I use ENUM

Vindicating the Underdog of Data Types

Boriss Mejías – Air Guitar Player
Solution Architect EDB

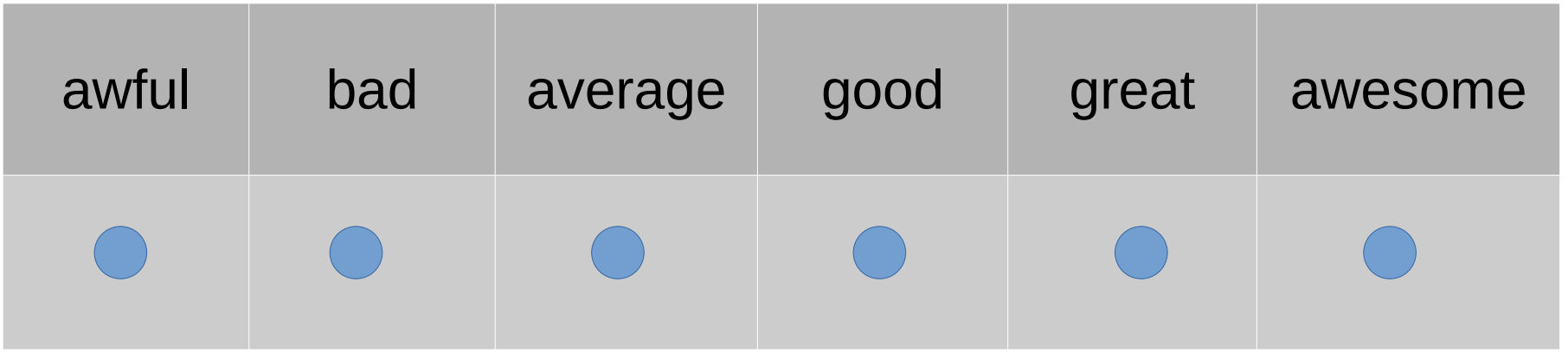
tchorix

FOSDEM 2021 – PostgreSQL Devroom

Feedback from FOSDEM visitors

What do you think about the

talk / speaker / devroom/ booth / FOSDEM ?



Register Feedback Devroom

```
CREATE TABLE devroom_feedback (
    dev_fbk_id INTEGER PRIMARY KEY
  , devroom_id INTEGER REFERENCES devrooms(id)
  , moment     TIMESTAMP WITH TIME ZONE
  , feedback   TEXT
);
```

Register Feedback Devroom

```
CREATE TABLE devroom_feedback (
    dev_fbk_id INTEGER PRIMARY KEY
  , devroom_id INTEGER REFERENCES devrooms(id)
  , moment     TIMESTAMP WITH TIME ZONE
  , feedback   TEXT
);

INSERT INTO devroom_feedback
  VALUES (1984, 42, now(), 'awesome');
INSERT 0 1

INSERT INTO devroom_feedback
  VALUES (1985, 42, now(), 'awezome');
INSERT 0 1
```

Guarantee valid Feedback

```
CREATE TABLE devroom_feedback (
    dev_fbk_id INTEGER PRIMARY KEY
  , devroom_id INTEGER REFERENCES devrooms(id)
  , moment     TIMESTAMP WITH TIME ZONE
  , feedback   TEXT
);

ALTER TABLE devroom_feedback
ADD CONSTRAINT "devroom_feedback_check"
CHECK (feedback = ANY(ARRAY['awful', 'bad',
                            'average', 'good',
                            'great', 'awesome']));
```

Only valid values

```
INSERT INTO devroom_feedback
   VALUES (1985, 42, now(), 'awezome');
```

Only valid values

```
INSERT INTO devroom_feedback
   VALUES (1985, 42, now(), 'awezome');

ERROR:  new row for relation "devroom_feedback" violates
        check constraint "devroom_feedback_check"
DETAIL:  Failing row contains (1985, 42, 2021-01-17
23:32:36.147916+01, awezome).
```

Guarantee valid Feedback

```
CREATE TABLE devroom_feedback (
    dev_fbk_id  INTEGER PRIMARY KEY
  , devroom_id  INTEGER REFERENCES devrooms(id)
  , moment      TIMESTAMP WITH TIME ZONE
  , feedback    TEXT
);

ALTER TABLE devroom_feedback
ADD CONSTRAINT "devroom_feedback_check"
CHECK (feedback = ANY(ARRAY['awful', 'bad',
                            'average', 'good',
                            'great', 'awesome']));
```

