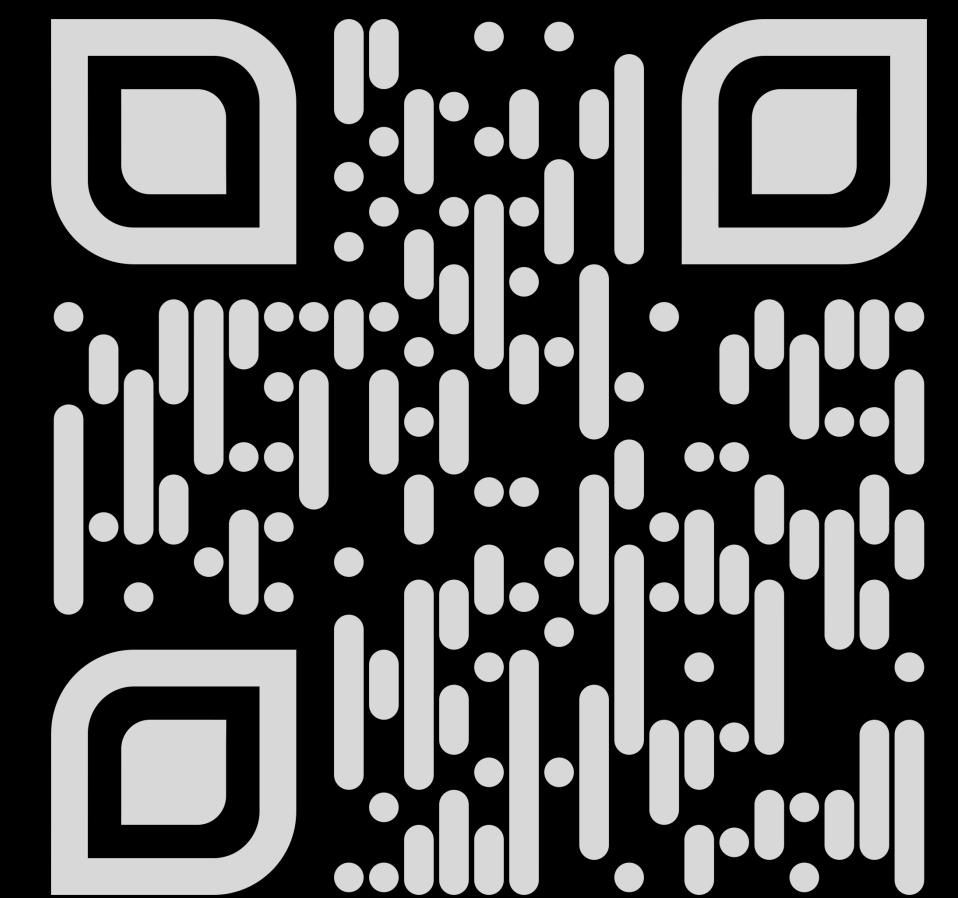




Пора тестировать миграции

CODE
SCORING



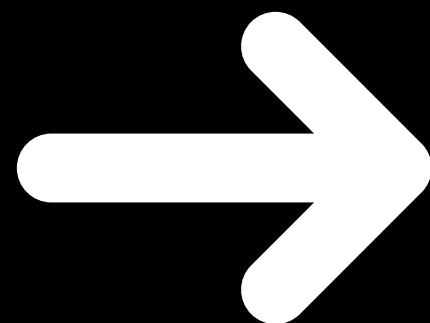
* Надежный * пайплайн



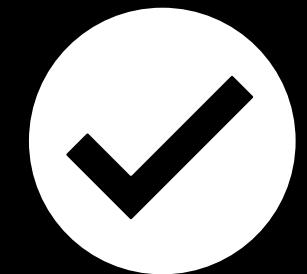
Надежный пайплайн



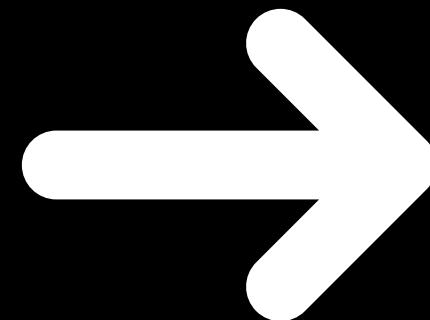
CR-1



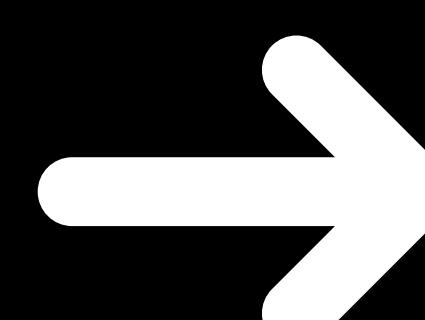
* Надежный * пайплайн



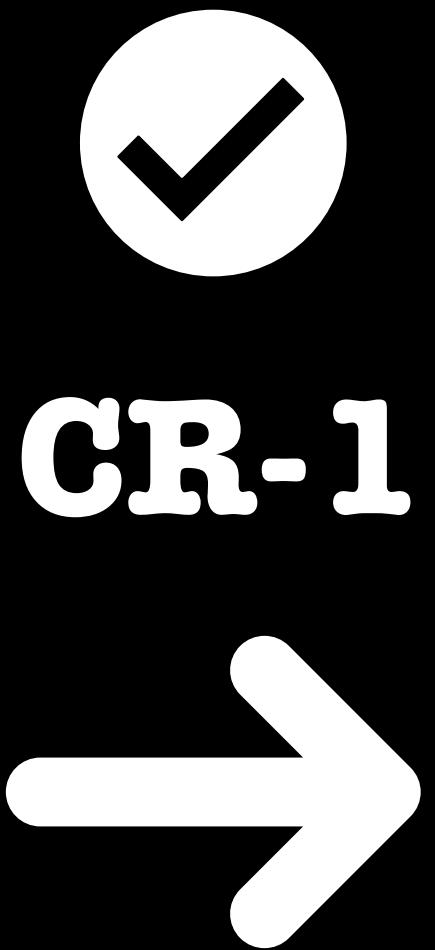
CR-1



CR-2



* Надежный * пайплайн



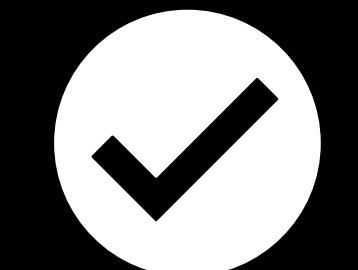
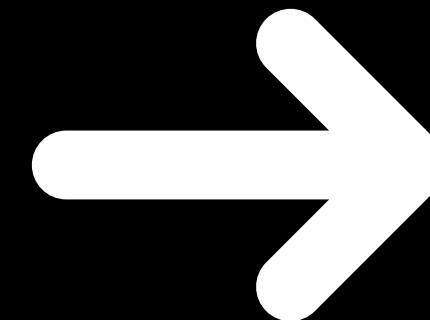
* Deployment *

A large white arrow pointing downwards from the sequence of images, with the text "* Deployment *" written along its path.

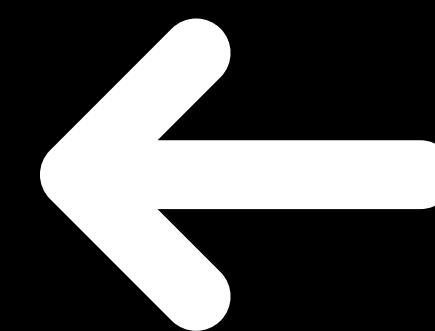
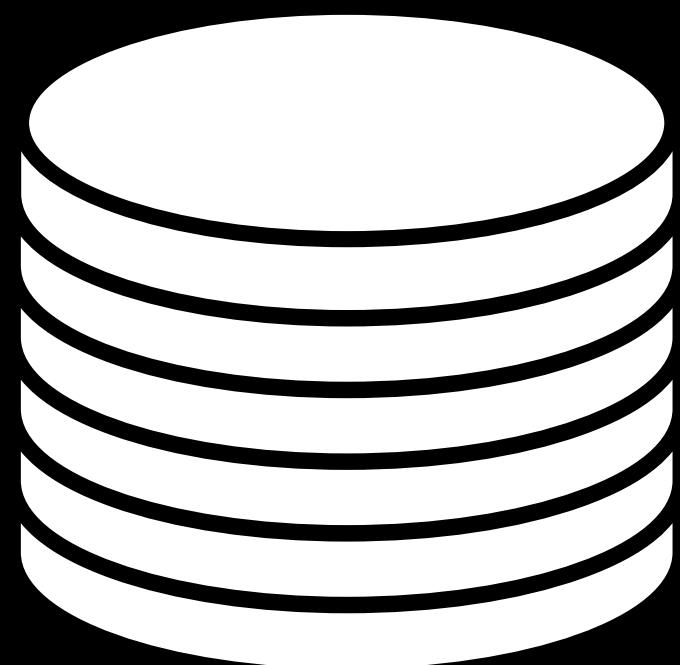
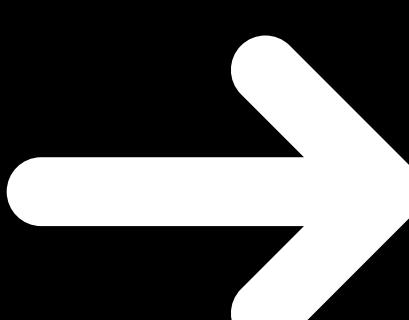
* Надежный * пайплайн



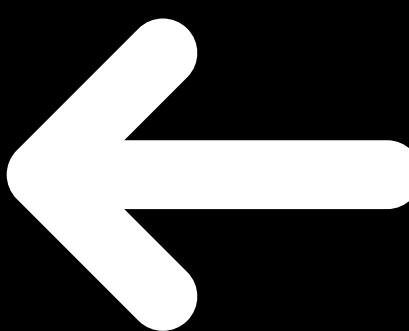
CR-1



CR-2



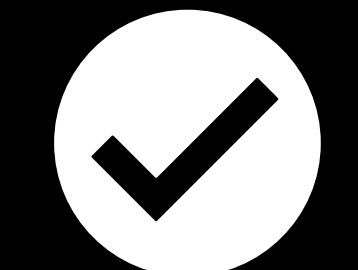
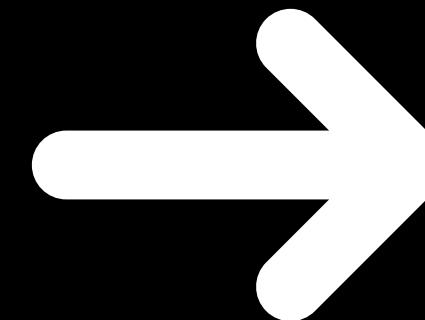
* Deployment *



