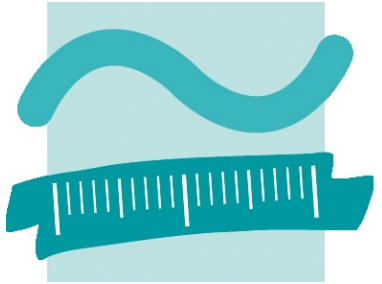
BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

PGMEMENTO

A GENERIC TRANSACTION-BASED AUDIT TRAIL FOR SPATIAL DATABASES

Felix Kunde and Petra Sauer



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences


BEUTH
University of Applied

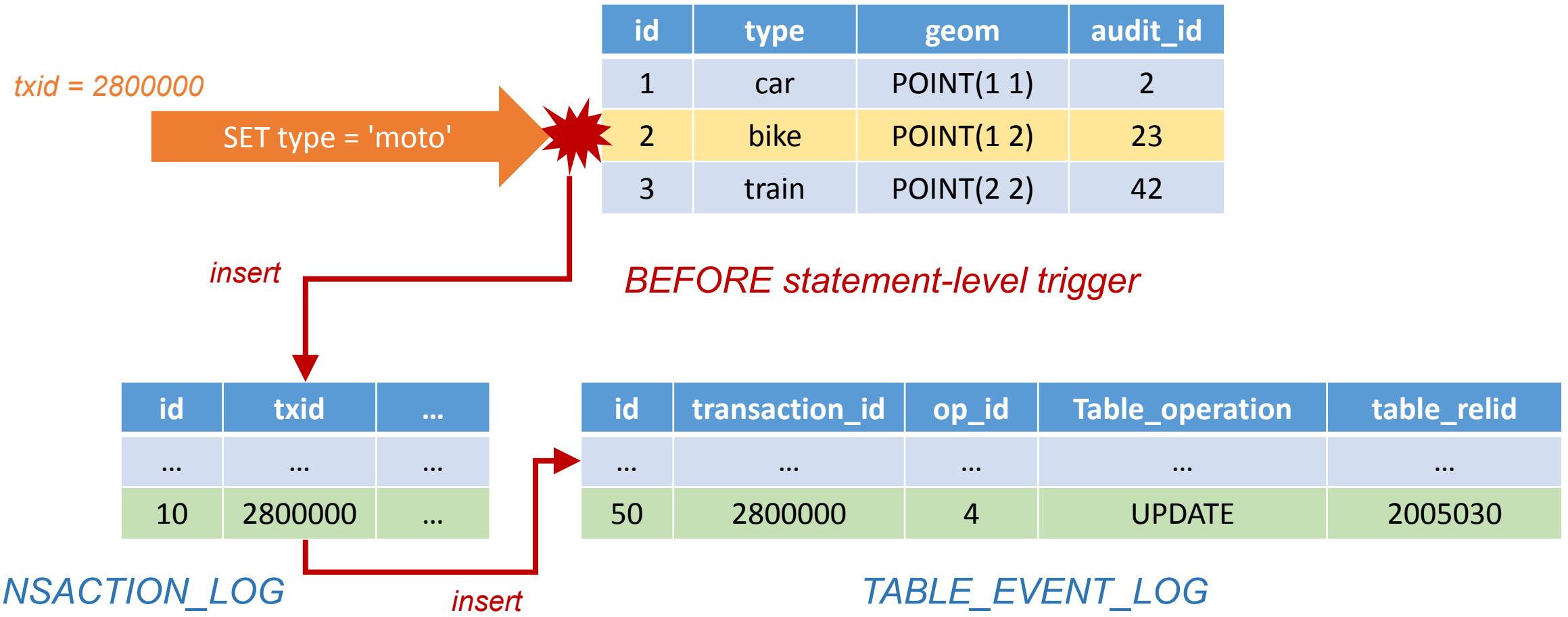
MOTIVATION

- Track all data changes in the database
- Revisit previous data versions
- Undo changes of certain write operations
- Work against multiple branches of a database

CONTEXT

- pgMemento is an Audit Trail in your DB
- Relies on transaction IDs, not timestamps
- Can help to define data lineage
- So far, its not version control to your DB

DML-AUDITING



id	type	geom	audit_id
1	car	POINT(1 1)	2
2	bike	POINT(1 2)	23
3	train	POINT(2 2)	42

id	txid	…
…	…	…
10	2800000	…

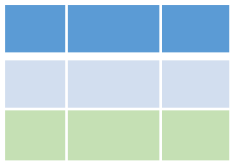
id	transaction_id	op_id	Table_operation	table_relid
…	…	…	…	…
50	2800000	4	UPDATE	2005030