Guarantee valid Feedback

```
CREATE TABLE talk_feedback (
    talk_fbk_id INTEGER PRIMARY KEY
  , talk_id     INTEGER REFERENCES talks(id)
  , moment      TIMESTAMP WITH TIME ZONE
  , feedback    TEXT
);

ALTER TABLE talk_feedback
ADD CONSTRAINT "talk_feedback_check"
CHECK (feedback = ANY(ARRAY['awful', 'bad',
                            'average', 'good',
                            'great', 'awesome']));
```

Normalization

Normalize Feedback

```
CREATE TABLE feedbacks (
    id      INTEGER PRIMARY KEY
  , name    TEXT
);

INSERT INTO feedbacks VALUES
    (1, 'awful'),
    (2, 'bad'),
    (3, 'average'),
    (4, 'good'),
    (5, 'great'),
    (6, 'awesome');
```

Normalize Feedback

```
CREATE TABLE feedbacks (
    id      INTEGER PRIMARY KEY
  , name    TEXT
);

CREATE TABLE devroom_feedback (
    dev_fbk_id  INTEGER PRIMARY KEY
  , devroom_id  INTEGER REFERENCES devrooms(id)
  , moment      TIMESTAMP WITH TIME ZONE
  , feedback_id INTEGER REFERENCES feedbacks(id)
);
```

Normalize Feedback

```
CREATE TABLE feedbacks (
    id      INTEGER PRIMARY KEY
  , name    TEXT
);

CREATE TABLE talk_feedback (
    talk_fbk_id INTEGER PRIMARY KEY
  , talk_id     INTEGER REFERENCES talks(id)
  , moment      TIMESTAMP WITH TIME ZONE
  , feedback_id INTEGER REFERENCES feedbacks(id)
);
```

Inserting Feedback

```
INSERT INTO devroom_feedback
  VALUES (1984, 42, now(), 6);

INSERT INTO devroom_feedback
  VALUES (1985, 42, now(), 6);

INSERT INTO devroom_feedback
  VALUES (1986, 42, now(),
                    (SELECT id FROM feedbacks
                     WHERE name='awesome'));
```

Enter ENUM

ENUM Feedback

```
CREATE TYPE feedback AS ENUM
  ('awful', 'bad', 'average',
   'good', 'great', 'awesome');
```

ENUM Feedback

```
CREATE TYPE feedback AS ENUM
  ('awful', 'bad', 'average',
   'good', 'great', 'awesome');

CREATE TABLE devroom_feedback (
    dev_fbk_id  INTEGER PRIMARY KEY
  , devroom_id  INTEGER REFERENCES devrooms(id)
  , moment      TIMESTAMP WITH TIME ZONE
  , feedback    feedback
);
```

Only valid values

```
INSERT INTO devroom_feedback
   VALUES (1984, 42, now(), 'awesome');
INSERT 0 1
```

Only valid values

```
INSERT INTO devroom_feedback
   VALUES (1984, 42, now(), 'awesome');
INSERT 0 1

INSERT INTO devroom_feedback
   VALUES (1985, 42, now(), 'awezome');

ERROR:  invalid input value for enum feedback: "awezome"
LINE 2: VALUES (1985, 42, now(), 'awezome');
                                 ^
```

ENUM Feedback

```
CREATE TYPE feedback AS ENUM
  ('awful', 'bad', 'average',
   'good', 'great', 'awesome');

CREATE TABLE devroom_feedback (
    dev_fbk_id  INTEGER PRIMARY KEY
  , devroom_id  INTEGER REFERENCES devrooms(id)
  , moment      TIMESTAMP WITH TIME ZONE
  , feedback    feedback
);
CREATE TABLE talk_feedback (
    talk_fbk_id INTEGER PRIMARY KEY
  , talk_id     INTEGER REFERENCES talks(id)
  , moment      TIMESTAMP WITH TIME ZONE
  , feedback    feedback
);
```

What about Performance?