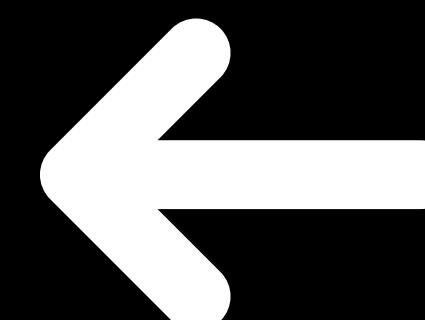
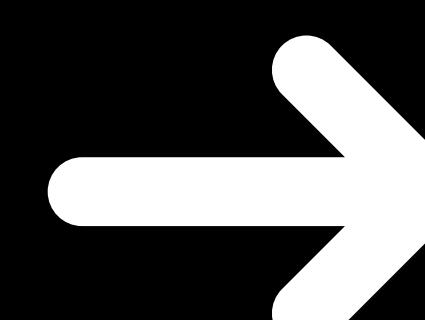
* Надежный * пайплайн



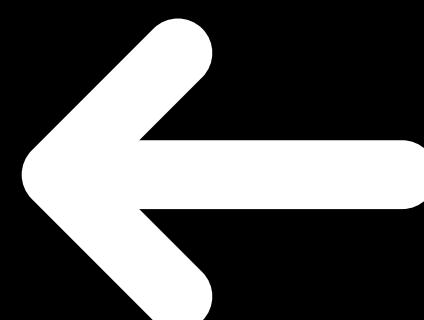
CR-1



CR-2



* Deployment *



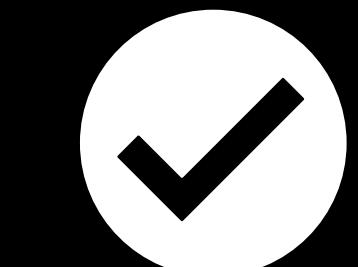


Дамы и господа

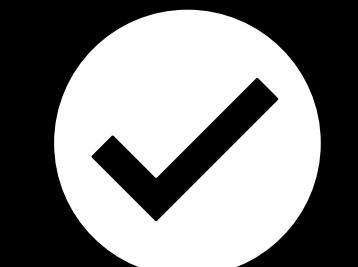
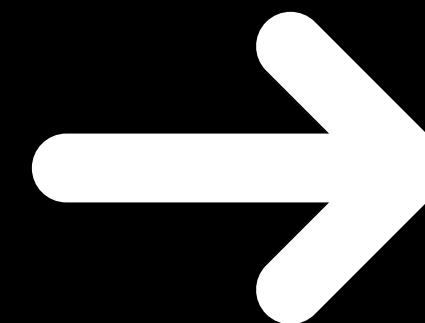
базы данных

больше нет

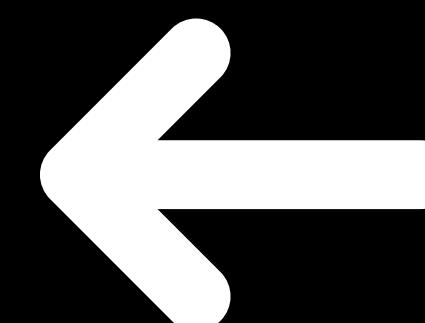
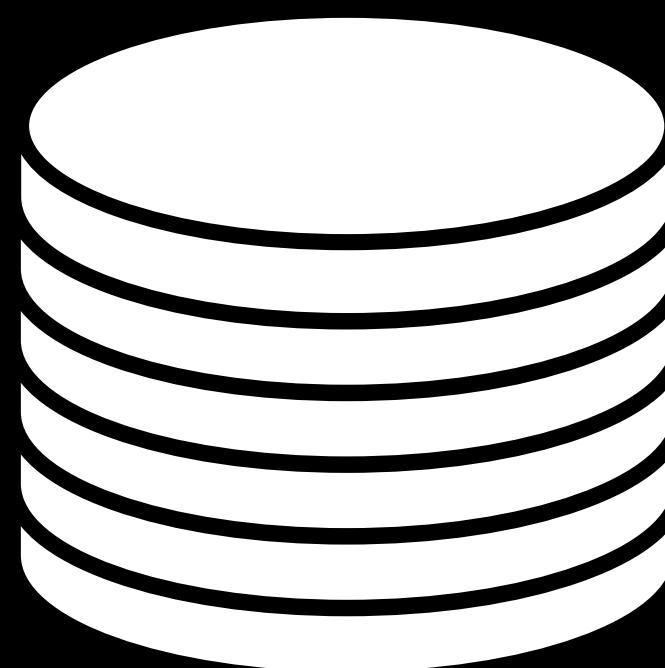
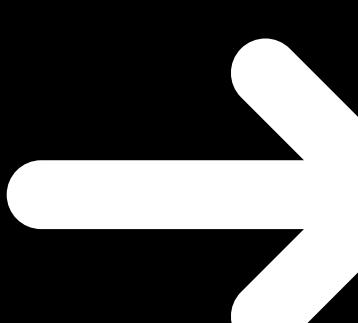
В чем проблема?



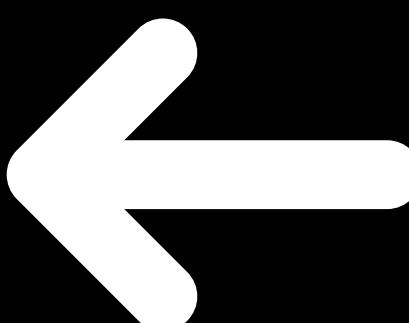
CR-1



CR-2



* Deployment *

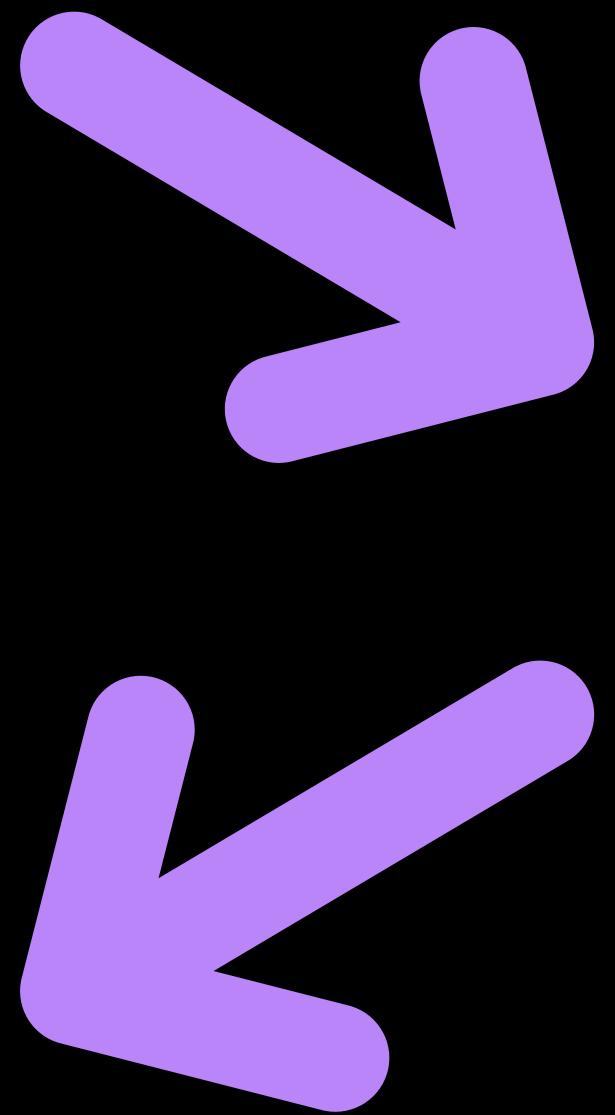


В чем проблема?



1. Больно ревьюить
2. Деградация пайплайна
3. Postmortem

* Надежный * пайплайн





Code of Conduct



```
1 -- migrate:up << Это upgrade-блок
2 SELECT * FROM *;
3
4 -- migrate:down << Это downgrade-блок
5 SELECT * FROM *;
6
```

📁 test_data
📁 valid/001
⌚ 001.sql
⌚ 002.sql
⌚ 003.sql
⌚ 004.sql
⌚ 005.sql
⌚ 006.sql
⌚ 007.sql
⌚ 008.sql



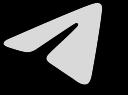
Пора тестировать
миграции



Давайте
автоматизируем
up-down



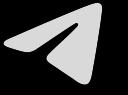
1	Column	Type	Constraints	Default
2				
3	<code>id</code>	SERIAL	PRIMARY KEY	
4	<code>username</code>	TEXT	NOT NULL	
5	<code>current_mood</code>	mood	NOT NULL	'ok'
6	<code>updated_at</code>	TEXT	NOT NULL	
7				



1	Column	Type	Constraints	Default
2				
3	<code>id</code>	SERIAL	PRIMARY KEY	
4	<code>username</code>	TEXT	NOT NULL	
5	<code>current_mood</code>	mood	NOT NULL	'ok'
6	<code>updated_at</code>	TEXT	NOT NULL	
7				



```
1 -- migrate:up
2 ALTER TYPE mood ADD VALUE 'happy' AFTER 'ok';
3
4 -- migrate:down
5
```



1	Column	Type	Constraints	Default
2				
3	<code>id</code>	SERIAL	PRIMARY KEY	
4	<code>username</code>	TEXT	NOT NULL	
5	<code>current_mood</code>	mood	NOT NULL	'ok'
6	<code>updated_at</code>	TEXT	NOT NULL	
7				

up
 down



```
1 -- migrate:up
2 ALTER TYPE mood ADD VALUE 'happy' AFTER 'ok';
3
4 -- migrate:down
5
```



1	Column	Type	Constraints	Default
2				
3	<code>id</code>	SERIAL	PRIMARY KEY	
4	<code>username</code>	TEXT	NOT NULL	
5	<code>current_mood</code>	mood	NOT NULL	'ok'
6	<code>updated_at</code>	TEXT	NOT NULL	
7				