DML-AUDITING

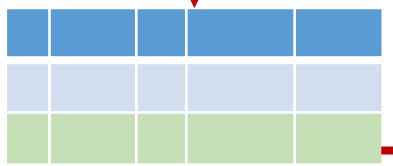
```
SELECT
  event_id
FROM
  table_event_log
WHERE
  transaction_id = txid_current()
  AND table_relid = 2005030
  AND op_id = 4;
```

id	type	geom	audit_id
1	car	POINT(1 1)	2
2	moto	POINT(1 2)	23
3	train	POINT(2 2)	42

AFTER row-level trigger



TRANSACTION_LOG



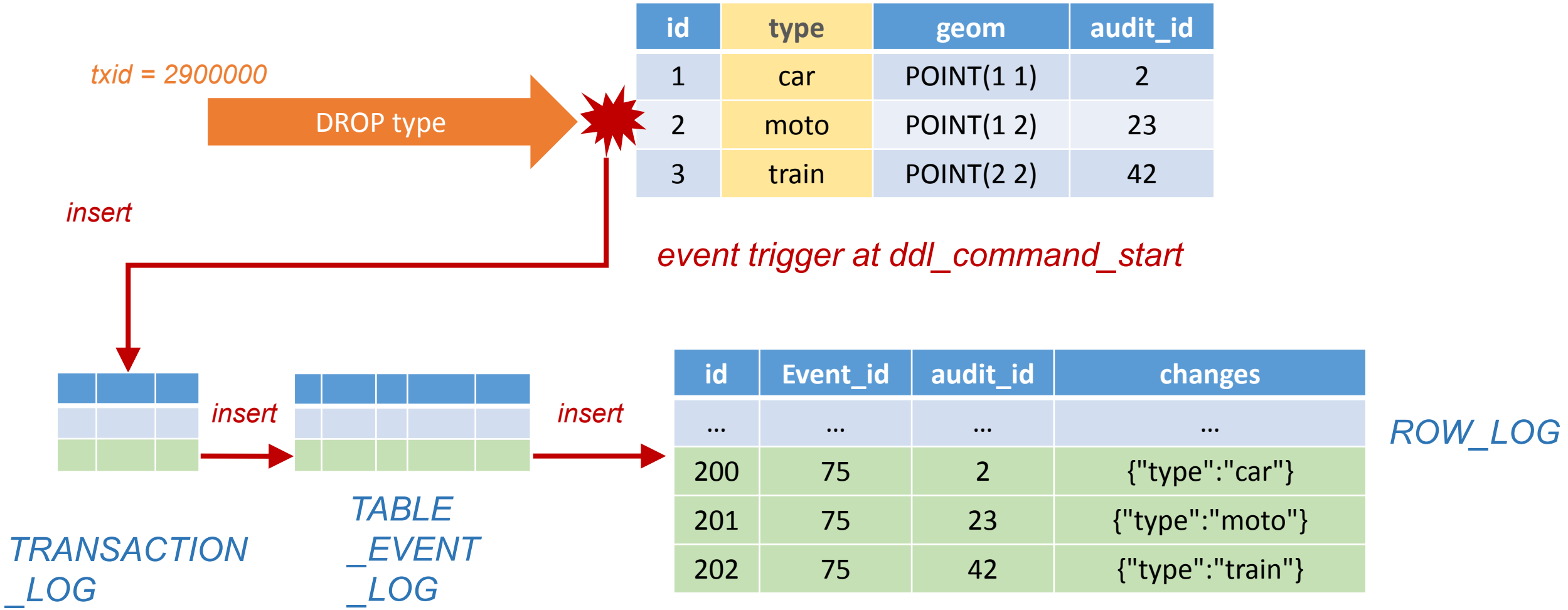
TABLE_EVENT_LOG

insert

id	Event_id	audit_id	changes
…	…	…	…
100	50	23	{"type":"bike"}

ROW_LOG

DDL-AUDITING



id	type	geom	audit_id
1	car	POINT(1 1)	2
2	moto	POINT(1 2)	23
3	train	POINT(2 2)	42

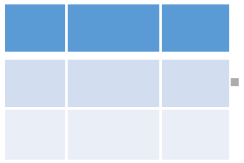
id	Event_id	audit_id	changes
…	…	…	…
200	75	2	{"type":"car"}
201	75	23	{"type":"moto"}
202	75	42	{"type":"train"}

