

Bitemporality: Tracking Reproducible Revisions in PostgreSQL Using RANGE Types

Miroslav Šedivý

 eumiro

Bitemporality: Tracking Reproducible Revisions in PostgreSQL Using RANGE Types

- INSERT, UPDATE and DELETE without losing information
 - Time-versioning entities with attributes
 - RANGE types in PostgreSQL 9.x+
 - GiST extension
 - Python and Psycopg2
 - Modifying data (concurrently)
 - Reading data (consistently)

Miroslav Šedivý

[ˈmɪrɔslav ˈʃɛɟɪviː]

- born in Bratislava, Czechoslovakia
- M.Sc. at INSA Lyon, France
- now working in Karlsruhe, Germany

- used MySQL and Oracle before PostgreSQL
- came to Python 2.5 from Perl/Java in 2008

A simple data model with records evolving over time

A simple data model with records evolving over time

name	born	alive_in_1992
Godefroy de Montmirail	1079	FALSE
Jacquouille la Fripouille	NULL	FALSE
Frénégonde de Pouilles	1095	FALSE
Béatrice de Montmirail	1964	TRUE
Jacques-Henri Jacquart	1952	TRUE
Hubert de Montmirail	1960	TRUE

A simple data model with records evolving over time

name	born	alive_in_1992
Godefroy de Montmirail	1079	FALSE
Jacquouille la Fripouille	NULL	FALSE
Frénégonde de Pouilles	1095	FALSE
Béatrice de Montmirail	1964	TRUE
Jacques-Henri Jacquart	1952	TRUE
Hubert de Montmirail	1960	TRUE

time_zone	utc_offset	observes_dst
Europe/Berlin	+01:00	TRUE
Europe/Paris	+01:00	TRUE

A really simple data model

```
CREATE TABLE customer (id INTEGER PRIMARY KEY,  
                        name TEXT NOT NULL UNIQUE,  
                        fee INTEGER NOT NULL);
```

A really simple data model

```
CREATE TABLE customer (id INTEGER PRIMARY KEY,  
                        name TEXT NOT NULL UNIQUE,  
                        fee INTEGER NOT NULL);
```

```
SELECT * FROM customer;
```

id	name	fee
1	alice	10
2	bob	20

A really simple data model

```
CREATE TABLE customer (id INTEGER PRIMARY KEY,  
                        name TEXT NOT NULL UNIQUE,  
                        fee INTEGER NOT NULL);
```

```
SELECT * FROM customer;
```

id	name	fee
1	alice	10
2	bob	20

```
INSERT INTO customer (id, name, fee) VALUES (3, 'carol', 30);
```

id	name	fee
1	alice	10
2	bob	20
3	carol	30

A really simple data model

```
CREATE TABLE customer (id INTEGER PRIMARY KEY,  
                        name TEXT NOT NULL UNIQUE,  
                        fee INTEGER NOT NULL);
```

```
SELECT * FROM customer;
```

id	name	fee
1	alice	10
2	bob	20

```
INSERT INTO customer (id, name, fee) VALUES (3, 'carol', 30);
```

id	name	fee
1	alice	10
2	bob	20
3	carol	30

- When did we insert the entry id = 3?

When did we insert an entry?

```
CREATE TABLE customer (id INTEGER PRIMARY KEY,  
                        name TEXT UNIQUE NOT NULL,  
                        fee INTEGER NOT NULL,  
                        inserted_on TIMESTAMPTZ NOT NULL DEFAULT NOW());
```

When did we insert an entry?

```
CREATE TABLE customer (id INTEGER PRIMARY KEY,  
                        name TEXT UNIQUE NOT NULL,  
                        fee INTEGER NOT NULL,  
                        inserted_on TIMESTAMPTZ NOT NULL DEFAULT NOW());
```

id	name	fee	inserted_on
1	alice	10	2019-01-01
2	bob	20	2019-01-01

When did we insert an entry?

```
CREATE TABLE customer (id INTEGER PRIMARY KEY,  
                        name TEXT UNIQUE NOT NULL,  
                        fee INTEGER NOT NULL,  
                        inserted_on TIMESTAMPTZ NOT NULL DEFAULT NOW());
```

id	name	fee	inserted_on
1	alice	10	2019-01-01
2	bob	20	2019-01-01

```
INSERT INTO customer (id, name, fee) VALUES (3, 'carol', 30);
```

id	name	fee	inserted_on
1	alice	10	2019-01-01
2	bob	20	2019-01-01
3	carol	30	2019-03-12

When did we insert an entry?

```
CREATE TABLE customer (id INTEGER PRIMARY KEY,  
                        name TEXT UNIQUE NOT NULL,  
                        fee INTEGER NOT NULL,  
                        inserted_on TIMESTAMPTZ NOT NULL DEFAULT NOW());
```

id	name	fee	inserted_on
1	alice	10	2019-01-01
2	bob	20	2019-01-01

```
INSERT INTO customer (id, name, fee) VALUES (3, 'carol', 30);
```

id	name	fee	inserted_on
1	alice	10	2019-01-01
2	bob	20	2019-01-01
3	carol	30	2019-03-12

- UPDATE with updated_on?

When did we insert an entry?

```
CREATE TABLE customer (id INTEGER PRIMARY KEY,  
                        name TEXT UNIQUE NOT NULL,  
                        fee INTEGER NOT NULL,  
                        inserted_on TIMESTAMPTZ NOT NULL DEFAULT NOW());
```

id	name	fee	inserted_on
1	alice	10	2019-01-01
2	bob	20	2019-01-01

```
INSERT INTO customer (id, name, fee) VALUES (3, 'carol', 30);
```

id	name	fee	inserted_on
1	alice	10	2019-01-01
2	bob	20	2019-01-01
3	carol	30	2019-03-12

- UPDATE with updated_on?
- DELETE with ... ?

When is the entry valid?

```
CREATE TABLE customer (id INTEGER NOT NULL, -- NOT PRIMARY KEY
                        name TEXT NOT NULL, -- NOT UNIQUE
                        fee INTEGER NOT NULL,
                        valid_since TIMESTAMPTZ NOT NULL DEFAULT NOW(),
                        valid_until TIMESTAMPTZ NOT NULL DEFAULT NULL); -- or infinity?
```


When is the entry valid?

```
CREATE TABLE customer (id INTEGER NOT NULL,           -- NOT PRIMARY KEY
                        name TEXT NOT NULL,           -- NOT UNIQUE
                        fee INTEGER NOT NULL,
                        valid_since TIMESTAMPTZ NOT NULL DEFAULT NOW(),
                        valid_until TIMESTAMPTZ NOT NULL DEFAULT NULL); -- or infinity?
```

id	name	fee	valid_since	valid_until
1	alice	10	2019-01-01	NULL
2	bob	20	2019-01-01	NULL