✓ up
✓ down
✗ up



```
1 -- migrate:up
2 ALTER TYPE mood ADD VALUE 'happy' AFTER 'ok';
3
4 -- migrate:down
5
```



Выводы

1. up-down не проверяет
идемпотентность



Выводы

1. up-down не проверяет идемпотентность
2. down важнее up, так как отвечает за приведение к инварианту

Staircase-тест



1. накатываем все миграции

Staircase-тест



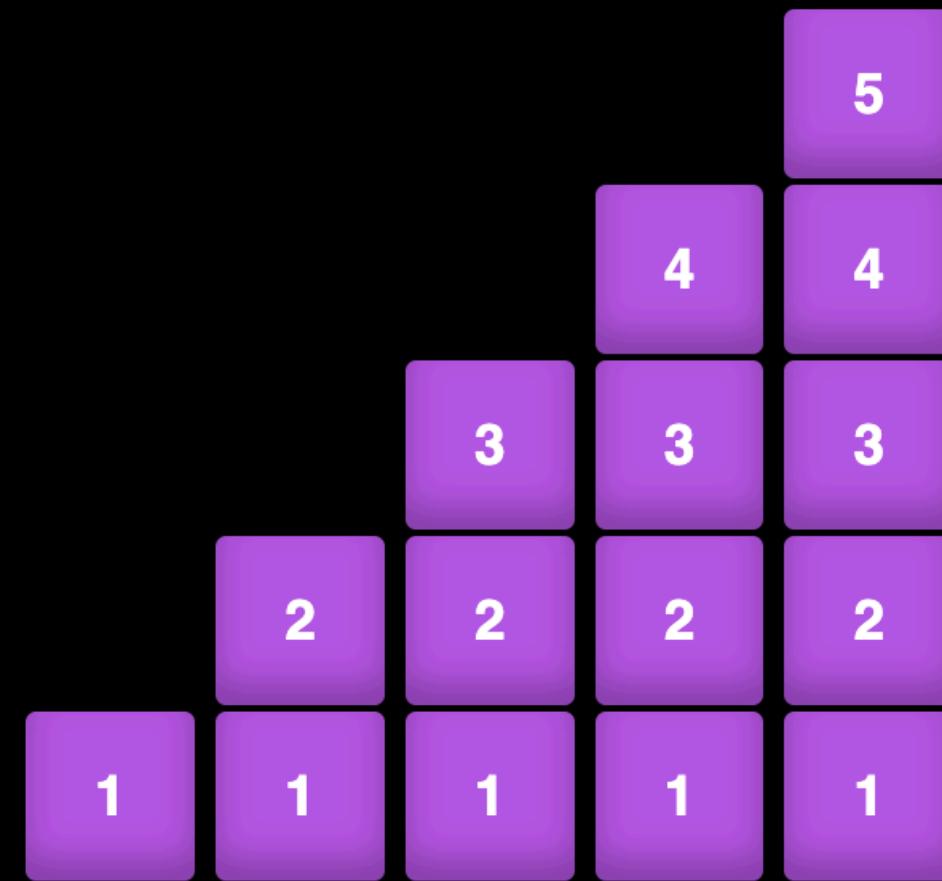
1. накатываем все миграции
2. итеративный `down > up > down`
до `depth: int = 0`



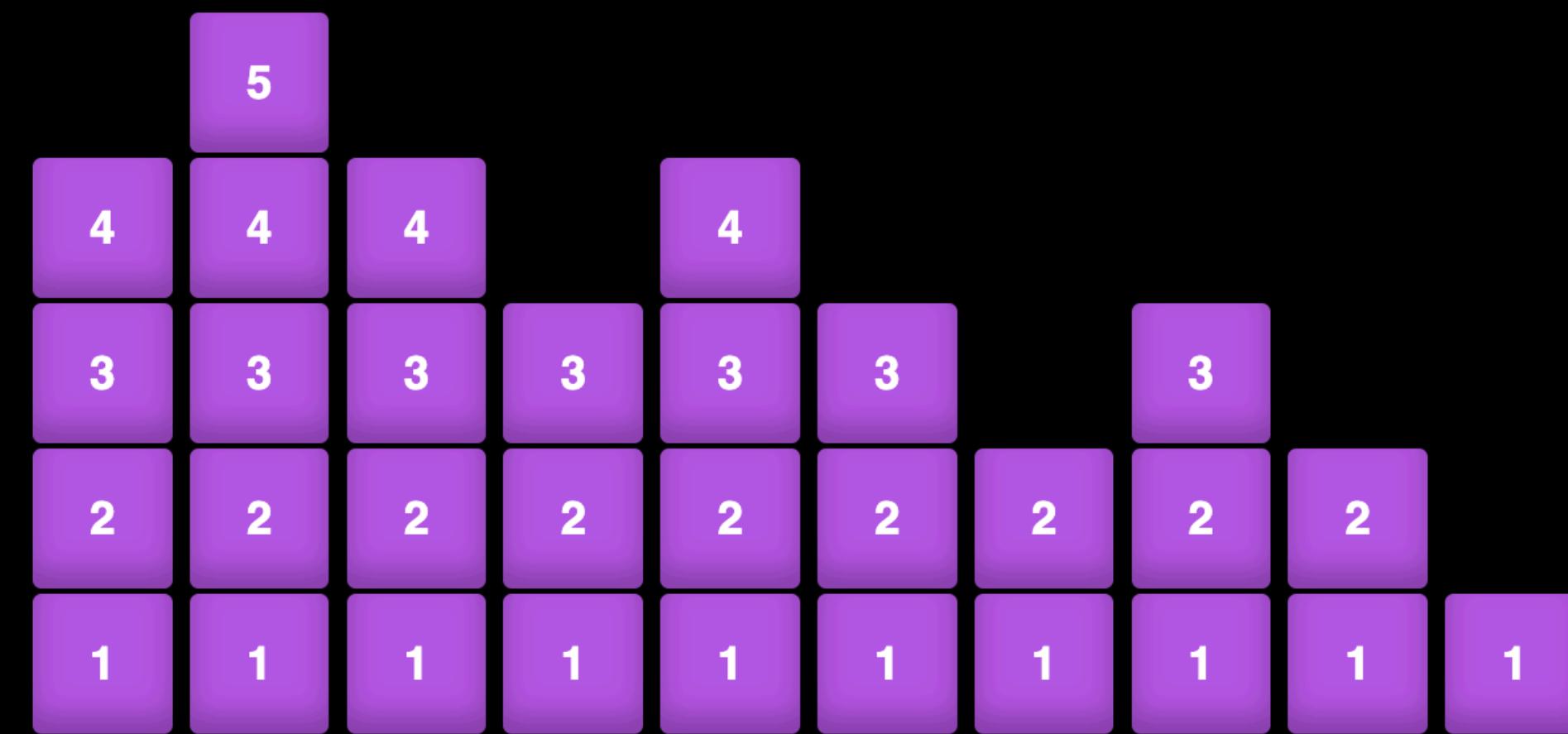
Staircase-тест

1. накатываем все миграции
2. итеративный `down > up > down`
до `depth: int = 0`
3. накатываем откаченное

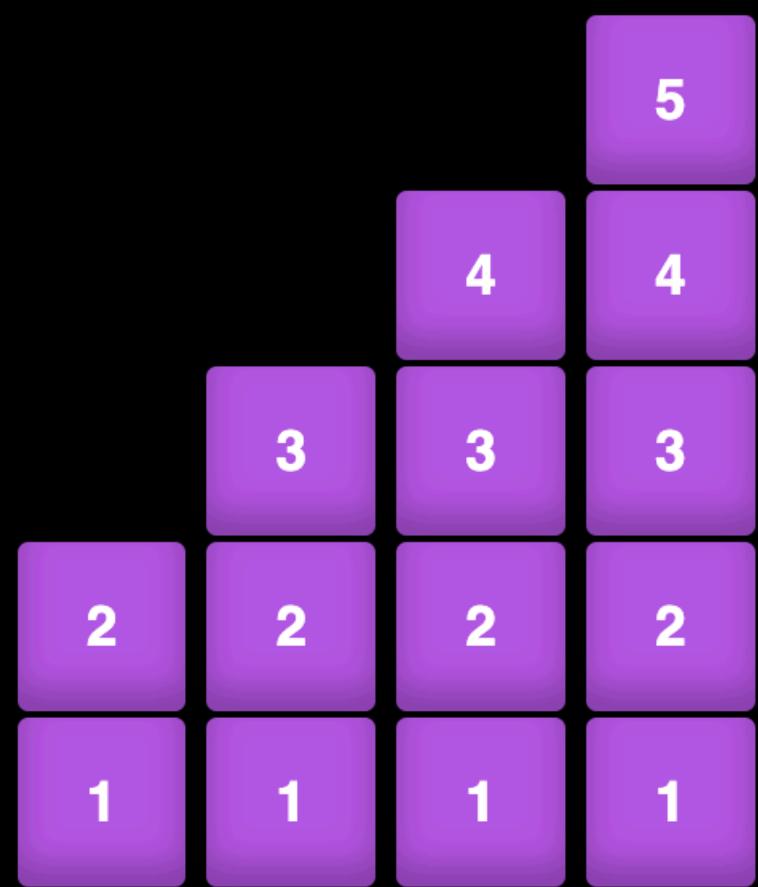
Staircase-Tect



1. Actualize



2. down > up > down



3. Re-
actualize

Есть проблема



```
1 -- migrate:up
2 DO $$
3 BEGIN
4     IF NOT EXISTS (
5         SELECT 1
6         FROM pg_enum
7         WHERE enumlabel = 'happy'
8         AND enumtypid = 'mood'::regtype
9     ) THEN
10        ALTER TYPE mood ADD VALUE 'happy' AFTER 'ok';
11    END IF;
12 END;
13 $$;
14
15 -- migrate:down
16
```

