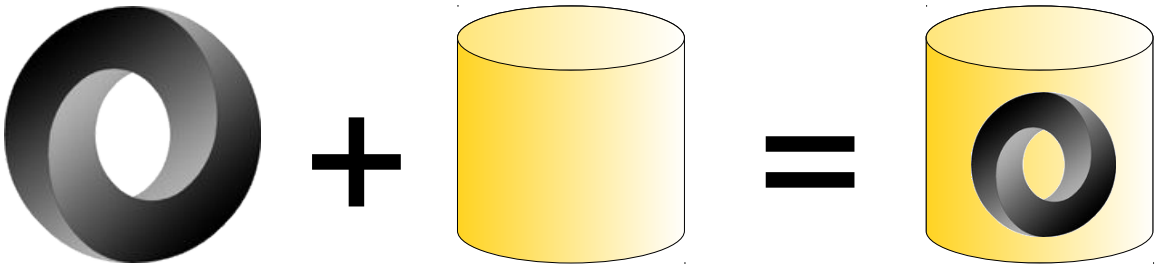
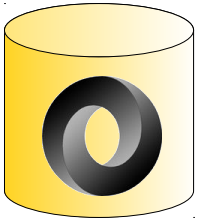


JSON-Daten in SQL-DBs

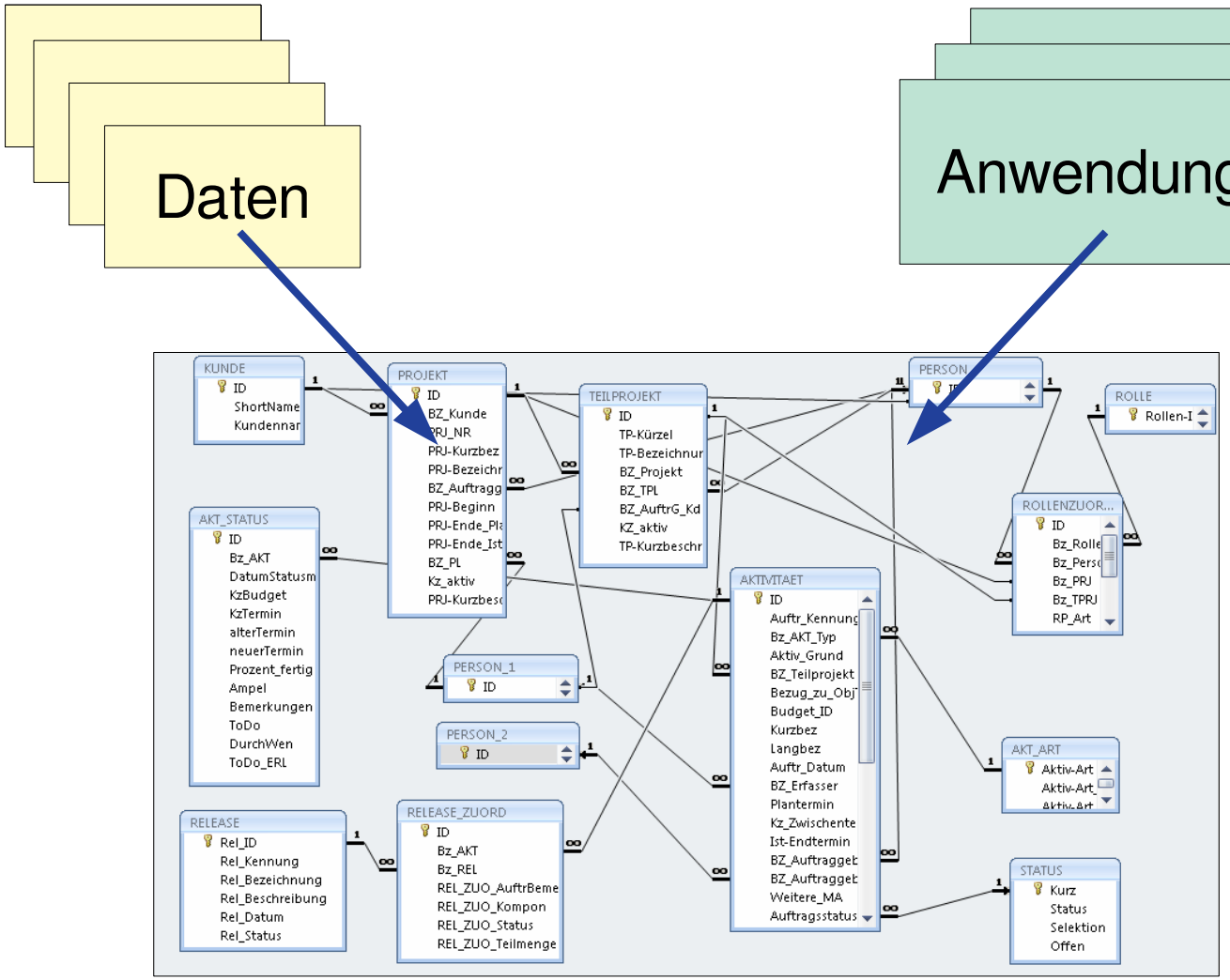
Uwe Berger
bergeruw@gmx.net

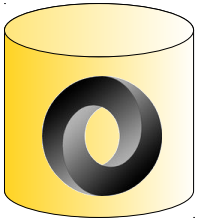
Uwe Berger

- Beruf: Softwareentwickler
- Freizeit: u.a. mit Hard- und Software rumspielen
- Linux seit ca. 1995
- BraLUG e.V.
- bergeruw@gmx.net