DDL-AUDITING

id	geom	audit_id
1	POINT(1 1)	2
2	POINT(1 2)	23
3	POINT(2 2)	42

event trigger at ddl_command_end

```
SELECT * FROM pg_event_trigger_ddl_commands()
```



id	audit_table_id	column_name	data_type	txid_range
…	…	…	…	
100	1	type	text	[2700000,2900000)

AUDIT_TABLE_LOG

AUDIT_COLUMN_LOG

QUERIES

For which transactions column 'type' exists in the logs?

```
SELECT DISTINCT
  e.transaction_id
FROM
  pgmemento.table_event_log e
JOIN
  pgmemento.row_log r
  ON r.event_id = e.id
WHERE
  r.audit_id = 23
  AND (r.changes ? 'type');
```

Which tuples once contained certain combinations of key(s) and value(s)?

```
SELECT DISTINCT
  audit_id
FROM
  pgmemento.row_log
WHERE
  changes @> '{"type": "bike"}'::jsonb;
```

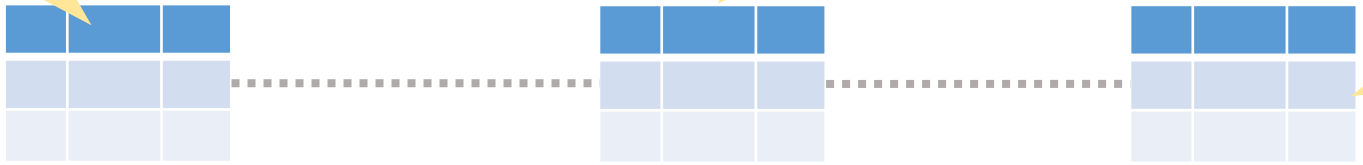
THE CHALLENGE

RESTORE PREVIOUS VERSIONS OF TUPLES

Define restore timestamp by picking a transaction ID

Filter data by table

Historic data is here. Highly scattered in case of UPDATES. No information about table, transaction or time



TRANSACTION_LOG *TABLE_EVENT_LOG* *ROW_LOG*

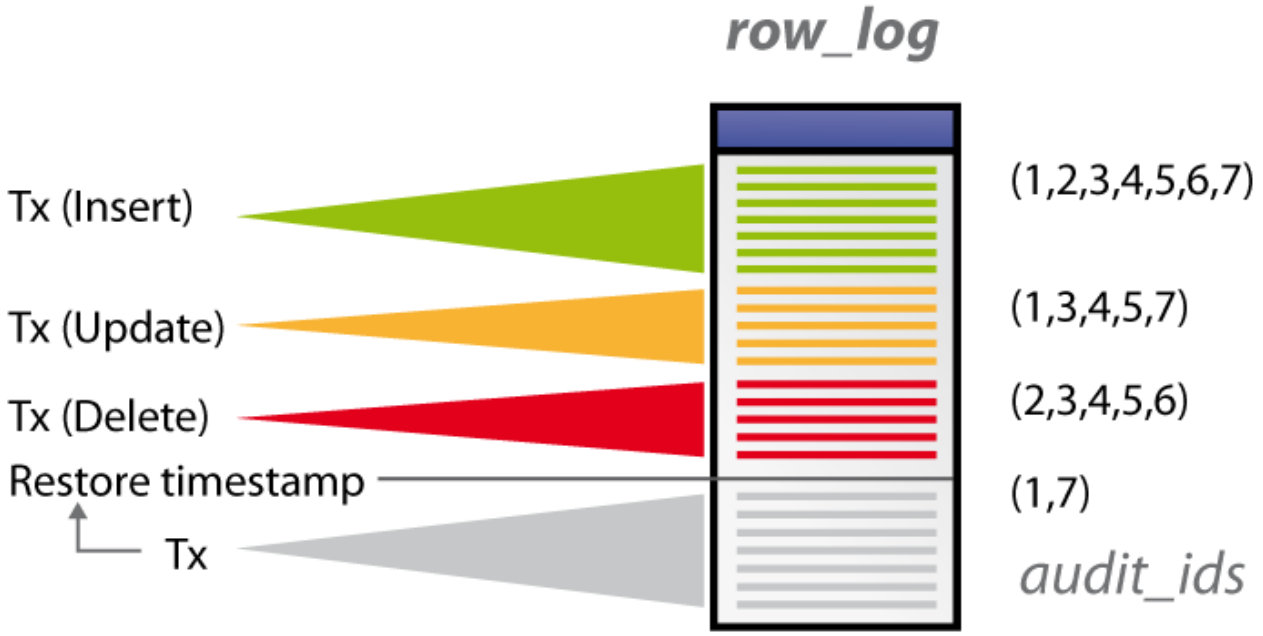
These tables know about the historic structure of relations