Миграция
идемпотентна, но
инвариант ломает



Снапшоты схем



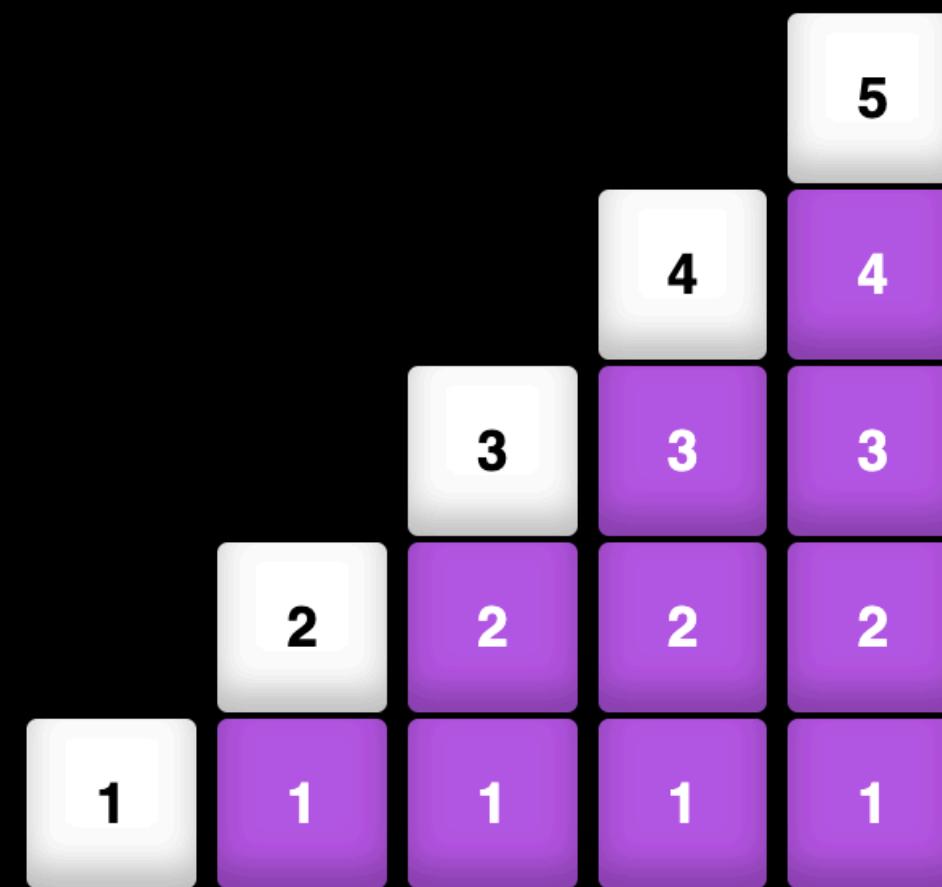
- snap1
- up
- down
- snap2

Снапшоты схем

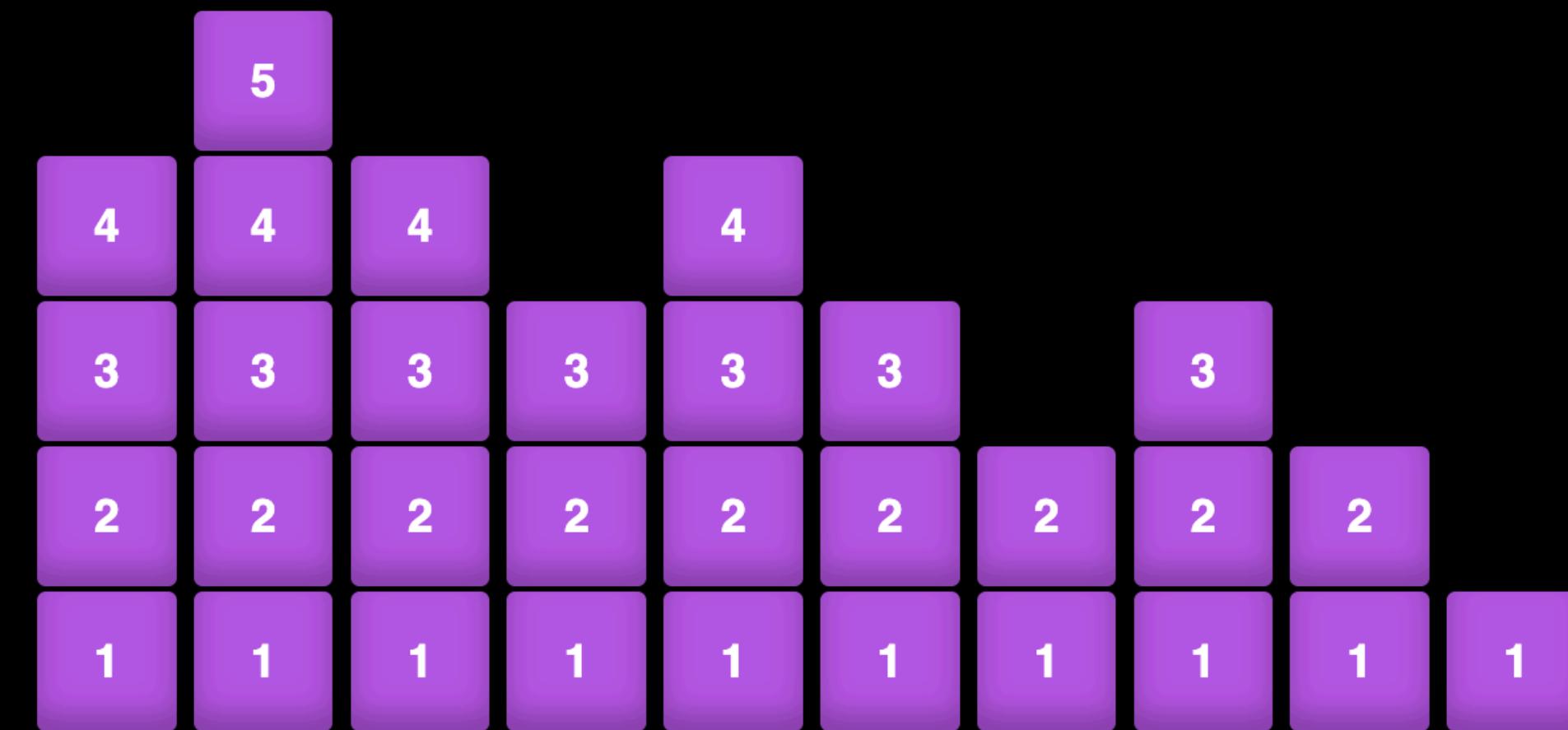


- snap1
- up
- down → snap1==snap2
- snap2

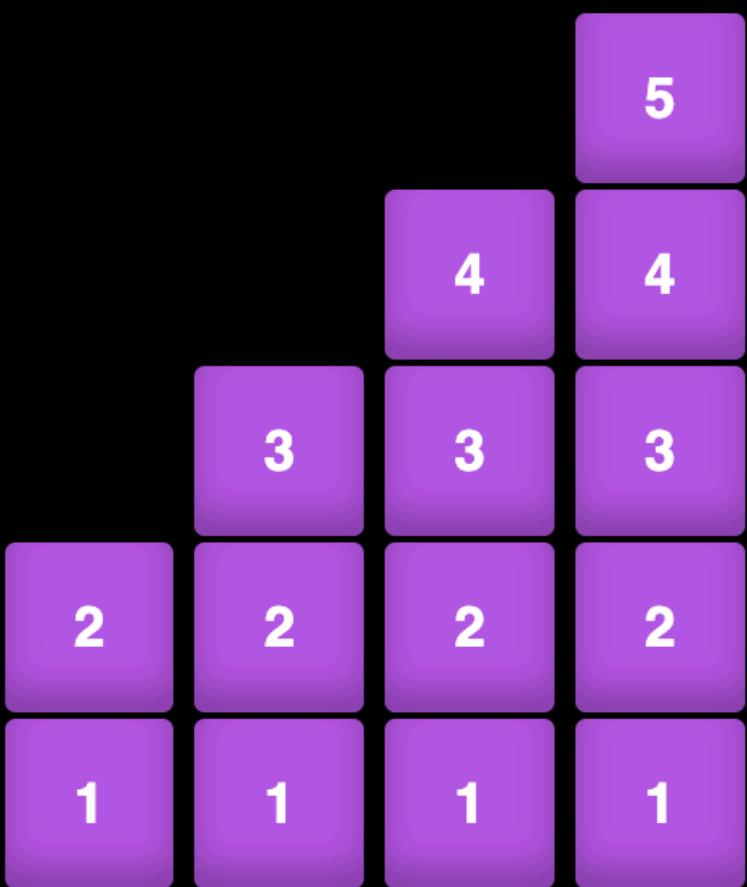
Staircase + Снапшоты



1. Actualize

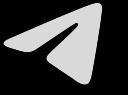


2. down > up > down



3. Re-actualize

белые – эталоны



1	Column	Type	Constraints	Default
2				
3	<code>id</code>	SERIAL	PRIMARY KEY	
4	<code>username</code>	TEXT	NOT NULL	
5	<code>current_mood</code>	mood	NOT NULL	'ok'
6	<code>updated_at</code>	TEXT	NOT NULL	
7				

up
 down
 сравнение схем



```
1 -- migrate:up
2 ALTER TYPE mood ADD VALUE 'happy' AFTER 'ok';
3
4 -- migrate:down
5
```