When is the entry valid?

```
CREATE TABLE customer (id INTEGER NOT NULL,           -- NOT PRIMARY KEY
                        name TEXT NOT NULL,           -- NOT UNIQUE
                        fee INTEGER NOT NULL,
                        valid_since TIMESTAMPTZ NOT NULL DEFAULT NOW(),
                        valid_until TIMESTAMPTZ NOT NULL DEFAULT NULL);  -- or infinity?
```

id	name	fee	valid_since	valid_until
1	alice	10	2019-01-01	NULL
2	bob	20	2019-01-01	NULL

```
INSERT INTO customer (id, name, fee) VALUES (3, 'carol', 30);
```

id	name	fee	valid_since	valid_until
1	alice	10	2019-01-01	NULL
2	bob	20	2019-01-01	NULL
3	carol	30	2019-03-12	NULL

Let's update an entry

```
id | name | fee | valid_since | valid_until
```

```
-----  
1 | alice | 10 | 2019-01-01 | NULL  
2 | bob   | 20 | 2019-01-01 | NULL  
3 | carol | 30 | 2019-03-12 | NULL
```

```
UPDATE customer SET valid_until = NOW() WHERE id = 1 and valid_until IS NULL;
```

```
INSERT INTO customer (id, name, fee) VALUES (1, 'alice', 15);
```

```
id | name | fee | valid_since | valid_until
```

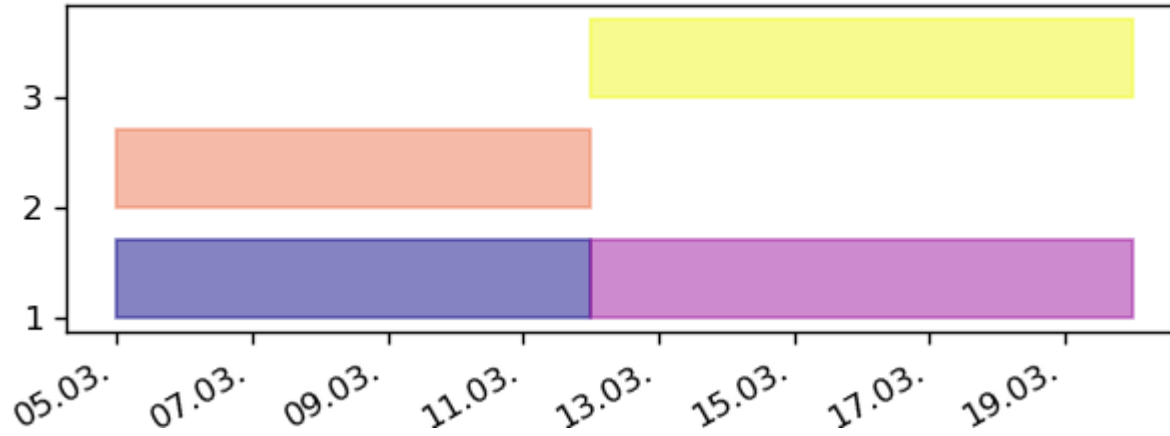
```
-----  
1 | alice | 10 | 2019-01-01 | 2019-03-12  
1 | alice | 15 | 2019-03-12 | NULL  
2 | bob   | 20 | 2019-01-01 | NULL  
3 | carol | 30 | 2019-03-12 | NULL
```

Let's delete (deactivate) an entry

id	name	fee	valid_since	valid_until
1	alice	10	2019-01-01	2019-03-12
1	alice	15	2019-03-12	NULL
2	bob	20	2019-01-01	NULL
3	carol	30	2019-03-12	NULL

```
UPDATE customer SET valid_until = NOW() WHERE id = 2 and valid_until IS NULL;
```

id	name	fee	valid_since	valid_until
1	alice	10	2019-01-01	2019-03-12
1	alice	15	2019-03-12	NULL
2	bob	20	2019-01-01	2019-03-12
3	carol	30	2019-03-12	NULL



id	name	fee	valid_since	valid_until
1	alice	10	2019-01-01	2019-03-12
1	alice	15	2019-03-12	NULL
2	bob	20	2019-01-01	2019-03-12
3	carol	30	2019-03-12	NULL

RANGES

id	name	fee	valid_since	valid_until
----	------	-----	-------------	-------------

1	alice	10	2019-01-01	2019-03-12
---	-------	----	------------	------------

id	name	fee	valid
----	------	-----	-------

1	alice	10	[2019-01-01, 2019-03-12)
---	-------	----	--------------------------

RANGES

```
id | name | fee | valid_since | valid_until
```

```
-----  
1 | alice | 10 | 2019-01-01 | 2019-03-12
```

```
id | name | fee | valid
```

```
-----  
1 | alice | 10 | [2019-01-01, 2019-03-12)
```

```
INT4RANGE | INT  
INT8RANGE | BIGINT  
NUMRANGE  | NUMERIC  
TSRANGE   | TIMESTAMP  
TSTZRANGE | TIMESTAMPTZ  
DATERANGE | DATE
```

... or define your own range types!

RANGES

```
id | name | fee | valid_since | valid_until
-----
1 | alice | 10 | 2019-01-01 | 2019-03-12
```

```
id | name | fee | valid
-----
1 | alice | 10 | [2019-01-01, 2019-03-12)
```

```
INT4RANGE | INT
INT8RANGE | BIGINT
NUMRANGE  | NUMERIC
TSRANGE   | TIMESTAMP
TSTZRANGE | TIMESTAMPTZ
DATERANGE | DATE
```

... or define your own range types!

```
[1, 4]: 1, 2, 3, 4
[1, 4): 1, 2, 3    ---> Pythonic range(1, 4)
(1, 4]: 2, 3, 4
(1, 4): 2, 3

(NULL,
 , NULL)
```

[..., NOW()) and [NOW(), ...) are adjacent in μ s!

RANGES

```
id | name | fee | valid_since | valid_until
-----
1 | alice | 10 | 2019-01-01 | 2019-03-12
```

```
id | name | fee | valid
-----
1 | alice | 10 | [2019-01-01, 2019-03-12)
```

```
INT4RANGE | INT
INT8RANGE | BIGINT
NUMRANGE  | NUMERIC
TSRANGE   | TIMESTAMP
TSTZRANGE | TIMESTAMPTZ
DATERANGE | DATE
```

... or define your own range types!

```
[1, 4]: 1, 2, 3, 4
[1, 4): 1, 2, 3    ---> Pythonic range(1, 4)
(1, 4]: 2, 3, 4
(1, 4): 2, 3
(NULL,
 , NULL)
```