Query and merge JSONB logs to form complete tuples and populate them back to relational rows

AUDIT_TABLE_LOG *AUDIT_COLUMN_LOG*

RESTORE – PART 1

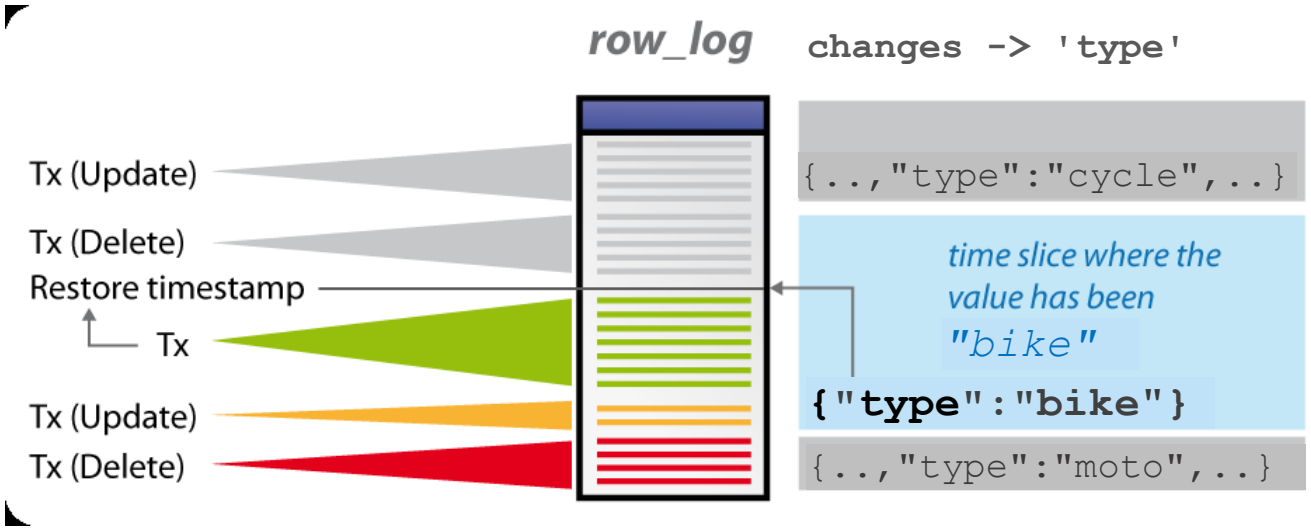
- Restore based on txid range (inclusive lower boundary and exclusive upper boundary)
- Only pick audit_ids where their last event havn‘t been a delete
- Apply the restore query for rows with audit_id 1 and 7



RESTORE – PART 2

- Strategy 1: Concat all JSONB logs in reverse order starting from recent state until upper boundary of search window as duplicate keys get overwritten

- Strategy 2: For each column, search for first occurence in the logs after upper boundary of search window. If nothing found query recent value.



RESTORE – PART 3

```
SELECT
  p.*
FROM
  generate_log_entries(1,2800000,'my_table') entries
LATERAL (
  SELECT
    *
  FROM
    jsonb_populate_record(
       null::my_table,
       entries
    )
) p;
```

Works only with a template. Could be the actual table, but to be correct in case of any DDL changes, a temporary template can be created on the fly with information from audit_column_log.

REVERT

- Query all changes and referenced events for a given txid (or range of txids) in reverse order
- Loop over result set and perform the opposite event
- Consider dependencies between tables in order to avoid foreign key violations

REVERT

op_id	Event	Reverse Event	Log Content
1	CREATE TABLE	DROP TABLE	-
2	ALTER TABLE ADD COLUMN	ALTER TABLE DROP COLUMN	-
3	INSERT	DELETE	NULL
4	UPDATE	UPDATE	Changed fields of changed rows
5	ALTER TABLE ALTER COLUMN	ALTER TABLE ALTER COLUMN	All rows of altered columns
6	ALTER TABLE DROP COLUMN	ALTER TABLE ADD COLUMN	All rows of deleted columns
7	DELETE	INSERT	All fields of deleted rows
8	TRUNCATE	INSERT	All fields of table
9	DROP TABLE	CREATE TABLE	All fields of table (logged as truncate)