Как брать snapshot-ы

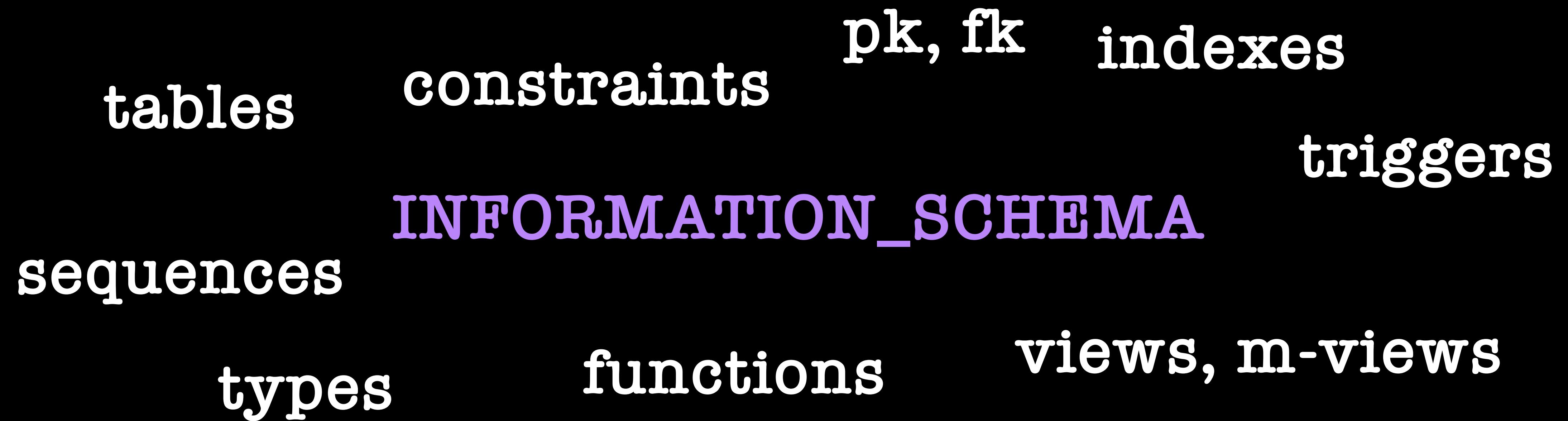
^

Как брать snapshot-ы

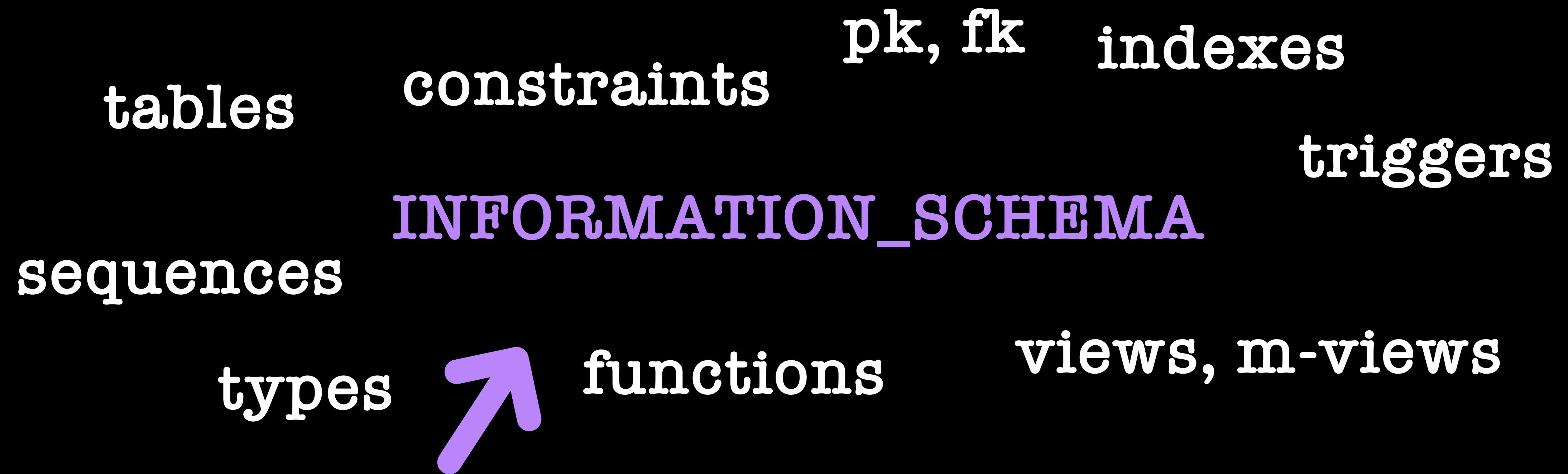
INFORMATION_SCHEMA

ISO/IEC 9075-11

Как брать snapshot-ы!



Как брать snapshot-ы!



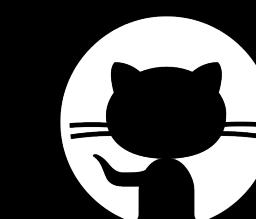
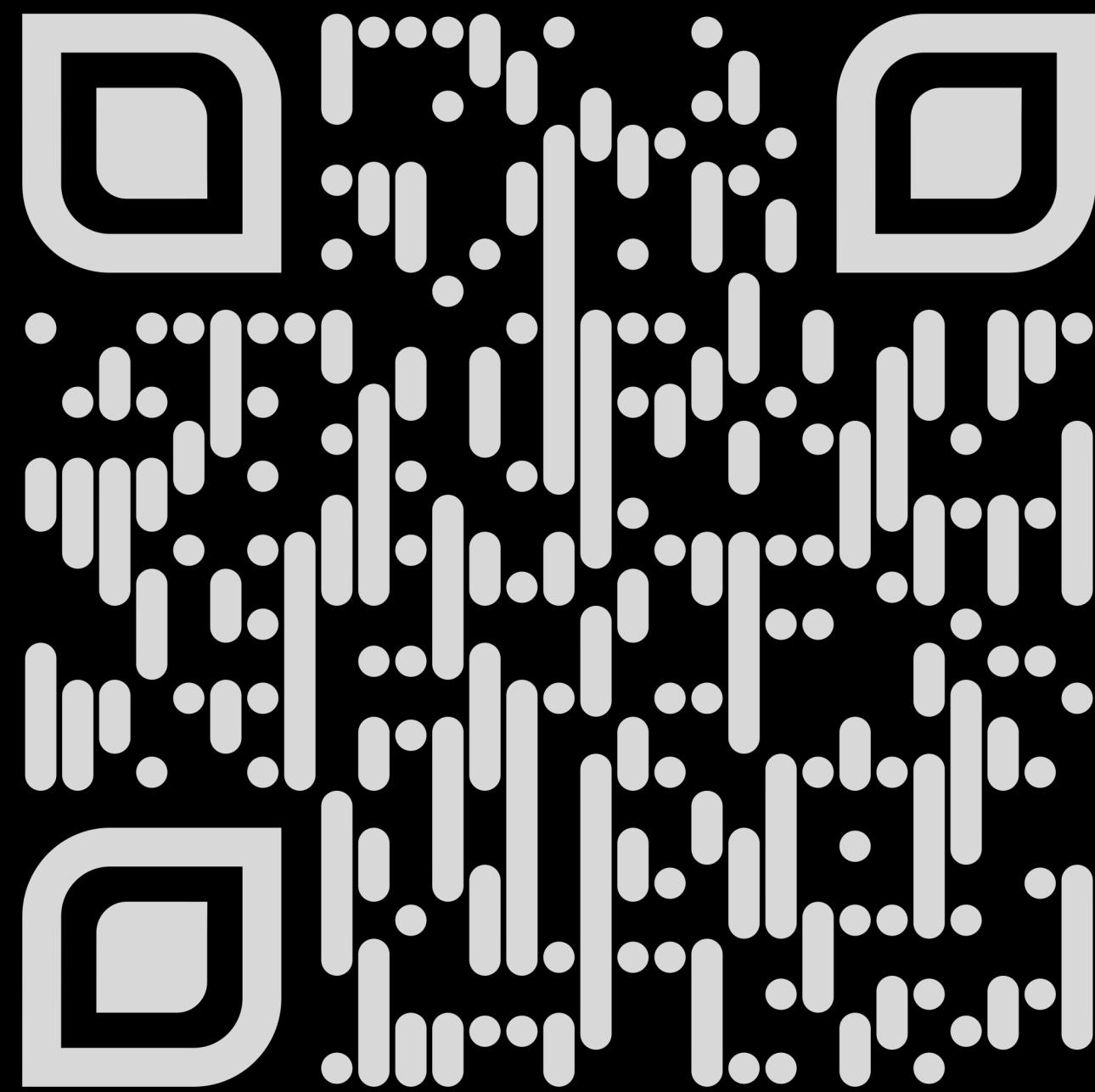
Python / Go & pg-driver

Сериализуем!

ISO/IEC 9075-11

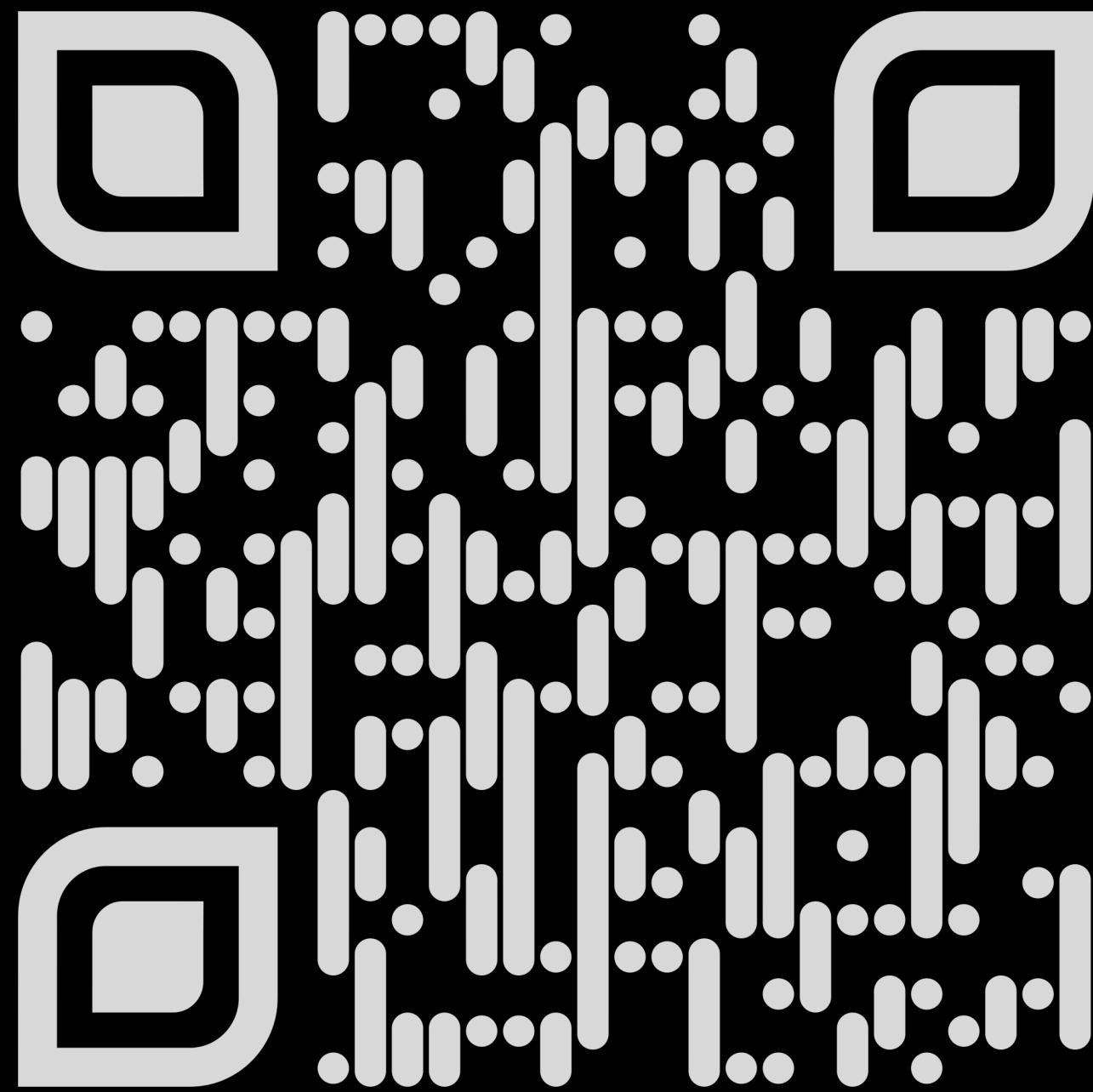
Как сделать staircase[↗]

