

Binary trees

Key/value based approach:

```
{ "value":3,
  "left": {
    "value":1,
    "left": {"value":0},
    "right": {"value":2}
  },
  "right": {
    "value":5,
    "left": {"value":4},
    "right": {"value":6}
  }
}
```

Schema:

```
{ "type": "object",
  "oneOf": [{" $ref": "#/definitions/leaf" },
            {" $ref": "#/definitions/node" } ],
  "definitions": {
    "leaf": {
      "type": "object",
      "properties": {
        "value": {"type": "integer"},
        "right": false,
        "left": false
      },
      "required": ["value"],
    },
    "node": {
      "type": "object",
      "properties": {
        "value": {"type": "integer"},
        "left": {" $ref": "#"},
        "right": {" $ref": "#"},
        "required": ["value", "left", "right"]
      }
    }
  }
}
```

Variant schema: Just remove required "value" from branch

Queries (as postgres Path values, first for objects then for arrays):

```
'strict $.**.value'
'strict $.**.left.value'
'lax $.**.value'
'strict $.**?(@ > 3) '
'strict $.**?(@ > $.value) '
```

Array-based approach:

```
{ "value":3,
  "children": [
    {"value":1,
      "children": [{"value":0},{ "value":2}]},
    {"value":5,
      "children": [{"value":4},{ "value":6}]}
  ]
}
```

Schema:

```
{
  "properties": {
    "value": {"type": "integer"},
    "children": {"type": "array",
      "items" : {"$ref": "#"},
      "minItems":2,
      "maxItems":2}},
  "required": ["value"]
}
```

Variant schema: Kind of tricky. Requires a separate definitions for branches/leafs, with leafs requiring "value" and branches requiring "children" (and having data optional).

Queries (as postgres Path values, first for objects then for arrays):

```
'strict $.**.value'
'strict $.**.children[0].value'
'strict $.children[*].children[*].value'
'strict $.**?(@ > 3) '
'strict $.**?(@ > $.value) '
```

Flights

Key/value pairs can be used to have primary keys (because there are no compound keys!) Tiny example data (used to build schema):

```
{ "Airports": {"GOT": "Gothenburg"},
  "FlightCodes": {"SK111": "SAS"},
  "Flights": {"SK111" : {dep : "GOT", dest: "FRA"}}
}
```

Schema:

```
{ "type": "object",
  "properties": {
    "Airports": {
      "type": "object",
      "additionalProperties": {"type": "string"}
    },
    "FlightCodes": {
      "type": "object",
      "additionalProperties": {"type": "string"}
    },
    "Flights": {
      "type": "object",
      "additionalProperties": {
        "type": "object",
        "properties": {
          "dep": {"type": "string"},
          "dest": {"type": "string"}
        },
        "required": ["dep", "dest"],
        "additionalProperties": false
      }
    }
  },
  "required": ["Airports", "FlightCodes", "Flights"]
}
```

Complete data:

```
{ "Airports": {"GOT": "Gothenburg",
               "FRA": "Frankfurt",
               "ORY": "Paris",
               "MUC": "Munich",
               "MLA": "Malta"},
  "FlightCodes": {"SK111": "SAS",
                  "AF222": "Air France",
                  "AB222": "Air Berlin",
                  "KM111": "Air Malta"},
  "Flights": {"SK111" : {dep : "GOT", dest: "FRA"},
              "AF222" : {dep : "ORY", dest: "MLA"},
              "AB222" : {dep : "FRA", dest: "MUC"},
              "KM111" : {dep : "MUC", dest: "MLA"}}}
```

Query to automatically export database:

-- Convert all three tables into one big JSON document

WITH

```
Ap AS (SELECT json_object_agg(code, city)
      AS jsondata FROM Airports
      ),
```

```
Fc AS (SELECT json_object_agg(code, airlineName)
      AS jsondata FROM FlightCodes
      ),
```

```
F AS (SELECT json_object_agg(
      code, jsonb_build_object(
        'departureAirport', departureAirport,
        'destinationAirport', destinationAirport
      )
    ) AS jsondata FROM Flights
    )
```

```
SELECT jsonb_pretty(
  jsonb_build_object(
    'Airports', (SELECT jsondata FROM Ap),
    'FlightCodes', (SELECT jsondata FROM Fc),
    'Flights', (SELECT jsondata FROM F)
  )
)
```

```
);
```

BONUS!:

```
'strict $.Flights.keyvalue()?(@.value.dest == "MLA").key'
```

Finds the flight code of all flights to Malta

Applications

(A) Here we use applicant id numbers as keys, and associate it with the applicants name and a list of all their choices. We array positions to represent choice numbers (indexed from 0 instead of 1).

```
{
  "a1": { "name": "Andersson", "choices": [
    { "code": "MPSOF", "meritScore": 750 },
    { "code": "MPALG", "meritScore": 750 },
    { "code": "MPCSN", "meritScore": 800 }
  ] },
  "a2": { "name": "Jonsson", "choices": [
    { "code": "MPALG", "meritScore": 700 }
  ] },
  "a3": { "name": "Larsson", "choices": [
    { "code": "MPCSN", "meritScore": 850 },
    { "code": "MPALG", "meritScore": 850 }
  ] }
}
```

(B)

```
{
  "additionalProperties": {
    "type": "object",
    "properties": {
      "name": { "type": "string" },
      "choices": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "code": { "type": "string" },
            "meritScore": { "type": "integer" }
          },
          "additionalProperties": false,
          "required": ["code", "meritScore"]
        }
      }
    },
    "required": ["choices", "name"]
  },
  "type": "object"
}
```

(C) Note that application 1 has array index 0

```
'$.**?(@.choices[0].meritScore > 800).choices[0]'
```

BONUS PATH!:

```
'$.**?(@.choices[0].meritScore > 800).name'
```

Finds the name of the applicant in question