[..., NOW()) and [NOW(), ...) are adjacent in μ s!

```
= <> < > <= >= @> && << >> &< &> -|- + * - LOWER( UPPER( LOWER_INF( UPPER_INF(
```

```
CREATE TABLE customer (id INTEGER NOT NULL, -- NO PRIMARY KEY
                        name TEXT NOT NULL,
                        fee INTEGER NOT NULL,
                        valid TSTZRANGE NOT NULL DEFAULT TSTZRANGE(NOW(), NULL));
```

```
CREATE TABLE customer (id INTEGER NOT NULL, -- NO PRIMARY KEY
                        name TEXT NOT NULL,
                        fee INTEGER NOT NULL,
                        valid TSTZRANGE NOT NULL DEFAULT TSTZRANGE(NOW(), NULL));
```

id	name	fee	valid
1	alice	10	[2019-01-01, NULL)
2	bob	20	[2019-01-01, NULL)

```
CREATE TABLE customer (id INTEGER NOT NULL, -- NO PRIMARY KEY
                        name TEXT NOT NULL,
                        fee INTEGER NOT NULL,
                        valid TSTZRANGE NOT NULL DEFAULT TSTZRANGE(NOW(), NULL));
```

id	name	fee	valid
1	alice	10	[2019-01-01, NULL)
2	bob	20	[2019-01-01, NULL)

```
-- insert
INSERT INTO customer (id, name, fee) VALUES (3, 'carol', 30);

-- delete
UPDATE customer SET valid = TSTZRANGE(LOWER(valid), NOW()) WHERE id = 2 and UPPER_INF(valid);

-- update
UPDATE customer SET valid = TSTZRANGE(LOWER(valid), NOW()) WHERE id = 1 and UPPER_INF(valid);
INSERT INTO customer (id, name, fee) VALUES (1, 'alice', 15);
```

```
CREATE TABLE customer (id INTEGER NOT NULL, -- NO PRIMARY KEY
                        name TEXT NOT NULL,
                        fee INTEGER NOT NULL,
                        valid TSTZRANGE NOT NULL DEFAULT TSTZRANGE(NOW(), NULL));
```

id	name	fee	valid
1	alice	10	[2019-01-01, NULL)
2	bob	20	[2019-01-01, NULL)

```
-- insert
INSERT INTO customer (id, name, fee) VALUES (3, 'carol', 30);

-- delete
UPDATE customer SET valid = TSTZRANGE(LOWER(valid), NOW()) WHERE id = 2 and UPPER_INF(valid);

-- update
UPDATE customer SET valid = TSTZRANGE(LOWER(valid), NOW()) WHERE id = 1 and UPPER_INF(valid);
INSERT INTO customer (id, name, fee) VALUES (1, 'alice', 15);
```

id	name	fee	valid
1	alice	10	[2019-01-01, 2019-03-12)
1	alice	15	[2019-03-12, NULL)
2	bob	20	[2019-01-01, 2019-03-12)
3	carol	30	[2019-03-12, NULL)

```
CREATE EXTENSION btree_gist;
```

```
-- Generalized Search Tree
```

```
CREATE EXTENSION btree_gist;
```

```
-- Generalized Search Tree
```

```
CREATE TABLE customer (  
  id INTEGER NOT NULL,  
  name TEXT NOT NULL,  
  fee INTEGER NOT NULL,  
  valid TSTZRANGE NOT NULL,  
  EXCLUDE USING GIST (id WITH =,  
                      valid WITH &&)  
);
```

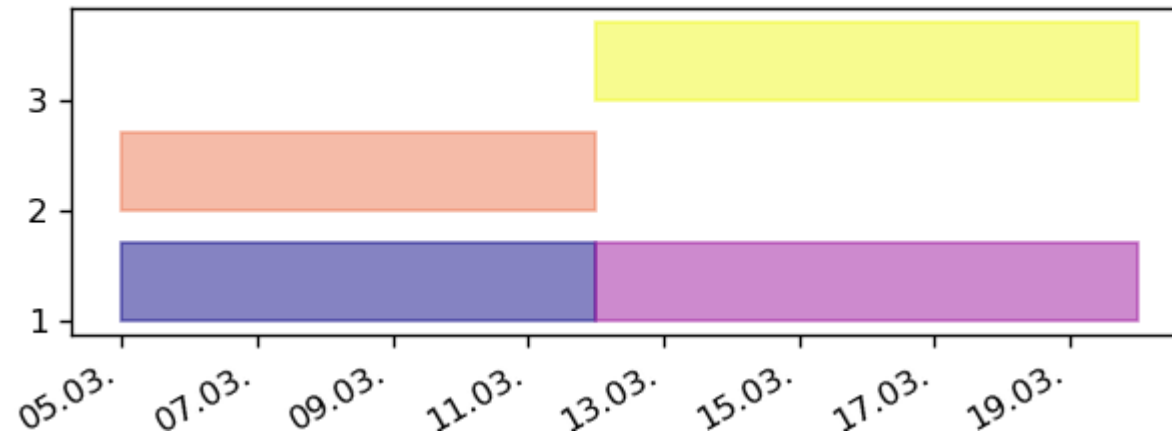
id	name	fee	valid
1	alice	10	[2019-01-01, 2019-03-12)
1	alice	15	[2019-03-12, NULL)
2	bob	20	[2019-01-01, 2019-03-12)
3	carol	30	[2019-03-12, NULL)

```
CREATE EXTENSION btree_gist;
```

```
-- Generalized Search Tree
```

```
CREATE TABLE customer (  
  id INTEGER NOT NULL,  
  name TEXT NOT NULL,  
  fee INTEGER NOT NULL,  
  valid TSTZRANGE NOT NULL,  
  EXCLUDE USING GIST (id WITH =,  
                      valid WITH &&)  
);
```

id	name	fee	valid
1	alice	10	[2019-01-01, 2019-03-12)
1	alice	15	[2019-03-12, NULL)
2	bob	20	[2019-01-01, 2019-03-12)
3	carol	30	[2019-03-12, NULL)



Let's get that PRIMARY KEY back!

```
CREATE TABLE customer (  
  id SERIAL PRIMARY KEY,  
  name TEXT UNIQUE);
```

```
CREATE TABLE customer_rev (  
  id INTEGER REFERENCES customer(id),  
  fee INTEGER NOT NULL,  
  valid TSTZRANGE NOT NULL,  
  EXCLUDE USING GIST (id WITH =,  
                      valid WITH &&));
```

Let's get that PRIMARY KEY back!

```
CREATE TABLE customer (  
  id SERIAL PRIMARY KEY,  
  name TEXT UNIQUE);
```

```
CREATE TABLE customer_rev (  
  id INTEGER REFERENCES customer(id),  
  fee INTEGER NOT NULL,  
  valid TSTZRANGE NOT NULL,  
  EXCLUDE USING GIST (id WITH =,  
                      valid WITH &&));
```

id	name
1	alice
2	bob
3	carol

id	fee	valid
1	10	[2019-01-01, 2019-03-12)
1	15	[2019-03-12, NULL)
2	20	[2019-01-01, 2019-03-12)
3	30	[2019-03-12, NULL)

Let's get that PRIMARY KEY back!

```
CREATE TABLE customer (  
  id SERIAL PRIMARY KEY,  
  name TEXT UNIQUE);
```

```
CREATE TABLE customer_rev (  
  id INTEGER REFERENCES customer(id),  
  fee INTEGER NOT NULL,  
  valid TSTZRANGE NOT NULL,  
  EXCLUDE USING GIST (id WITH =,  
                      valid WITH &&));
```

id	name
1	alice
2	bob
3	carol

id	fee	valid
1	10	[2019-01-01, 2019-03-12)
1	15	[2019-03-12, NULL)
2	20	[2019-01-01, 2019-03-12)
3	30	[2019-03-12, NULL)

```
SELECT username  
FROM customer JOIN customer_rev USING (id)  
WHERE UPPER_INF(valid) -- WHERE valid @> '2019-01-15'
```

Let's get that PRIMARY KEY back!

```
CREATE TABLE customer (  
  id SERIAL PRIMARY KEY,  
  name TEXT UNIQUE);
```

```
CREATE TABLE customer_rev (  
  id INTEGER REFERENCES customer(id),  
  fee INTEGER NOT NULL,  
  valid TSTZRANGE NOT NULL,  
  EXCLUDE USING GIST (id WITH =,  
                      valid WITH &&));
```

id	name
1	alice
2	bob
3	carol

id	fee	valid
1	10	[2019-01-01, 2019-03-12)
1	15	[2019-03-12, NULL)
2	20	[2019-01-01, 2019-03-12)
3	30	[2019-03-12, NULL)

```
SELECT username  
FROM customer JOIN customer_rev USING (id)  
WHERE UPPER_INF(valid) -- WHERE valid @> '2019-01-15'
```

Let's add another time dimension!