Как сделать staircase



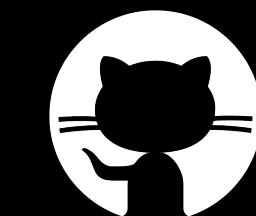
[realkarych/seqwall](https://github.com/realkarych/seqwall)

Как делать staircase



```
2025/05/03 01:07:42 Step 3 (up-down-up) completed successfully!
2025/05/03 01:07:42
🎉 Staircase test completed successfully!
```

```
@@ -376,8 +376,8 @@
      },
      {
        "ColumnName": "updated_at",
        "DataType": "timestamp with time zone",
        "UDTName": "timestamptz",
        "DataType": "timestamp without time zone",
        "UDTName": "timestamp",
        "DateTimePrecision": {
          "Int64": 6,
          "Valid": true
        }
      }
    }
  }
}
```



realkarych/seqwall

Промежуточные итоги

↗

1. идемпотентность \neq соблюдение
инварианта

Промежуточные итоги



1. идемпотентность \neq соблюдение инварианта
2. используй снапшот-тесты



Всё? Победа?

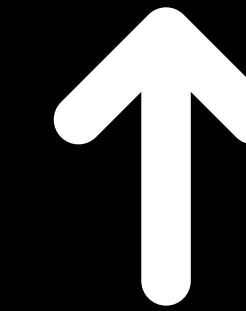


Что не так?



```
1 -- migrate:up
2 ALTER TABLE users
3   ALTER COLUMN updated_at TYPE timestamp
4   USING updated_at::timestamp;
5
6 -- migrate:down
7 ALTER TABLE users
8   ALTER COLUMN updated_at TYPE text
9   USING updated_at::text;
10
```

id	username	current_mood	updated_at
1	alice	happy	2025-03-14 12:00:00+03
2	bob	sad	2025-03-14 14:30:00+03
3	charlie	ok	2025-03-14 09:15:00+03

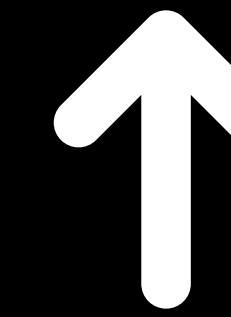


Что не так?



```
1 -- migrate:up
2 ALTER TABLE users
3   ALTER COLUMN updated_at TYPE timestamp
4   USING updated_at::timestamp;
5
6 -- migrate:down
7 ALTER TABLE users
8   ALTER COLUMN updated_at TYPE text
9   USING updated_at::text;
10
```

id	username	current_mood	updated_at
1	alice	happy	2025-03-14 12:00:00+03:00
2	bob	sad	2025-03-14 14:30:00+03:00
3	charlie	ok	2025-03-14 09:15:00+03:00



Что не так?



```
1 -- migrate:up
2 ALTER TABLE users
3   ALTER COLUMN updated_at TYPE timestamp
4   USING updated_at::timestamp;
5
6 -- migrate:down
7 ALTER TABLE users
8   ALTER COLUMN updated_at TYPE text
9   USING updated_at::text;
10
```

id	username	current_mood	updated_at
1	alice	happy	2025-03-14 12:00:00+03:00
2	bob	sad	2025-03-14 14:30:00+03:00
3	charlie	ok	2025-03-14 09:15:00+03:00

Пу-пу-пу,
timestampz...





**А что там с
даньми?**



Генерируем данные



Генерируем данные



Генерируем данные



- используем `faker` / `greenmask` / `hypothesis` / `regex`

Генерируем данные

↗

- используем **faker** / **greenmask** / **hypothesis** / **regex**
- не репрезентативно

Генерируем данные



- используем `faker` / `greenmask` /
`hypothesis` / `regex`
- не репрезентативно
- данные устаревают

Генерируем данные

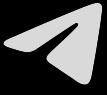


- используем faker / greenmask / hypothesis / regex
- не репрезентативно
- данные устаревают
- сильно с внешними фидами

Генерируем данные



- используем **faker / greenmask / hypothesis / regex**
- не репрезентативно
- данные устаревают
- сильно с внешними фидами
- невозможно на больших и связанных схемах



PROGRESS

user_id
UUID • PK
team_id
UUID • PK
task_code
TEXT • PK
status
TEXT

user_id + team_id

team_id + task_code

MEMBERSHIPS

user_id
UUID • PK
team_id
UUID • PK
joined_at
TIMESTAMP

TEAM_TASKS

team_id
UUID • PK
task_code
TEXT • PK
title
TEXT



Сэмплируем данные



- тащим данные с прода

Сэмплируем данные



- тащим данные с прода
- нужна инфраструктура

Сэмплируем данные



- тащим данные с прода
- нужна инфраструктура
- хорошо сэмплировать – сложно

Сэмплируем данные



- тащим данные с прода
- нужна инфраструктура
- хорошо сэмплировать – сложно
- скорость тестов VS объем данных
VS актуальность данных

Сэмплируем данные



- тащим данные с прода
- нужна инфраструктура
- хорошо сэмплировать – сложно
- скорость тестов VS объем данных
VS актуальность данных
- используем greenmask / свой велосипед

Генерация VS Сэмплы



Генерация VS Сэмпль¹

- Генерируем, если:
 - нет возможности сэмплировать
 - мало схем, слабые связи
 - данные семантически-очевидны (хеши, почты, имена, id...)

Генерация VS Сэмпльы

- Сэмплируем, если:
 - много схем
 - сложные связи, составные ключи
 - внешние фиды и неочевидные констрайнты



Данные нашли,
а что дальше?



А какие бывают миграций?



А какие бывают миграций?

1. возвратные – после up-down
данные те же

А какие бывают миграций?

1. возвратные – после up-down
данные те же
2. невозвратные – up влечет потерю
данных

Data Snapshots



Data Snapshots

- snap1
- up
- down
- snap2

Data Snapshots

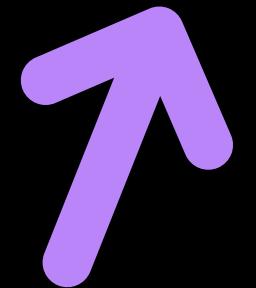
- snap1
 - up
 - down
 - snap2
- snap1==snap2

Профит!!



```
1 -- migrate:up
2 ALTER TABLE users
3   ALTER COLUMN updated_at TYPE timestamp
4   USING updated_at::timestamp;
5
6 -- migrate:down
7 ALTER TABLE users
8   ALTER COLUMN updated_at TYPE text
9   USING updated_at::text;
10
```

id	username	current_mood	updated_at
1	alice	happy	2025-03-14 12:00:00+03:00
2	bob	sad	2025-03-14 14:30:00+03:00
3	charlie	ok	2025-03-14 09:15:00+03:00



Поймаем ошибку на
сравнении данных
до и после up-down



Как брать snapshot-ы

^

Как брать snapshot-ы!

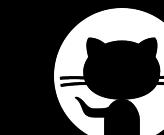


```
1 pg_dump --data-only --column-inserts -file=<before/after>.sql db  
2 diff <before>.sql <after>.sql
```

Как брать snapshot-ы!



```
1 pg_dump --data-only --column-inserts -file=<before/after>.sql db  
2 diff <before>.sql <after>.sql
```



[realkarych/seqwall](https://github.com/realkarych/seqwall) *To be continued...



Выводы

- тестировать без данных — сомнительно



Выводы

- тестировать без данных – сомнительно
- Отлавливаем невозвратные миграции



Выводы

- тестировать без данных – сомнительно
- Отлавливаем невозвратные миграции
- для сэмплов используем `pg_dump + diff` / самописный диффер



Ну теперь-то
точно победа?



Вспоминаем прошлый пример



```
1 -- migrate:up
2 ALTER TABLE users
3   ALTER COLUMN updated_at TYPE timestamp
4   USING updated_at::timestamp;
5
6 -- migrate:down
7 ALTER TABLE users
8   ALTER COLUMN updated_at TYPE text
9   USING updated_at::text;
10
```

<code>id</code>	<code>username</code>	<code>current_mood</code>	<code>updated_at</code>
1	alice	happy	2025-03-14 12:00:00+03
2	bob	sad	2025-03-14 14:30:00+03
3	charlie	ok	2025-03-14 09:15:00+03

Вспоминаем прошлый пример

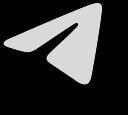


```
1 -- migrate:up
2 ALTER TABLE users
3   ALTER COLUMN updated_at TYPE timestamp
4   USING updated_at::timestamp
5
6 -- migrate:down
7 ALTER TABLE users
8   ALTER COLUMN updated_at TYPE text
9   USING updated_at::text;
10
```

<code>id</code>	<code>username</code>	<code>current_mood</code>	<code>updated_at</code>
1	alice	happy	2025-03-14 12:00:00+03
2	bob	sad	2025-03-14 14:30:00+03
3	charlie	ok	2025-03-14 09:15:00+03

access exclusive
lock

Ловим локи



Ловим локи в рантайме

pg_locks &
pg_stat_activity



```
1  SELECT
2    l.locktype,
3    l.mode,
4    l.granted,
5    a.query,
6    a.state,
7    a.wait_event_type,
8    a.wait_event,
9    a.query_start,
10   a.pid
11  FROM pg_locks l
12  JOIN pg_stat_activity a ON l.pid = a.pid
13 WHERE NOT l.granted OR a.state != 'idle';
14
```