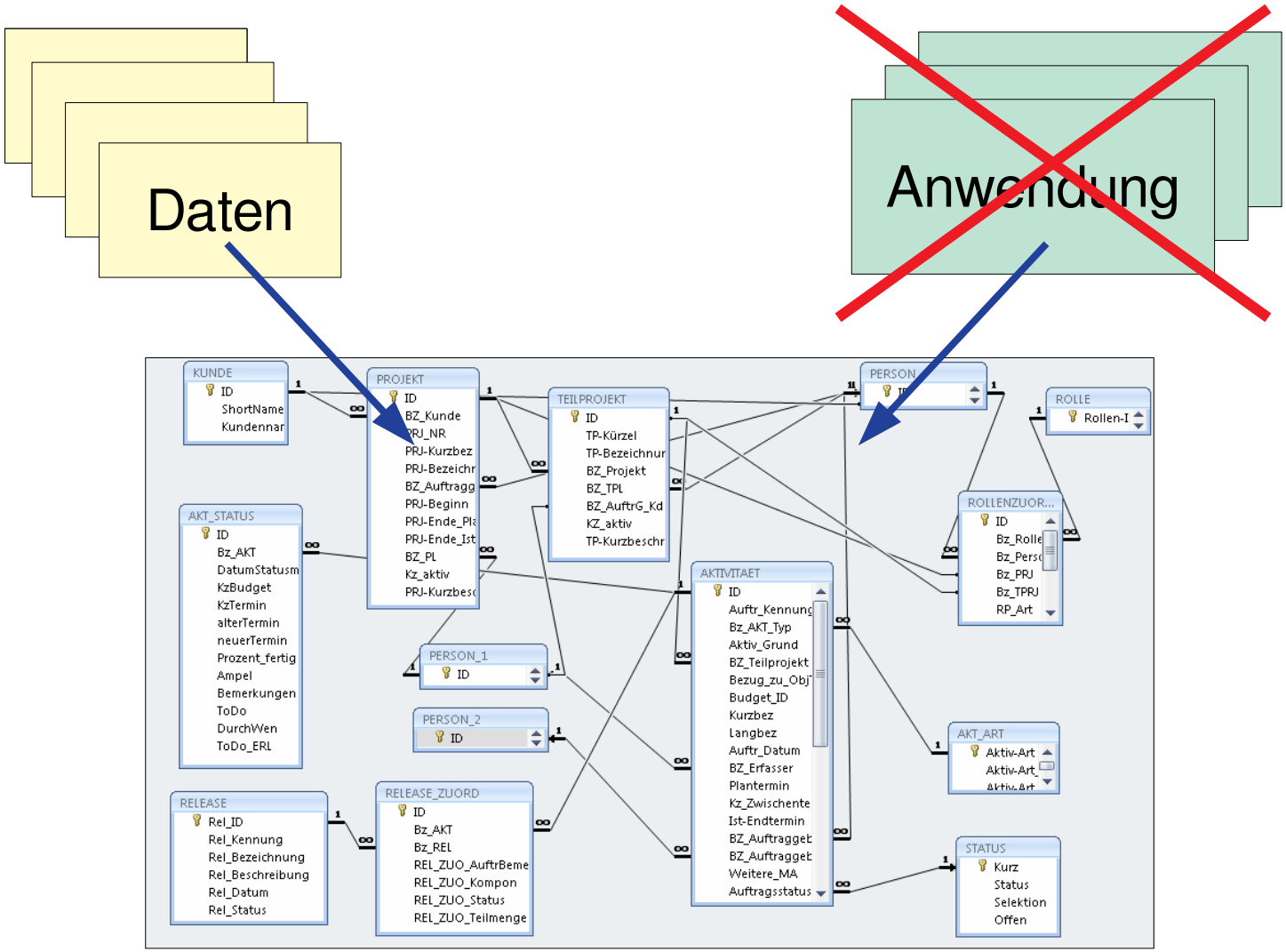


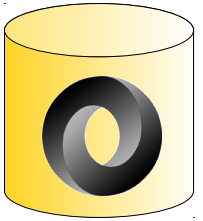
Meine Motivation



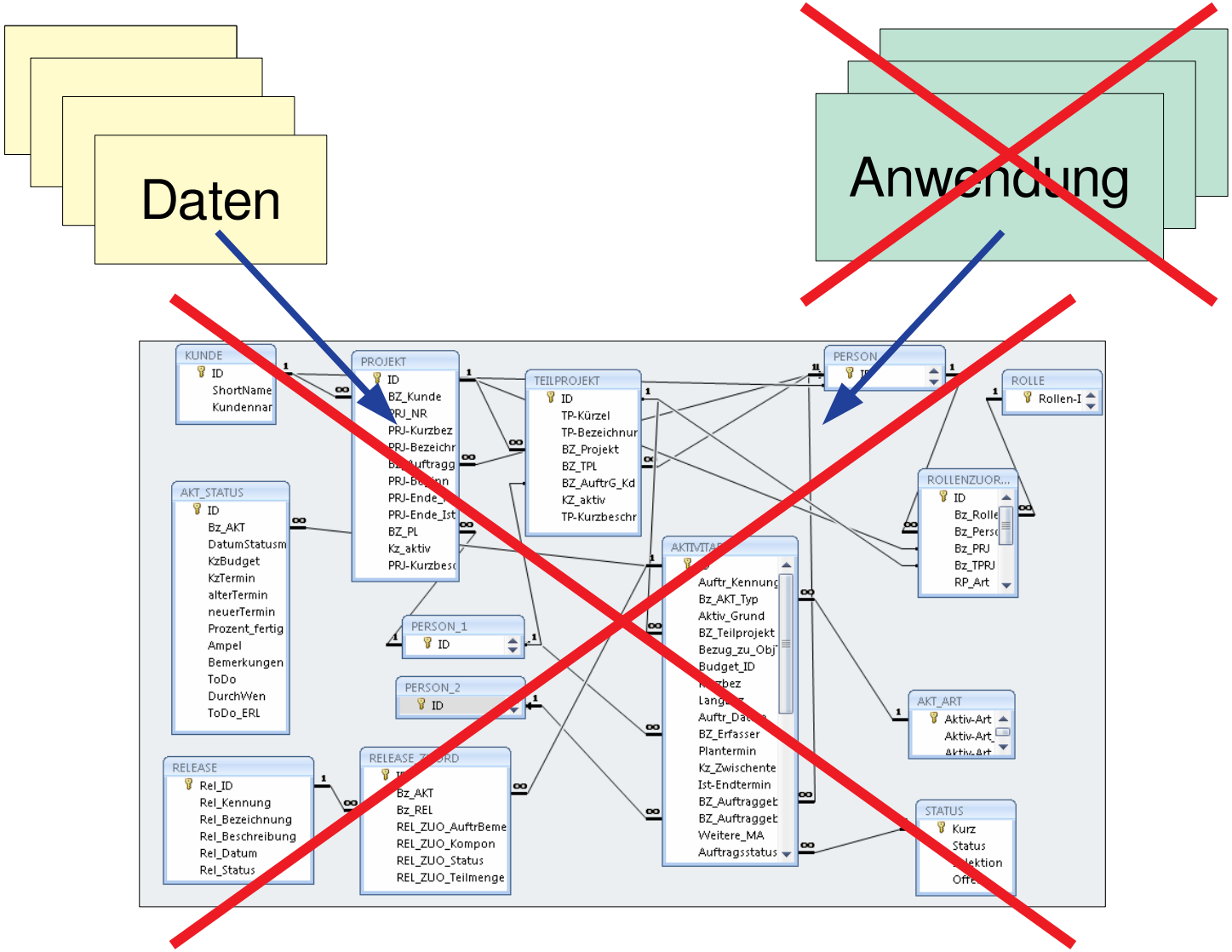


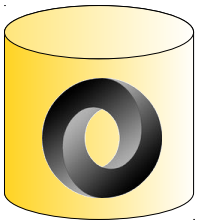
Meine Motivation





Meine Motivation