Bitemporal database design

A method of storing data records to represent both the history of the reality and the history of updates to these records in the database.

Bitemporal database design

A method of storing data records to represent both the history of the reality and the history of updates to these records in the database.

1. When the fact described by the record was known to the world
2. When the record was known to our database

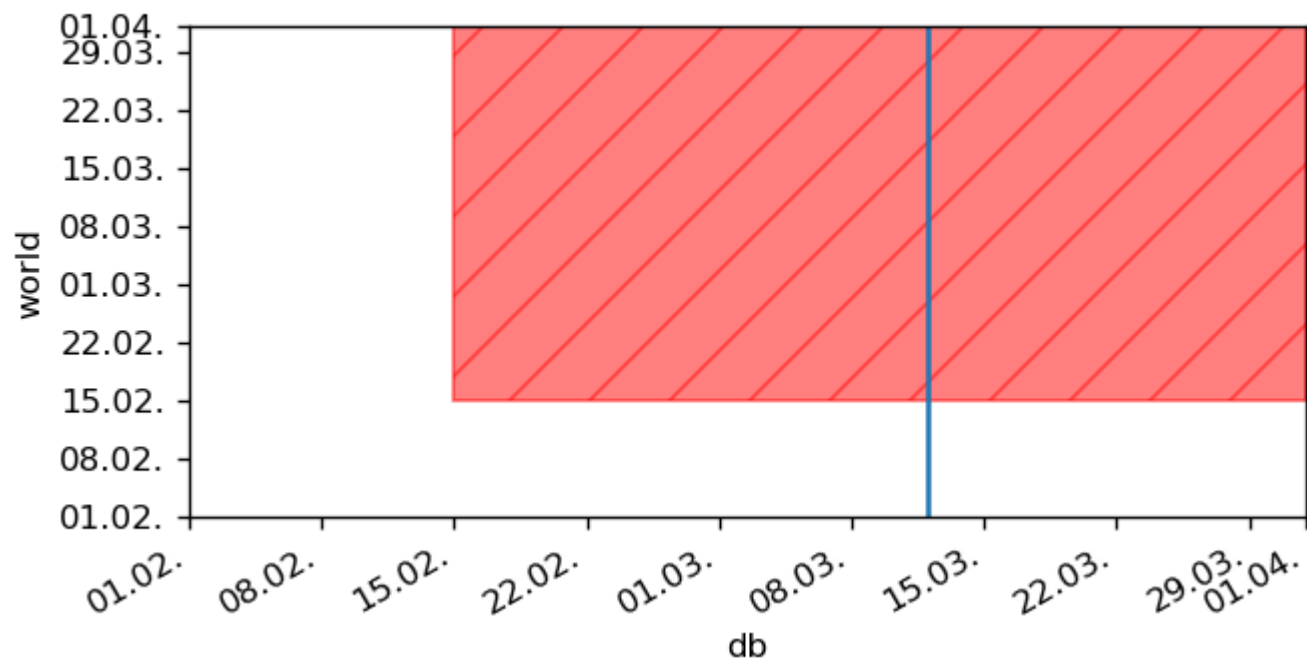
Bitemporal database design

A method of storing data records to represent both the history of the reality and the history of updates to these records in the database.

1. When the fact described by the record was known to the world
2. When the record was known to our database

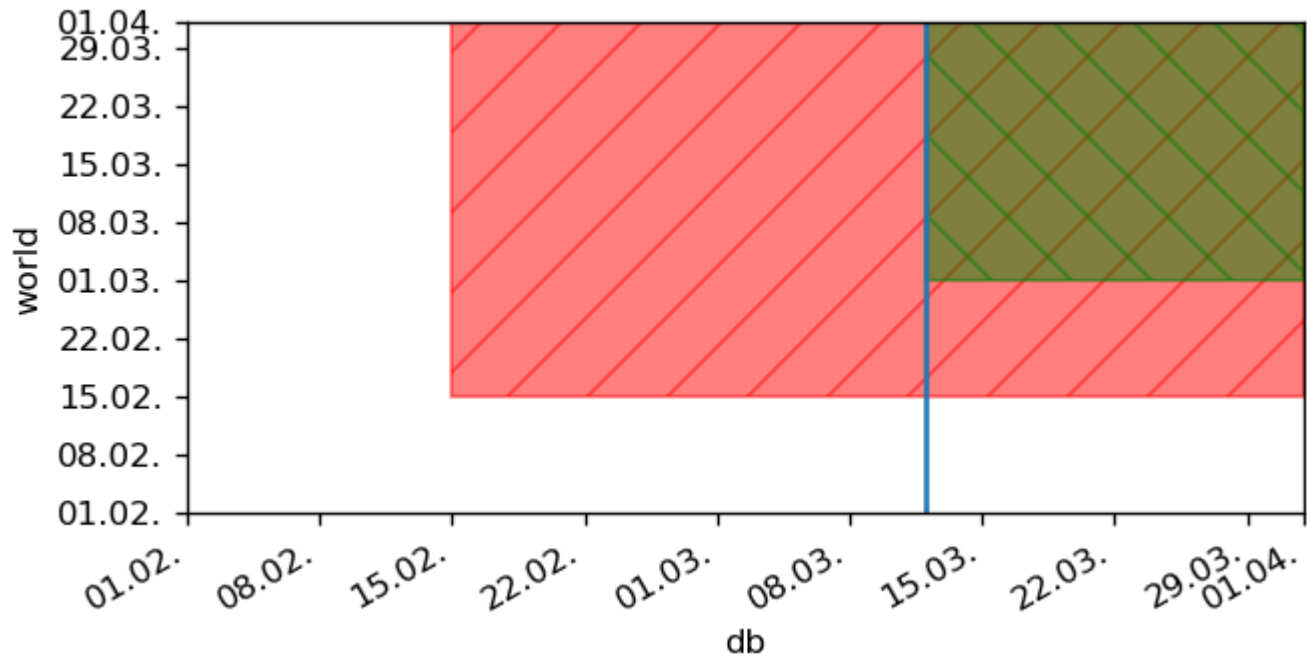
```
CREATE TABLE customer_rev (  
  id INTEGER REFERENCES customer(id),  
  fee INTEGER NOT NULL,  
  tsr_world TSTZRANGE NOT NULL,  
  tsr_db TSTZRANGE NOT NULL,  
  EXCLUDE USING GIST (id WITH =,  
                      tsr_world WITH &&,  
                      tsr_db WITH &&));
```

id	fee	tsr_world	tsr_db
1	10	[2019-02-15, NULL)	[2019-02-15, NULL)