Ловим локи статикой

squawk

```
> squawk example-migration.sql
warning[constraint-missing-not-valid]: By default new constraints require a table scan and
--> example-migration.sql:2:24
|
2 | ALTER TABLE table_name ADD CONSTRAINT field_name_constraint UNIQUE (field_name);
|                                     ^^^^^^^^^^^^^^^^^^
|
warning[disallowed-unique-constraint]: Adding a `UNIQUE` constraint requires an `ACCESS EXC
--> example-migration.sql:2:28
|
2 | ALTER TABLE table_name ADD CONSTRAINT field_name_constraint UNIQUE (field_name);
|                                     ^^^^^^^^^^^^^^^^^^
|
= help: Create an index CONCURRENTLY and create the constraint using the index.

Find detailed examples and solutions for each rule at https://squawkhq.com/docs/rules
Found 2 issues in 1 file (checked 1 source file)
```

А вот такое видели?



```
1 Select a, b, c from
2 some_table as ST WHERE
3 st.c = st.a Order by b
4
```

А вот такое видели?



```
1 Select a, b, c from
2 some_table as ST WHERE
3 st.c = st.a Order by b
4
```





Линтеры

```
$ pip install sqlfluff
$ echo " SELECT a + b FROM tbl; " > test.sql
$ sqlfluff lint test.sql --dialect ansi
== [test.sql] FAIL
L:  1 | P:  1 | LT01 | Expected only single space before 'SELECT' keyword.
          | Found '  '. [layout.spacing]
L:  1 | P:  1 | LT02 | First line should not be indented.
          | [layout.indent]
L:  1 | P:  1 | LT13 | Files must not begin with newlines or whitespace.
          | [layout.start_of_file]
L:  1 | P: 11 | LT01 | Expected only single space before binary operator '+'.
          | Found '  '. [layout.spacing]
L:  1 | P: 14 | LT01 | Expected only single space before naked identifier.
          | Found '  '. [layout.spacing]
L:  1 | P: 27 | LT01 | Unnecessary trailing whitespace at end of file.
          | [layout.spacing]
L:  1 | P: 27 | LT12 | Files must end with a single trailing newline.
          | [layout.end_of_file]
All Finished 🎉!
```

Линтеры

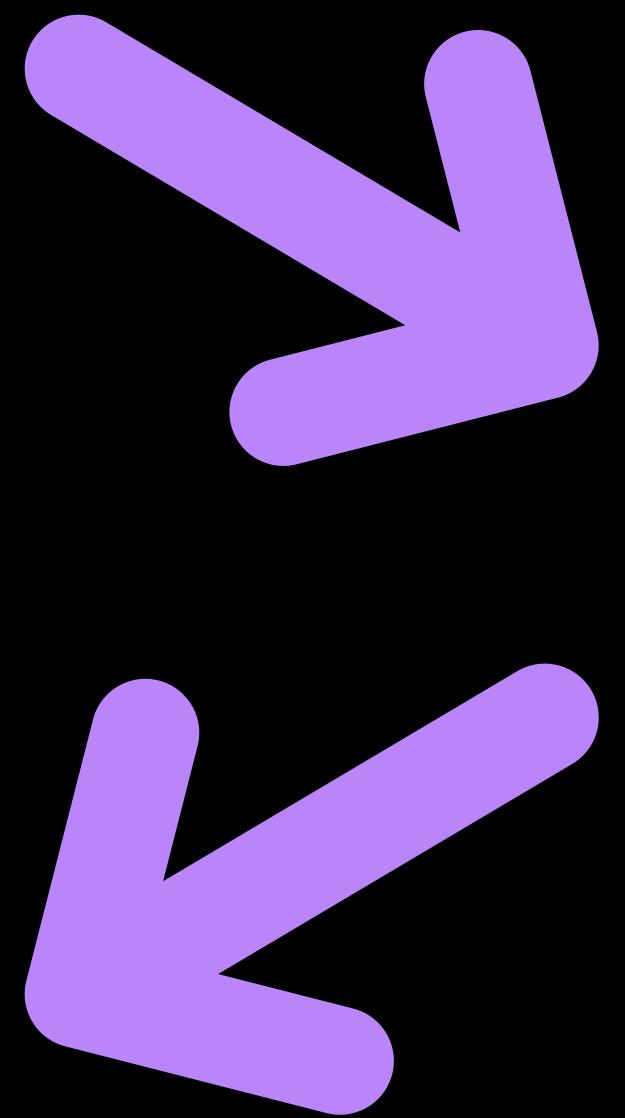
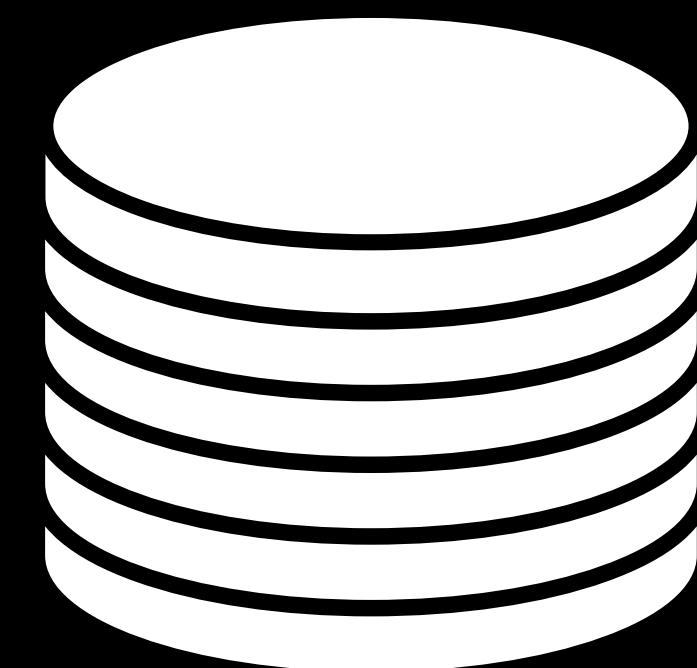


```
$ pip install sqlfluff
$ echo " SELECT a + b FROM tbl; " > test.sql
$ sqlfluff lint test.sql --dialect ansi
== [test.sql] FAIL
L:  1 | P:  1 | LT01 | Expected only single space before 'SELECT' keyword.
          | Found '  '. [layout.spacing]
L:  1 | P:  1 | LT02 | First line should not be indented.
          | [layout.indent]
L:  1 | P:  1 | LT13 | Files must not begin with newlines or whitespace.
          | [layout.start_of_file]
L:  1 | P: 11 | LT01 | Expected only single space before binary operator '+'.
          | Found '  '. [layout.spacing]
L:  1 | P: 14 | LT01 | Expected only single space before naked identifier.
          | Found '  '. [layout.spacing]
L:  1 | P: 27 | LT01 | Unnecessary trailing whitespace at end of file.
          | [layout.spacing]
L:  1 | P: 27 | LT12 | Files must end with a single trailing newline.
          | [layout.end_of_file]

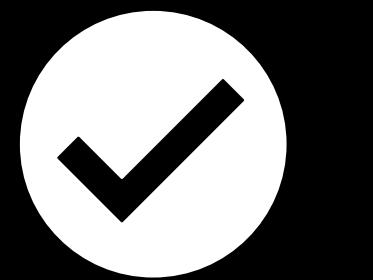
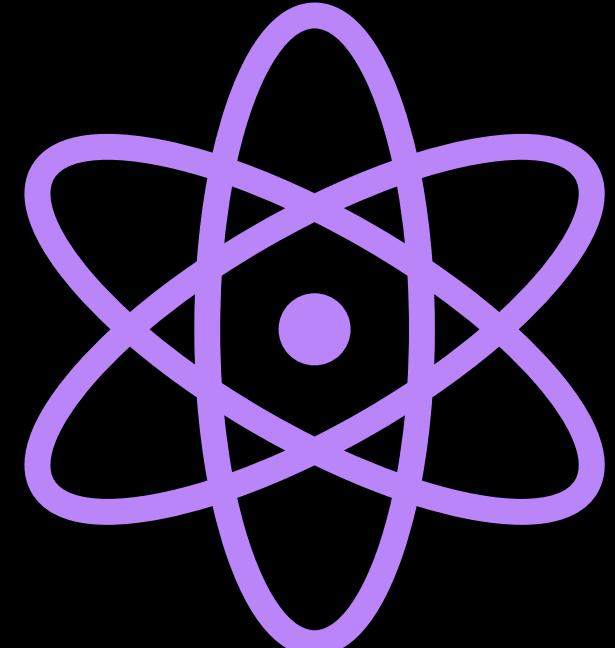
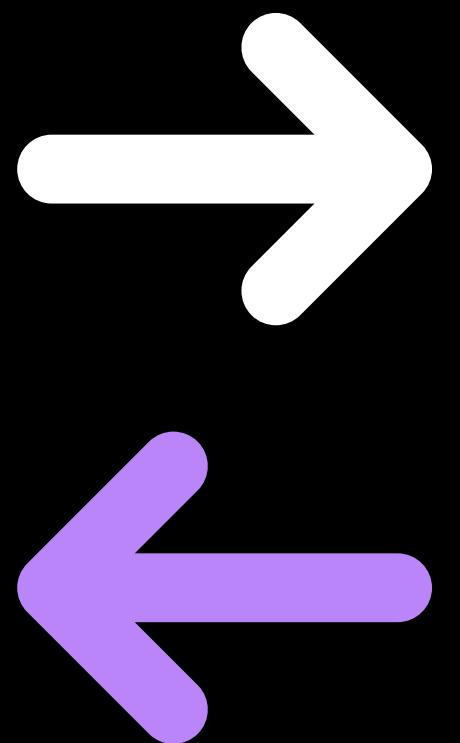
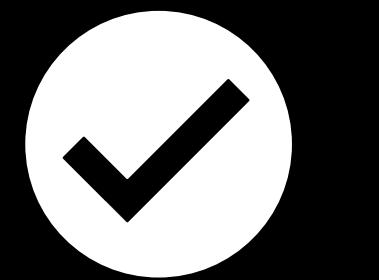
All Finished 🎉!
```