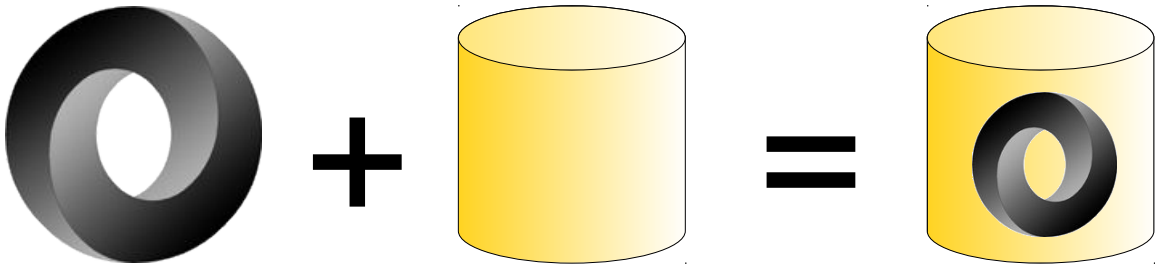


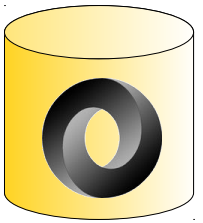


Meine Motivation

Der Plan!

- ...viele meiner anfallenden Daten sind JSON-Objekte
- ...man schreibt einfach jedes JSON-Objekt zeilenweise in eine Textspalte einer SQL-Tabelle...