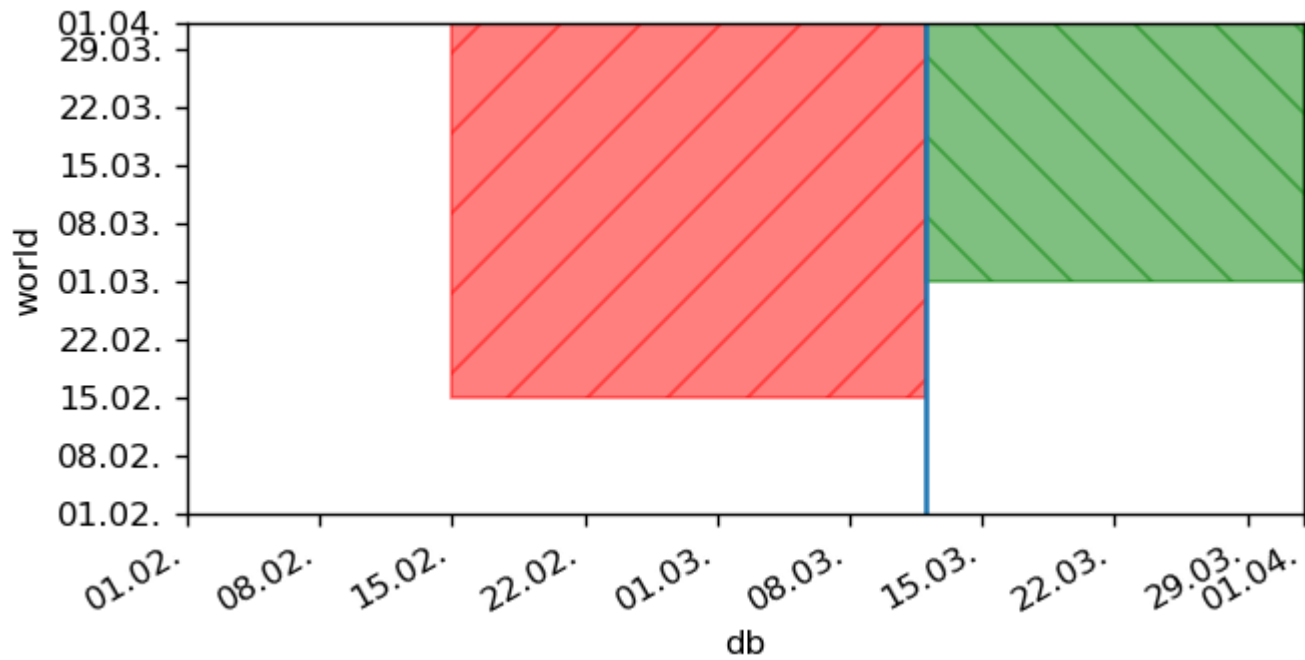


```
SELECT id, fee, tsr_world
FROM customer_rev
WHERE tsr_db @> '2018-03-12';
```

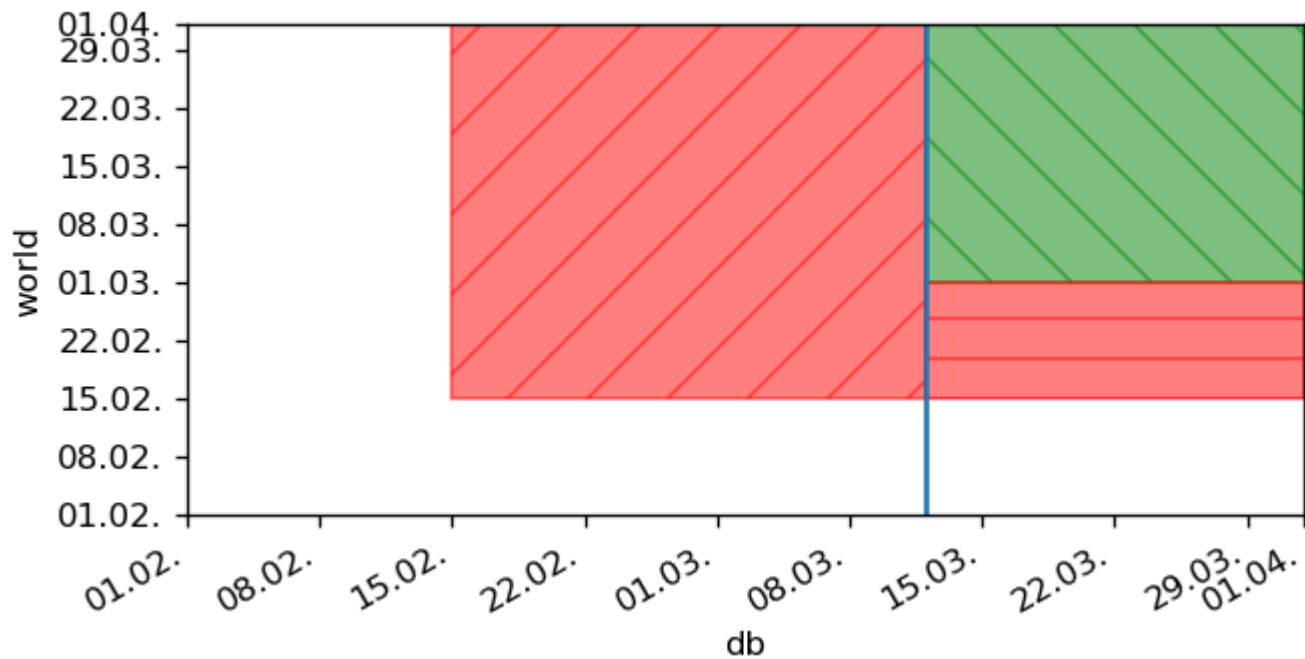

id	fee	tsr_world	tsr_db
1	10	[2019-02-15, NULL)	[2019-02-15, NULL)
1	15	[2019-03-01, NULL)	[2019-03-12, NULL)