Performance Comparison

Insert 2267709 rows into **devroom_feedback**

feedback	**TEXT**	**36.583757 s**
feedback	**TEXT CHECK**	**36.256998 s**
feedback_id	**INT FOREIGN KEY**	**50.505442 s**
feedback	**ENUM feedback**	**36.606752 s**

Approach Comparison

	Approach	Ins
feedback	**TEXT CHECK**	
feedback_id	**INT FOREIGN KEY**	
feedback	**ENUM feedback**	

Retrieve Data

```
-- TEXT CHECK and ENUM feedback
SELECT df.feedback, count(*) AS c
FROM devroom_feedback df
JOIN devrooms d ON (df.devroom_id = d.id)
WHERE d.name = 'Python'
GROUP BY df.feedback;
```

Retrieve Data

```
-- TEXT CHECK and ENUM feedback
SELECT df.feedback, count(*) AS c
FROM devroom_feedback df
JOIN devrooms d ON (df.devroom_id = d.id)
WHERE d.name = 'Python'
GROUP BY df.feedback;

-- INT FOREIGN KEY
SELECT f.name AS feedback, count(*) AS c
FROM devroom_feedback df
JOIN devrooms d ON (df.devroom_id = d.id)
JOIN feedbacks f  ON (df.feedback_id = f.id)
WHERE d.name = 'Python'
GROUP BY f.name;
```

Approach Comparison

	Approach	Ins	Get
feedback	**TEXT CHECK**		
feedback_id	**INT FOREIGN KEY**		
feedback	**ENUM feedback**		

Data Integrity

```
-- TEXT CHECK
SELECT * FROM devroom_feedback WHERE feedback = 'graet';
 dev_fbk_id | devroom_id | moment | feedback_id | id | name
------------+------------+--------+-------------+----+------
(0 rows)
Time: 92.324 ms

-- INT FOREIGN KEY
SELECT * FROM devroom_feedback df
JOIN feedbacks f ON (df.feedback_id = f.id)
WHERE f.name = 'graet';
 dev_fbk_id | devroom_id | moment | feedback_id | id | name
------------+------------+--------+-------------+----+------
(0 rows)
Time: 102.892 ms
```

Data Integrity

```
-- ENUM feedback
SELECT * FROM devroom_feedback WHERE feedback = 'graet';
ERROR:  invalid input value for enum feedback: "graet"
LINE 1: SELECT * FROM devroom_feedback WHERE feedback = 'graet';
                                                         ^
Time: 0.603 ms
```

Approach Comparison

	Approach	Ins	Get
feedback	**TEXT CHECK**		
feedback_id	**INT FOREIGN KEY**		
feedback	**ENUM feedback**		

Why is Normalization Relevant?

Update Feedback – TEXT CHECK

```
BEGIN;
ALTER TABLE devroom_feedback
DROP CONSTRAINT devroom_feedback_check;

UPDATE devroom_feedback SET feedback = 'meh'
WHERE feedback = 'average';

ALTER TABLE devroom_feedback
ADD CONSTRAINT "devroom_feedback_check"
CHECK (feedback = ANY(ARRAY['aweful', 'bad',
                            'meh', 'good',
                            'great', 'awesome']));

COMMIT;
```

Update Normalized Feedback

```
UPDATE feedbacks SET name = 'meh'
WHERE name = 'average';
```

Approach Comparison

	Approach	Ins	Get	Upd
feedback	**TEXT CHECK**			
feedback_id	**INT FOREIGN KEY**			
feedback	**ENUM feedback**			

Update ENUM feedback

```
ALTER TYPE feedback
RENAME VALUE 'average' TO 'meh';
```

Approach Comparison

	Approach	Ins	Get	Upd
feedback	**TEXT CHECK**			
feedback_id	**INT FOREIGN KEY**			
feedback	**ENUM feedback**			

Delete Data – No more Awesome

```
BEGIN;
ALTER TABLE devroom_feedback
DROP CONSTRAINT devroom_feedback_check;

UPDATE devroom_feedback SET feedback = 'great'
WHERE feedback = 'awesome';

ALTER TABLE devroom_feedback
ADD CONSTRAINT "devroom_feedback_check"
CHECK (feedback = ANY(ARRAY['aweful', 'bad', 'meh',
                            'good', 'great']));

COMMIT;
```

Delete Normalized Data

```
DELETE FROM feedbacks WHERE name = 'awesome';
```

Delete Normalized Data

```
DELETE FROM feedbacks WHERE name = 'awesome';
ERROR:  update or delete on table "feedbacks" violates
foreign key constraint "devroom_feedback_feedback_id_fkey"
on table "devroom_feedback"
DETAIL:  Key (id)=(6) is still referenced from table
"devroom_feedback".
```

Delete ENUM feedback

There is no `ALTER TYPE mood DROP VALUE 'awesome';`

Delete ENUM feedback

There is no ~~`ALTER TYPE mood DROP VALUE 'awesome';`~~

Create another type and run `ALTER COLUMN TYPE`

Approach Comparison

	Approach	Ins	Get	Upd	Del
feedback	**TEXT CHECK**				
feedback_id	**INT FOREIGN KEY**				
feedback	**ENUM feedback**				