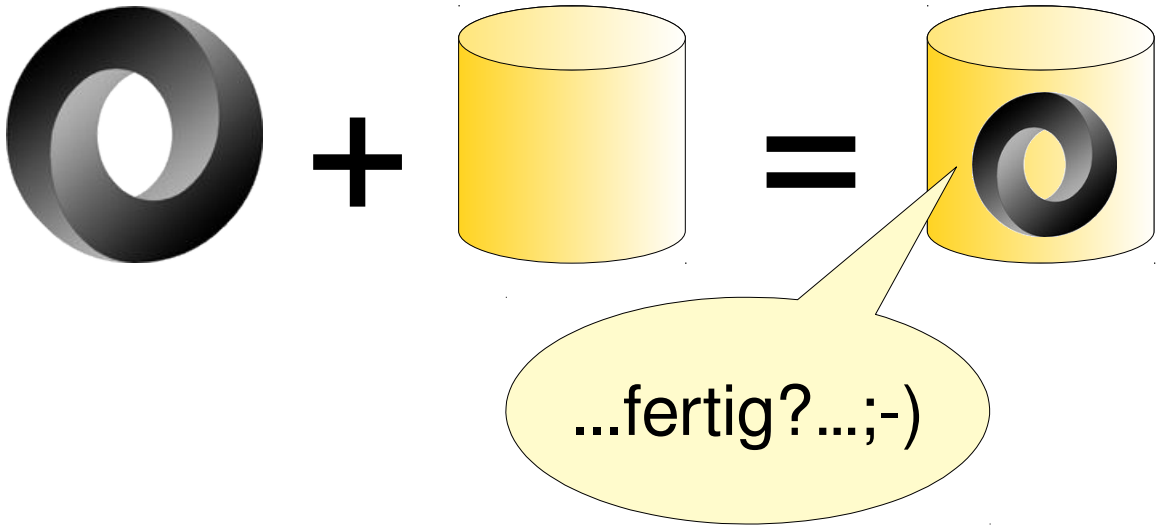


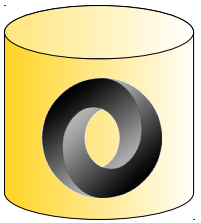


Meine Motivation

Der Plan!

- ...viele meiner anfallenden Daten sind JSON-Objekte
- ...man schreibt einfach jedes JSON-Objekt zeilenweise in eine Textspalte einer SQL-Tabelle...

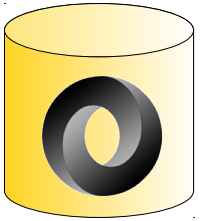




Meine Motivation

Der Plan!

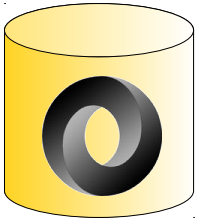
- ...viele meiner anfallenden Daten sind JSON-Objekte
- ...man schreibt einfach jedes JSON-Objekt zeilenweise in eine Textspalte einer SQL-Tabelle...



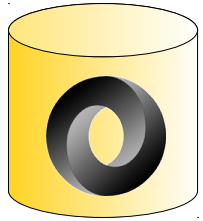
Inhalt

- JSON
- SQL → kennt ja jeder...
- JSON + SQL (am Bsp. von SQLite)
  - Erzeugen, Auslesen, Manipulieren von JSON-Objekten in SQL-Tabellen