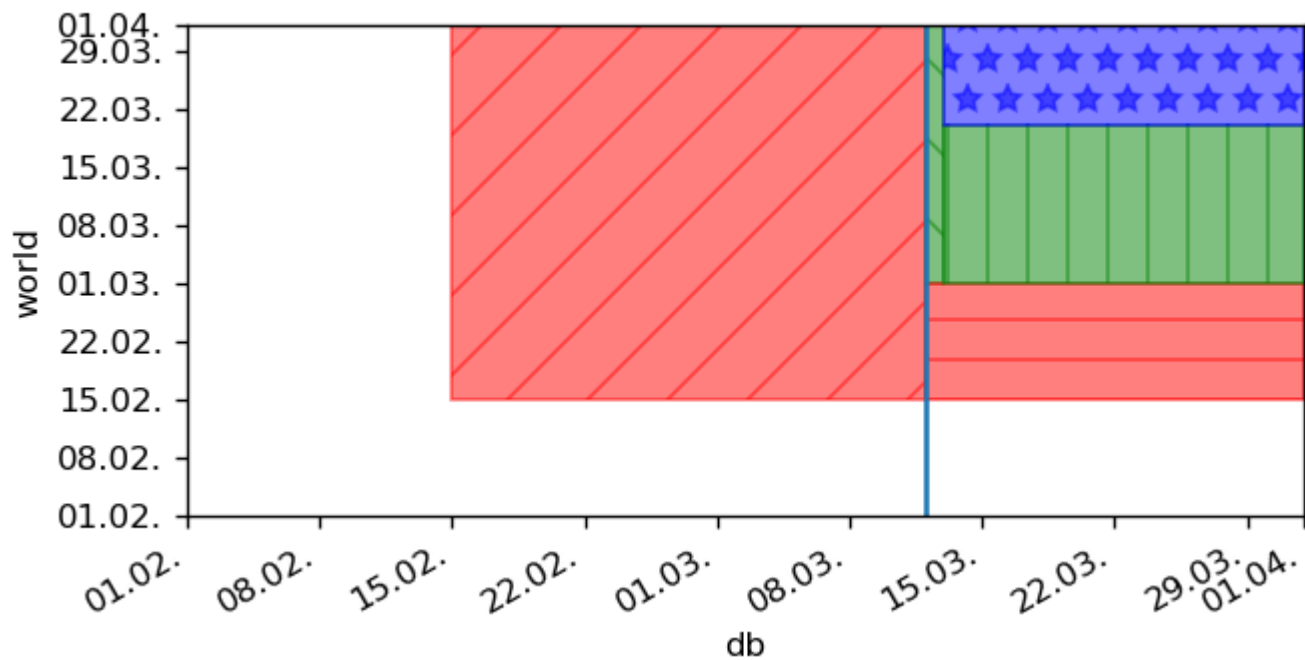
id	fee	tsr_world	tsr_db
1	10	[2019-02-15, NULL)	[2019-02-15, 2019-03-12)
1	15	[2019-03-01, NULL)	[2019-03-12, NULL)



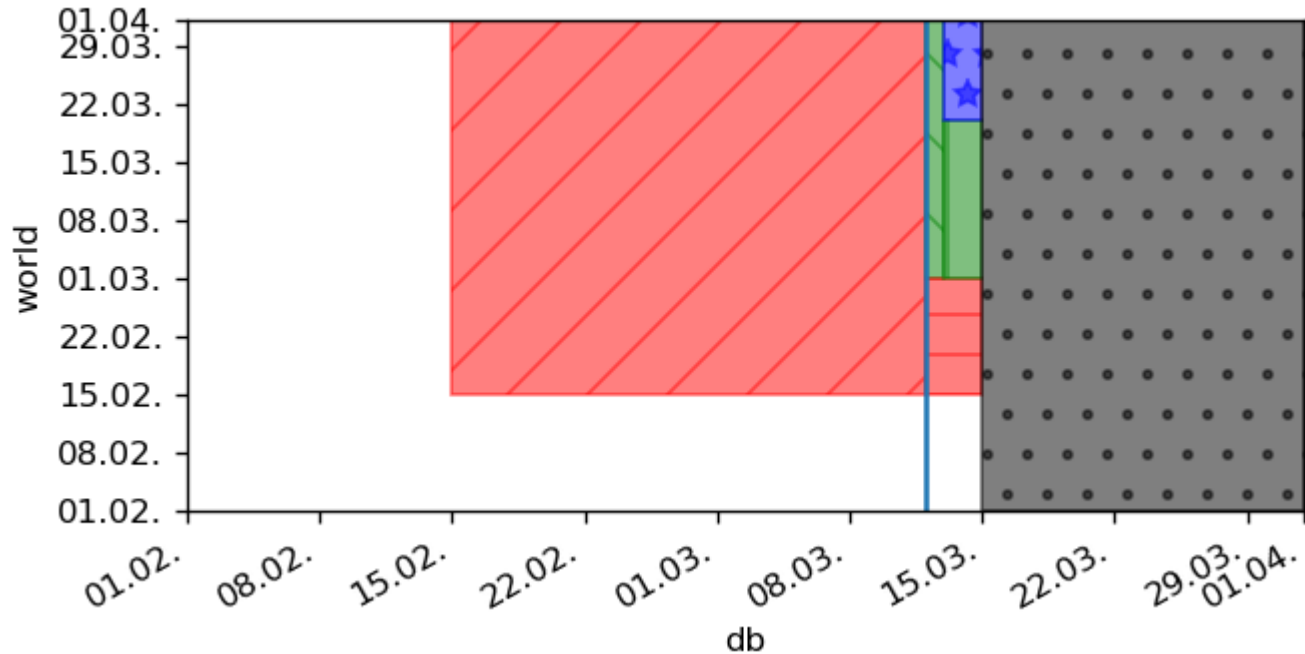
id	fee	tsr_world	tsr_db
1	10	[2019-02-15, NULL)	[2019-02-15, 2019-03-12)
1	10	[2019-02-15, 2019-03-01)	[2019-03-12, NULL)
1	15	[2019-03-01, NULL)	[2019-03-12, NULL)