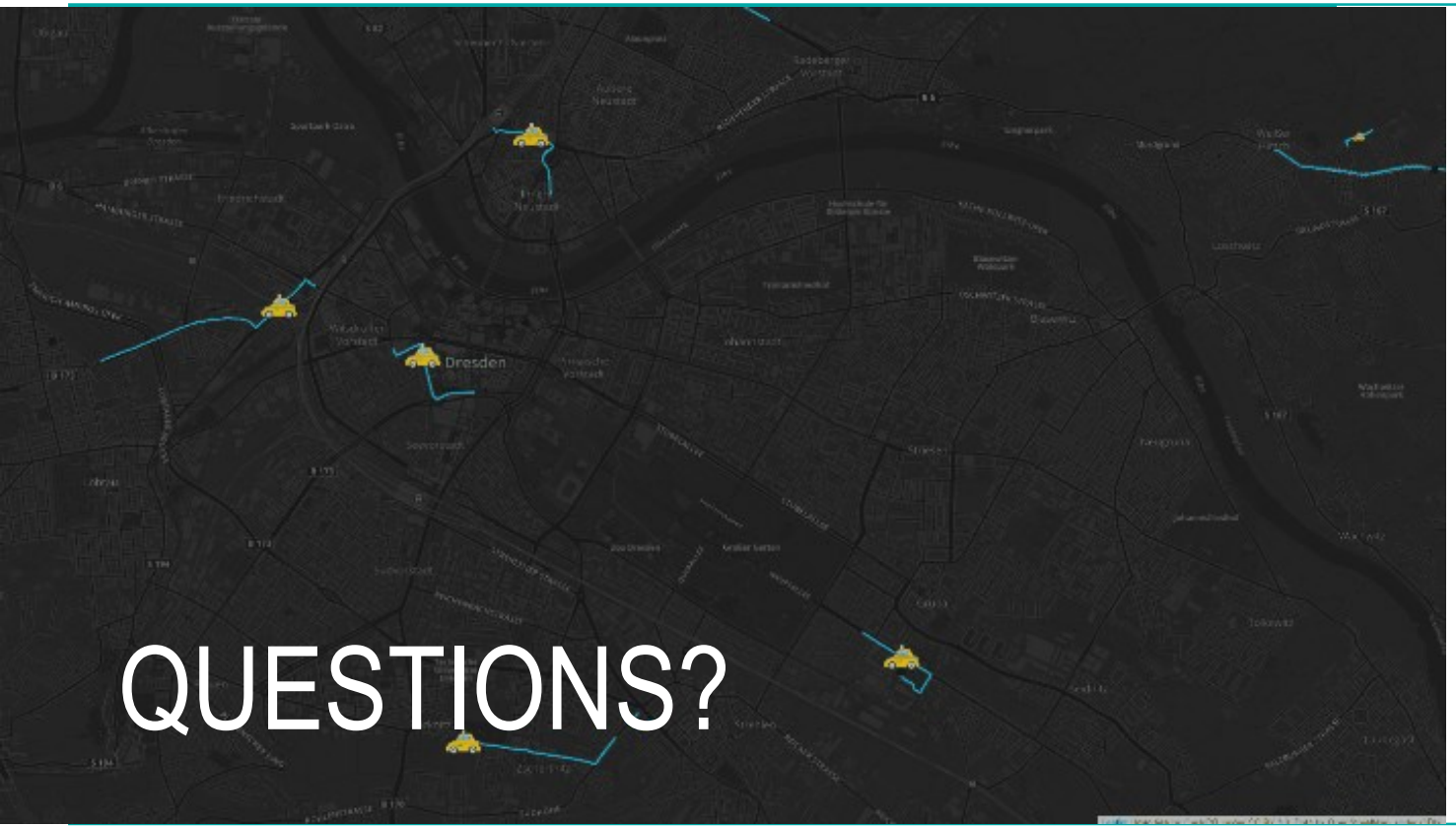
TO DOs

- Branching concept
- Log tables for more DB objects
- Extending the test suite
- Maybe: Logical decoding instead of triggers

TECHNICAL DETAILS

- Written entirely in PL/pgSQL
- Requires at least PostgreSQL 9.5
- Repo: github.com/pgmemento
- LGPL v3 Licence

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences



Felix Kunde

fkunde[at]beuth-hochschule.de

Petra Sauer

Sauer[at]beuth-hochschule.de

Funded by:

DLR

Bundesministerium für Wirtschaft und Energie

Smart Data