Extra Feature

ENUMs are ordered

```
CREATE TYPE feedback AS ENUM
  ('awful', 'bad', 'average',
   'good', 'great', 'awesome');
```

ENUMs are ordered

```
CREATE TYPE feedback AS ENUM
  ('awful', 'bad', 'average',
   'good', 'great', 'awesome');

SELECT d.name, count(*) as c
FROM devroom_feedback df
JOIN devrooms d ON (df.devroom_id = d.id)
WHERE df.feedback > 'good'
GROUP BY d.name;
```

Cast to alternative order

```
CREATE TYPE feedback AS ENUM
  ('awful','bad','average','good','great','awesome');

CREATE TYPE feedback_alt AS ENUM
  ('awful','bad','average','awesome','good','great');
```

Cast to alternative order

```
CREATE TYPE feedback AS ENUM
  ('awful','bad','average','good','great','awesome');

CREATE TYPE feedback_alt AS ENUM
  ('awful','bad','average','awesome','good','great');

SELECT d.name, count(*) as c
FROM devroom_feedback df
JOIN devrooms d ON (df.devroom_id = d.id)
WHERE df.feedback::text::feedback_alt > 'good'
GROUP BY d.name;
```

Add ordered values to ENUM

```
ALTER TYPE feedback RENAME VALUE 'average' TO 'meh';

ALTER TYPE feedback ADD VALUE 'unlucky' BEFORE 'meh';
ALTER TYPE feedback ADD VALUE 'fine' AFTER 'meh';

\dT+
                          List of data types
 Schema |   Name   | Internal name | Size | Elements |
--------+----------+---------------+------+----------+
 public | feedback | feedback      | 4    | awful   +|
        |          |               |      | bad     +|
        |          |               |      | unlucky +|
        |          |               |      | meh     +|
        |          |               |      | fine    +|
        |          |               |      | good    +|
        |          |               |      | great   +|
        |          |               |      | awesome  |
```

ENUM on the pg_catalog

```
SELECT e.enumsortorder, e.enumlabel
FROM pg_enum e
JOIN pg_type t ON (e.enumtypid = t.oid)
WHERE t.typname = 'feedback'
ORDER BY enumsortorder;
```

```
 enumsortorder | enumlabel
---------------+-----------
             1 | awful
             2 | bad
           2.5 | unlucky
             3 | meh
           3.5 | fine
             4 | good
             5 | great
             6 | awesome
```

Lookup Table ordered by id

Table feedbacks

id	name
1	awful
2	bad
3	meh
4	good
5	great
6	awesome

Lookup Table with rank

Table feedbacks

id	name	rank
1	awful	1
2	bad	2
3	meh	3
4	good	4
5	great	5
6	awesome	6

Lookup Table with rank

Table feedbacks

id	name	rank
1	awful	1
2	bad	2
3	meh	3
4	good	4
5	great	5
6	awesome	6
7	unlucky	2.5
8	fine	3.5

Lookup Table with rank

Table feedbacks

id	name	rank
1	awful	1
2	bad	2
7	unlucky	2.5
3	meh	3
8	fine	3.5
4	good	4
5	great	5
6	awesome	6

Better than good

```
SELECT d.name, count(*) as c
FROM devroom_feedback df
JOIN devrooms d ON (df.devroom_id = d.id)
WHERE df.feedback > 'good'
GROUP BY d.name;
```

Better than good

```
SELECT d.name, count(*) as c
FROM devroom_feedback df
JOIN devrooms d ON (df.devroom_id = d.id)
WHERE df.feedback > 'good'
GROUP BY d.name;

SELECT d.name, count(*) as c
FROM devroom_feedback_lt df
JOIN devrooms d ON (df.devroom_id = d.id)
JOIN feedbacks f ON (df.feedback_id = f.id)
WHERE f.rank > (SELECT rank
                FROM feedbacks WHERE name = 'good')
GROUP BY d.name;
```

Approach Comparison

	Approach	Ins	Get	Upd	Del	Ord
feedback	**TEXT CHECK**					
feedback_id	**INT FOREIGN KEY**					
feedback	**ENUM feedback**					

Closing Words

Now you know ENUM better

It maps your data modeling

Efficiently normalized

Closing Words

Now you know ENUM better

It maps your data modeling

Efficiently normalized

This presentation is specific to PostgreSQL

Thank you

Boriss Mejías
tchorix