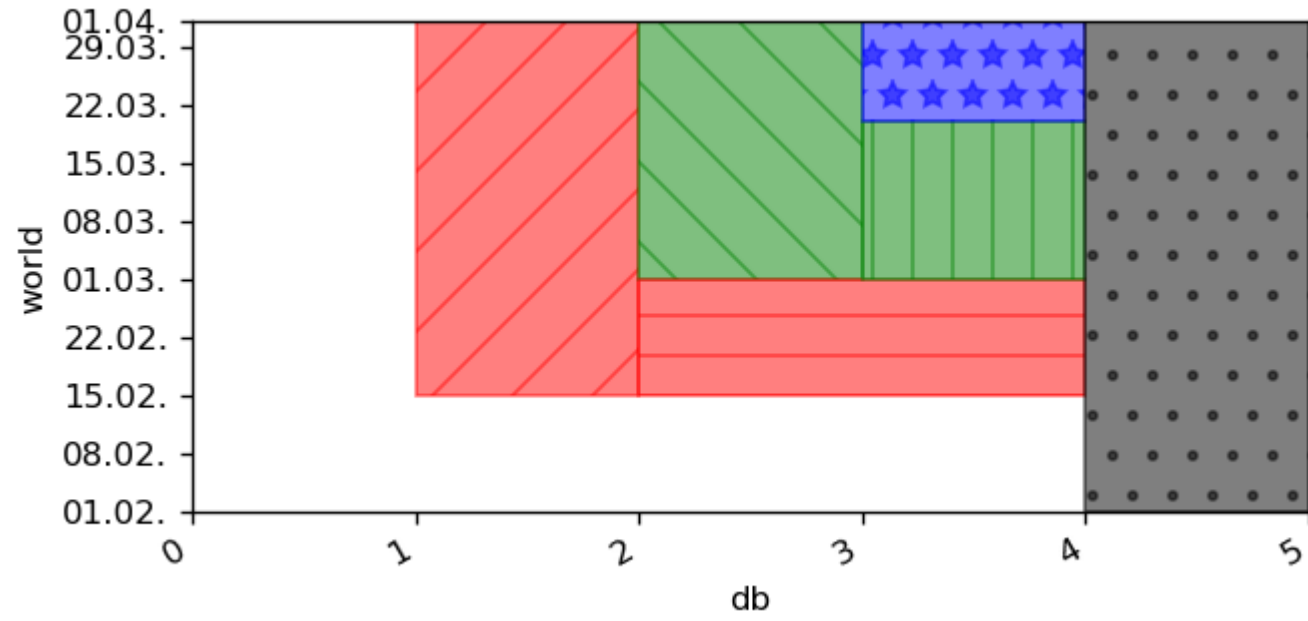


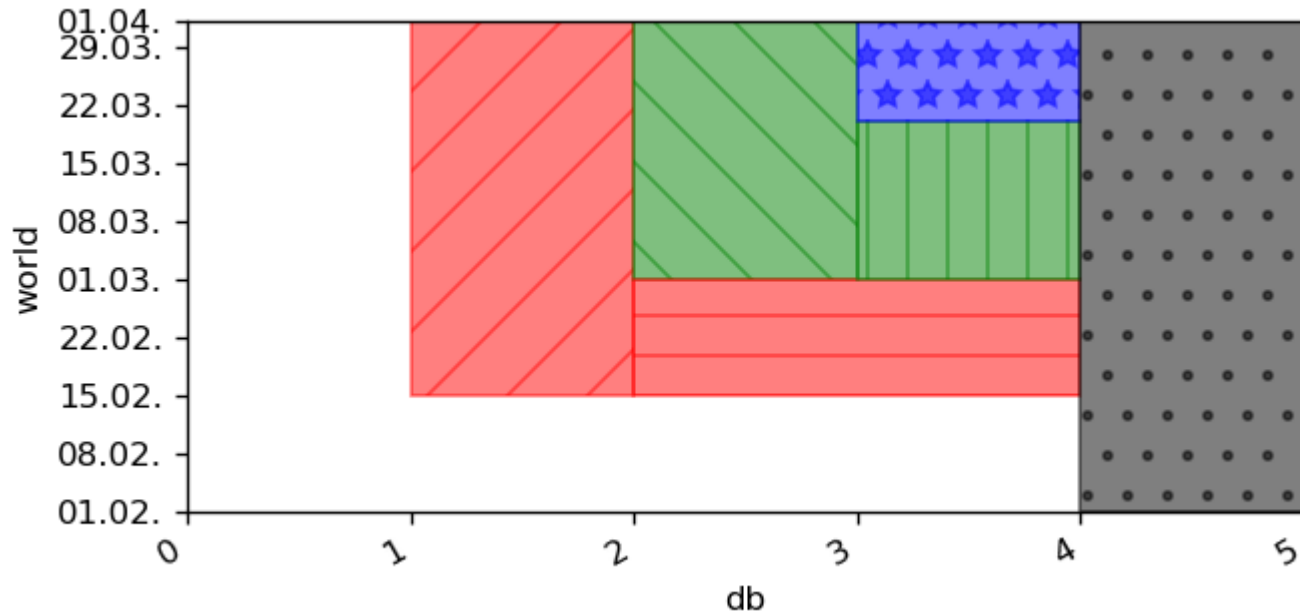
id	fee	tsr_world	tsr_db
1	10	[2019-02-15, NULL)	[2019-02-15, 2019-03-12)
1	10	[2019-02-15, 2019-03-01)	[2019-03-12, NULL)
1	15	[2019-03-01, NULL)	[2019-03-12, 2019-02-13)
1	15	[2019-03-01, 2019-03-20)	[2019-03-13, NULL)
1	18	[2019-03-20, NULL)	[2019-03-13, NULL)



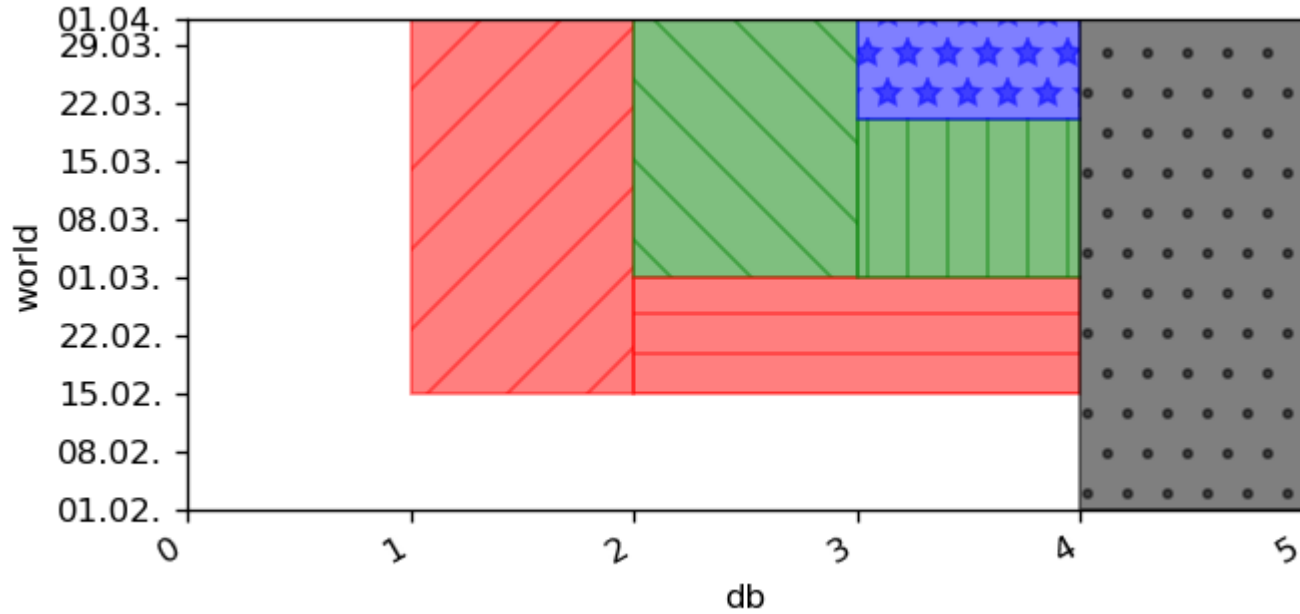
id	fee	tsr_world	tsr_db
1	10	[2019-02-15, NULL)	[2019-02-15, 2019-03-12)
1	10	[2019-02-15, 2019-03-01)	[2019-03-12, 2019-03-15)
1	15	[2019-03-01, NULL)	[2019-03-12, 2019-03-13)
1	15	[2019-03-01, 2019-03-20)	[2019-03-13, 2019-03-15)
1	18	[2019-03-20, NULL)	[2019-03-13, 2019-03-15)
1	0	[NULL, NULL)	[2019-03-15, NULL)







```
CREATE TABLE revision (
  revision_id INTEGER PRIMARY KEY,
  tsr_db TSTZRANGE DEFAULT TSTZRANGE(NOW(), NULL),
  desc TEXT NOT NULL UNIQUE,
  EXCLUDE USING GIST (tsr_db WITH &&));
```



```
CREATE TABLE revision (
  revision_id INTEGER PRIMARY KEY,
  tsr_db TSTZRANGE DEFAULT TSTZRANGE(NOW(), NULL),
  desc TEXT NOT NULL UNIQUE,
  EXCLUDE USING GIST (tsr_db WITH &&));
```

