purchase 3 new cars.

Observation
The officer is too ambitious to ignore the advice of the Superintendent.

Later one tuple is removed

Subordinate:
I recommend the purchase of 5 new cars.



Officer:
Purchase 3 new cars.

Observation
The decision is reasonable and the officer is wise.

Additional protection

Other than usual data security features we were thinking about additional measures. One solution came up is **Encryption**, but in the expense of resources.

Also the idea of encryption cannot solve the issue caused by malicious delete operations.

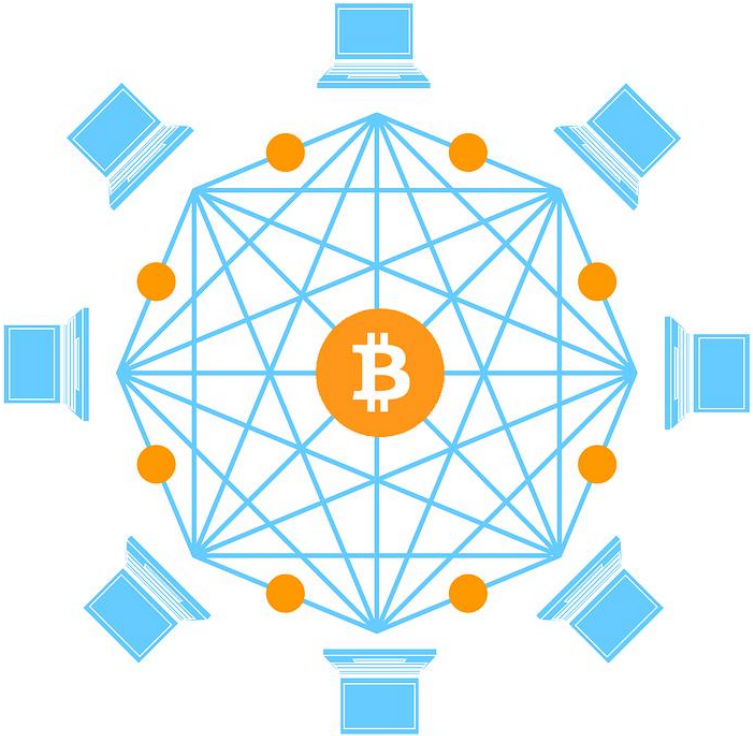
SOLUTION IS

VERY UNIQUE

AND FIRST OF

ITS KIND! 

OOPS!
WHO PUT THE PIC OF BLOCKCHAIN HERE?



https://www.benzinga.com/news/17/12/10941016/want-your-stock-to-rally-just-mention-bitcoin-or-blockchain

New Column hash_key

1.

On transaction tables, a new column hash_key varchar(32) added.

On very first row of the table the hash_key will be the md5 of the values of all columns of that row.

```
hash_key = md5( concat(
payment_id, customer_id, staff_id,
rental_id, amount, payment_date))
```

MD5

MD5("The quick brown fox jumps over the lazy dog") = 9e107d9d372bb6826bd81d3542a419d6

MD5("The quick brown fox jumps over the lazy dog.") = e4d909c290d0fb1ca068ffaddf22cbd0

hash_key Reference

The hash_key of next row will be md5 of the values of all columns + the hash_key of previous row

```
hash_key = md5( concat(
payment_id, customer_id, staff_id,
rental_id, amount, payment_date,
previous_hash_key
))
```

payment_date timestamp without time zone	hash_key character varying (32)
2007-05-14 13:44:29.996577	4b80f180aed28a47d811d405906e6755
2007-05-14 13:44:29.996577	2bc7ccefb8f4e52129917c08cc85add1
2007-05-14 13:44:29.996577	d1c122330fce369477afc5767ded3eab
2007-05-14 13:44:29.996577	6250c833bcdd64a8973bd5376b5446d3
2007-05-14 13:44:29.996577	8b9b3d60cb3ff76178beddf7a1fa70f3
2007-05-14 13:44:29.996577	5821795786646137e2a79c7308a800bf
2007-05-14 13:44:29.996577	6d4fa97300c3c88f7c709bb5998f9dbc
2007-05-14 13:44:29.996577	5cfd33f9dee2b97d3e95ec0d4e98d69c
2007-05-14 13:44:29.996577	621e8d96add01cee4daa56ea13deeca8
2007-05-14 13:44:29.996577	febd5dd4624b9d20f3f1131629208b2c
2007-05-14 13:44:29.996577	61b510bb22a365941f1451e9cbe8cdfe
2007-05-14 13:44:29.996577	979a9b21356b31f5bc3b9d5a7c4ed6d2
2007-05-14 13:44:29.996577	74b70c58307089b6ef3ad25d54fb4196
2007-05-14 13:44:29.996577	faf34a3257a3f1ad607661887e9b2c55

```
CREATE OR REPLACE FUNCTION check_payment(pay_id integer)
   RETURNS boolean AS
$func$
DECLARE
   my_rec RECORD;
BEGIN
   select payment_id, customer_id, staff_id,
      rental_id, amount, payment_date,
      md5( concat( payment_id, customer_id, staff_id, rental_id,
           amount, payment_date,
           (select hash_key from payment where payment_id =
               (select max(payment_id) from payment
                where payment_id<a.payment_id)))) gen_key,
      hash_key INTO my_rec
...
```

Using a PL/pgsql function the integrity of a record can be validated any time

```
Select check_payment(1204);
```

2.

Challenge

```
CREATE OR REPLACE FUNCTION payment_hash()
   RETURNS void AS
$func$
DECLARE
   pay_rec RECORD;
   my_hash VARCHAR(64);
BEGIN
   my_hash = '';
   FOR pay_rec IN
     SELECT payment_id, concat(payment_id, customer_id, staff_id,
        rental_id, amount, payment_date) rec_sumary from payment
   LOOP
     my_hash = md5(concat(pay_rec.rec_sumary, my_hash));
     UPDATE payment set hash_key = my_hash WHERE
     payment_id=pay_rec.payment_id;

     RAISE NOTICE '% - %', pay_rec.payment_id,my_hash;
   END LOOP;
END
$func$ LANGUAGE
plpgsql;
```

Using a PL/pgsql function all the hash_key can be updated sequentially

Suggestions

Adding an **application level key** as a co-ingredient of the hash_key provides protection against sequential update.

Keeping one or two insert-only backups at different locations will provide further assurance on data integrity.

THANKS!

Any questions?

You can find me at @ShajiAlbert &
shaji.albert@mail.com