  - Performance

JSON

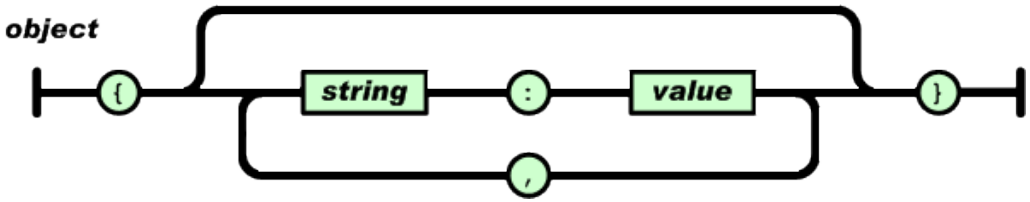


- JSON = **J**ava**S**cript **O**bject **N**otation
  - kompaktes , lesbares, strukturiertes Datenformat
  - „Javascript“ → jedes gültige JSON-Objekt ist (fast) ein gültiges Javascript
- Spezifikation Anfang der 2000er durch Douglas Crockford
- Standards:
  - RFC 4627, 7159, 8259
  - ECMA-404 (nur Syntax)
- wird hauptsächlich zur Übertragung und Speicherung von strukturierten Daten verwendet

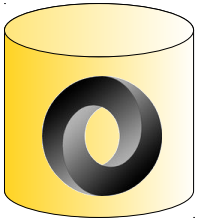


JSON (Syntax)

JSON-Objekt:

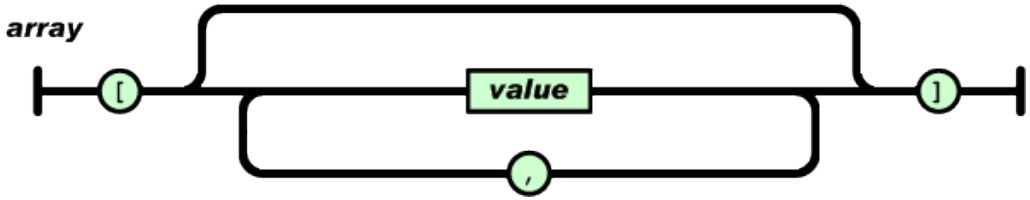



```
{
   "a" : "Text"
}


{
   "a" : "Text",
   "b" : "noch ein Text",
   "c" : 42
}
```