revision_id	tsr_db	desc
-------------	--------	------

0	[NULL, NULL)	In the beginning, the Earth was without form, and void...
---	--------------	---

customer_rev

id	fee	tsr_world	revs
1	10	[2019-02-15, NULL)	[1, 2)
1	10	[2019-02-15, 2019-03-01)	[2, 4)
1	15	[2019-03-01, NULL)	[2, 3)
1	15	[2019-03-01, 2019-03-20)	[3, 4)
1	18	[2019-03-20, NULL)	[3, 4)
1	0	[NULL, NULL)	[4, NULL)

revision

revision_id	tsr_db	desc
0	[NULL, 2019-02-15)	In the beginning, the Earth was without form, and void..
1	[2019-02-15, 2019-03-12)	Start at 10
2	[2019-03-12, 2019-03-13)	Increase to 15
3	[2019-03-13, 2019-03-15)	Increase to 18
4	[2019-03-15, NULL)	Give for free!

Writing into DB

- table revision
 - close previous `tsr_db`
 - insert a new one
- other tables
 - close previous revs
 - insert new rows

Writing into DB

- table revision
 - close previous tsr_db
 - insert a new one
- other tables
 - close previous revs
 - insert new rows

```
with Revision(desc='update from #123') as rev:  
  rev.modify(...)  
  rev.modify(...)
```

```

class Revision:
    def __init__(self, desc):
        self.desc = desc
        self.revision_id = None
        self.todos = []
        self.conn = psycopg.connect('postgres://USER@HOST:PORT/DBNAME')

    def __enter__(self):
        with self.conn.cursor() as cur:
            cur.execute("""UPDATE revision SET tsr_db = TSTZRANGE(LOWER(tsr_db), NOW())
                           WHERE UPPER_INF(tsr_db) RETURNING revision_id""")
            self.revision_id = next(cur)[0] + 1 # raise StopIteration?
            cur.execute("""INSERT INTO revision (revision_id, desc) VALUES (%s, %s)""",
                        (self.revision_id, self.desc))

        return self

    def __exit__(self, exc_type, exc_value, exc_traceback):
        if exc_type is None and self.todos:
            while True:
                confirm = input(f'{len(self.todos)} todos. commit? (yes/no)')
                if confirm == 'yes':
                    self.conn.commit()
                    return
                elif confirm == 'no':
                    break
            self.conn.rollback()

    def modify(self, ...):
        ...

```

Reading from DB

- consistent at any time

```
SELECT MAX(revision_id)
FROM revision;
```

```
SELECT ...
FROM customer_rev
WHERE revs @> %(revision_id)s
AND ...
```

Reading from DB

- consistent at any time

```
SELECT MAX(revision_id)
FROM revision;
```

```
SELECT ...
FROM customer_rev
WHERE revs @> %(revision_id)s
AND ...
```

- direct SQL or service?

Reading from DB

- consistent at any time

```
SELECT MAX(revision_id)
FROM revision;
```

```
SELECT ...
FROM customer_rev
WHERE revs @> %(revision_id)s
AND ...
```

- direct SQL or service?
 - logging, usage statistics (upgrades?)
 - caching
 - updates in service?

Consistent serial reading

What is “now”?

```
today = datetime.datetime.utcnow()  
yesterday = (datetime.datetime.utcnow() - datetime.timedelta(1))
```


Consistent serial reading

What is “now”?

```
today = datetime.datetime.utcnow()  
yesterday = (datetime.datetime.utcnow() - datetime.timedelta(1))
```

```
now = datetime.datetime.utcnow()  
today = now  
yesterday = (now - datetime.timedelta(1))
```

Consistent serial reading

What is “now”?

```
today = datetime.datetime.utcnow()  
yesterday = (datetime.datetime.utcnow() - datetime.timedelta(1))
```

```
now = datetime.datetime.utcnow()  
today = now  
yesterday = (now - datetime.timedelta(1))
```

What revision are we looking at?

```
data = service.method_one(*args)  
data = service.method_two(*args)
```

Consistent serial reading

What is “now”?

```
today = datetime.datetime.utcnow()
yesterday = (datetime.datetime.utcnow() - datetime.timedelta(1))
```

```
now = datetime.datetime.utcnow()
today = now
yesterday = (now - datetime.timedelta(1))
```

What revision are we looking at?

```
data = service.method_one(*args)
data = service.method_two(*args)
```

```
revision_id, data = service.method_one(*args)
revision_id, data = service.method_one(*args, revision_date=datetime(2019, 2, 1, 13, ...))
revision_id, data = service.method_one(*args, revision_id=17)

_, data = service.method_two(*args, revision_id=revision_id)
```

Revisions like commits in VCS?

- single branch
- “test” branches through clones?

Lessons learned

Lessons learned

- INT4RANGE, TSTZRANGE, EXCLUDE USING GIST

Lessons learned

- INT4RANGE, TSTZRANGE, EXCLUDE USING GIST
- psycopg2 blocking transactions on modified rows

Lessons learned

- INT4RANGE, TSTZRANGE, EXCLUDE USING GIST
- psycopg2 blocking transactions on modified rows
- use context managers for “transactions” in code

Lessons learned

- INT4RANGE, TSTZRANGE, EXCLUDE USING GIST
- psycopg2 blocking transactions on modified rows
- use context managers for “transactions” in code
- read external sources (time, DB, ...) independently from your runtime

Lessons learned

- INT4RANGE, TSTZRANGE, EXCLUDE USING GIST
- psycopg2 blocking transactions on modified rows
- use context managers for “transactions” in code
- read external sources (time, DB, ...) independently from your runtime
- Python3, psycopg2 and PostgreSQL are cool

name	born	alive_in_1992
Godefroy de Montmirail	1079	FALSE
Jacquouille la Fripouille	NULL	FALSE
Frénégonde de Pouilles	1095	FALSE
Béatrice de Montmirail	1964	TRUE
Jacques-Henri Jacquart	1952	TRUE
Hubert de Montmirail	1960	TRUE

name	born	alive_in_1992
Godefroy de Montmirail	1079	FALSE
Jacquouille la Fripouille	NULL	FALSE
Frénégonde de Pouilles	1095	FALSE
Béatrice de Montmirail	1964	TRUE
Jacques-Henri Jacquart	1952	TRUE
Hubert de Montmirail	1960	TRUE

time_zone	utc_offset	observes_dst
Europe/Berlin	+01:00	TRUE
Europe/Paris	+01:00	TRUE

name	born	alive_in_1992
Godefroy de Montmirail	1079	FALSE
Jacquouille la Fripouille	NULL	FALSE
Frénégonde de Pouilles	1095	FALSE
Béatrice de Montmirail	1964	TRUE
Jacques-Henri Jacquart	1952	TRUE
Hubert de Montmirail	1960	TRUE

time_zone	utc_offset	observes_dst
Europe/Berlin	+01:00	TRUE
Europe/Paris	+01:00	TRUE

Miroslav Šedivý

 eumiro  eumiro  in šedivý