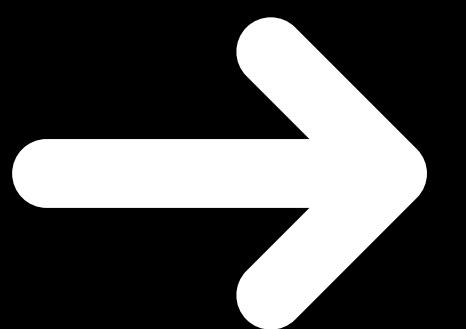
Надежный пайплайн



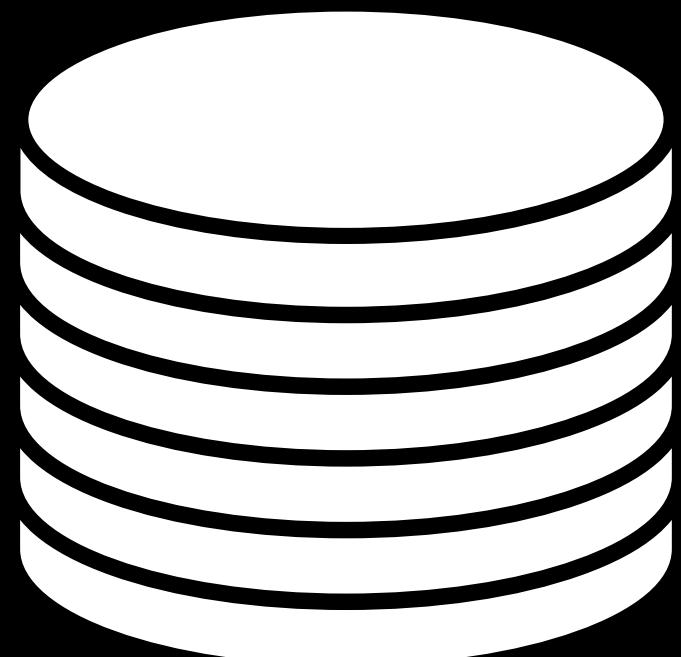
Надежный пайплайн



CR-1

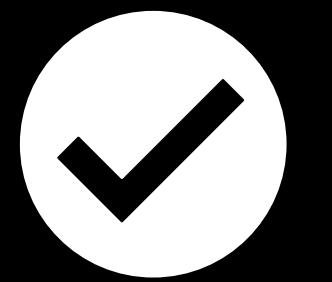
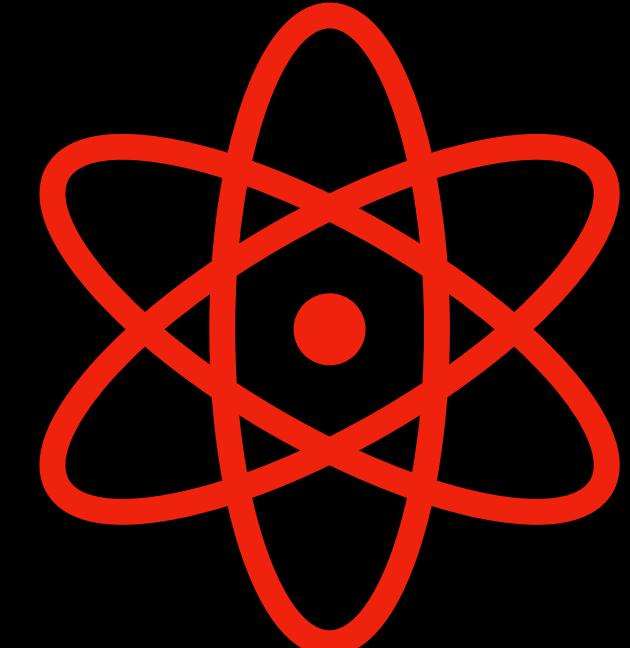
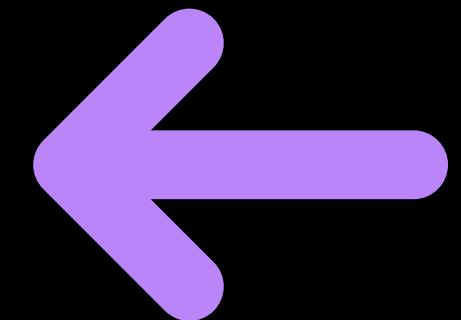
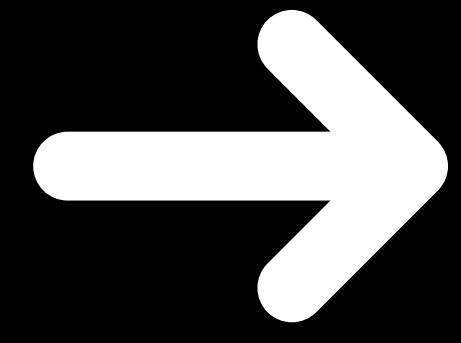
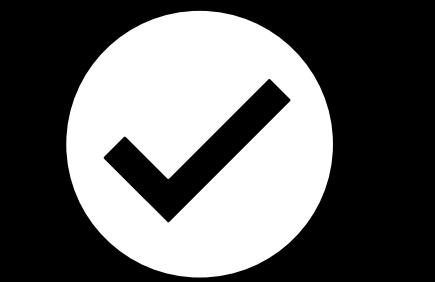


CR-2

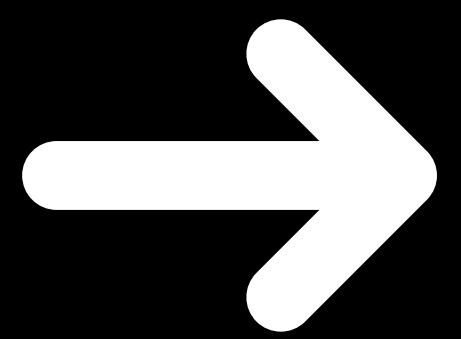


Delivery

Надежный пайплайн



CR-1



CR-2



Delivery



Роадмап

1. Пишем интеграционные тесты



Роадмап

1. Пишем интеграционные тесты
2. Включаем жестокий стат.анализ и линтеры



Роадмап

1. Пишем интеграционные тесты
2. Включаем жестокий стат.анализ и линтеры
3. Тестируем схему, локи



Роадмап

1. Пишем интеграционные тесты
2. Включаем жестокий стат.анализ и линтеры
3. Тестируем схему, локи
4. Тестируем данные

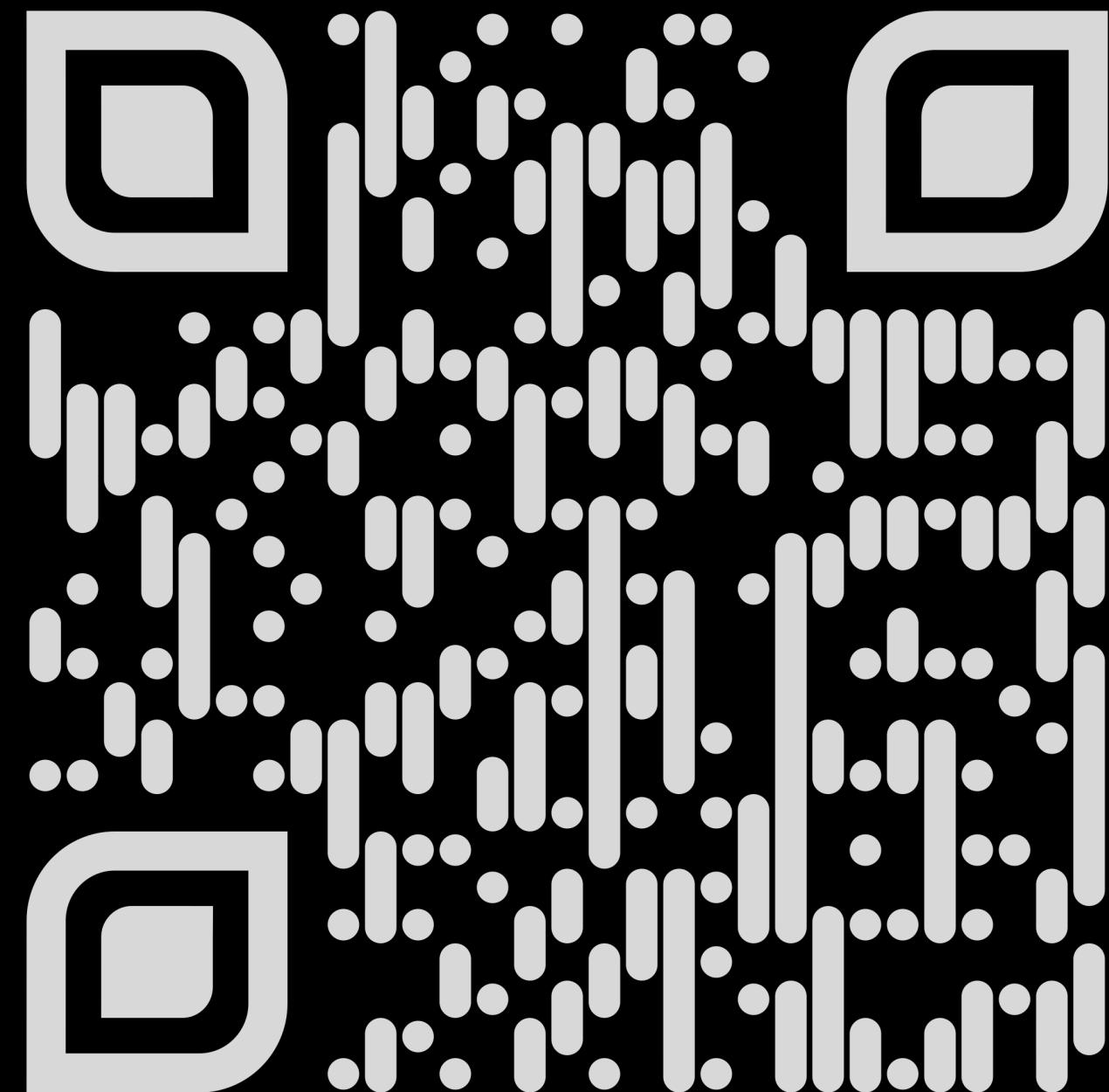


Роадмап

1. Пишем интеграционные тесты
2. Включаем жесткий стат.анализ и линтеры
3. Тестируем схему, локи
4. Тестируем данные
5. Тестируем многофазные миграции, регрессии перформанса, ...



Пора тестировать миграции



фидбэк →
← слайды