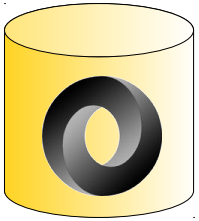


JSON (Syntax)

JSON-Array:

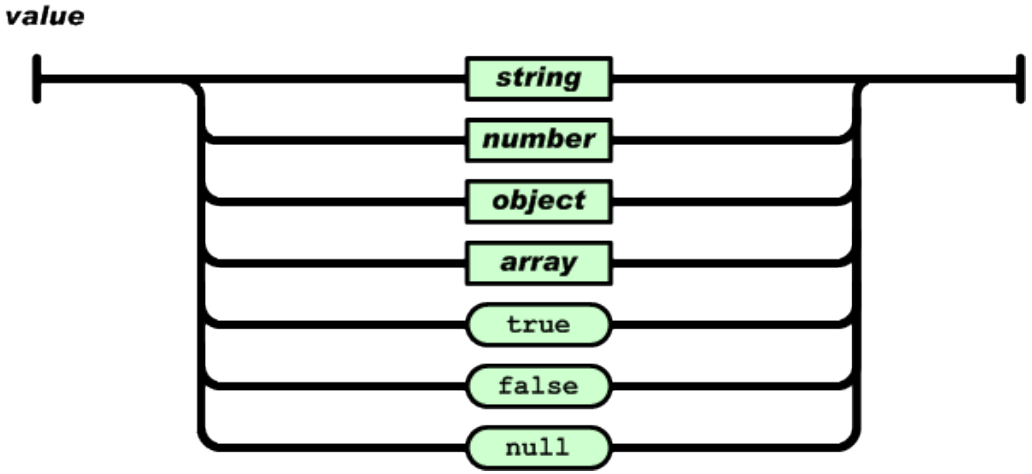



```
{
  "a" : [1, 2, 3]
}

{
  "a" : [1, "txt", true]
}
```

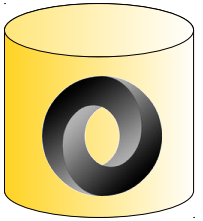


JSON (Syntax)

JSON-Value:

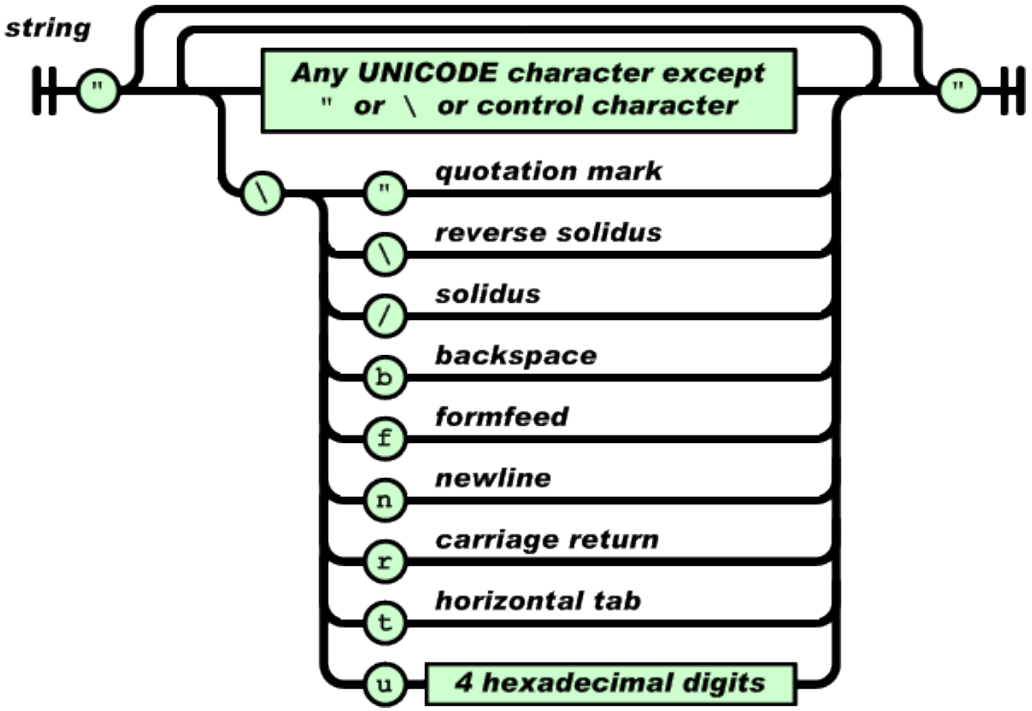



```
{
  "a" : "text",
  "b" : 42,
  "c" : {"c1":10, "c2":11},
  "xy":
    [
      {"x" : 1, "y" : 2},
      {"x" : 3, "y" : 4},
    ],
  "d" : true,
  "e" : null
}
```

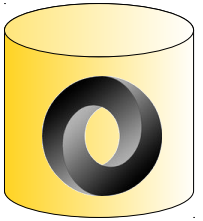


JSON (Syntax)

JSON-String:

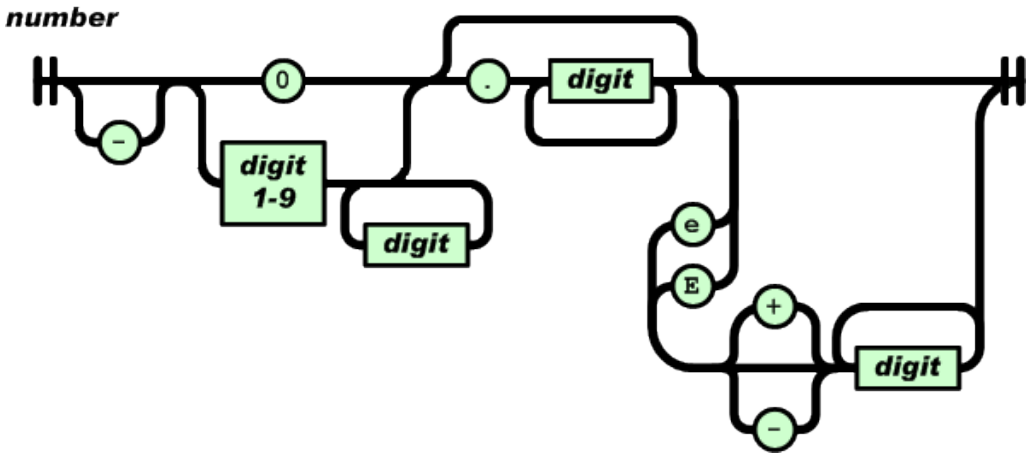



```
{
  "a" : "ein Text",
  "b" : "äöß geht auch",
  "c" : "newline\n",
  "d" : "\"Titel\"",
  "e" : "2019\/03\/16",
  "f" : "c:\\temp\\",
  "g" : "\u1234"
}
```

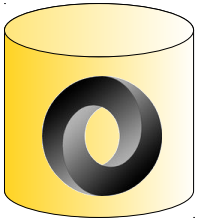


JSON (Syntax)

JSON-Number:




```
{
  "a" : 42,
  "b" : -42,
  "c" : 42.1234,
  "d" : 1.0E+2
  "e" : 1.0E-3
}
```

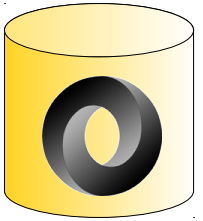


JSON (Beispiel)

sysinfo2mqtt

→ https://github.com/boerge42/mqtt_clients/tree/master/sysinfo2mqtt

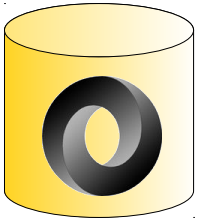
```
{
  "hostname" : "zerow", "uptime"   : "2 days, 2:34:21",
  "load"     : [ "0.32", "0.25", "0.27" ],
  "ram"      : { "free"  : "234832",
                 "share" : "21912",
                 "buffer": "39472",
                 "total" : "492620"
               },
  "swap"     : { "free" : "102396",
                 "total": "102396"
               },
  "processes": "108",
  "time"     : { "unix"    : "1536080941",
                 "readable": "2018\/09\/04 19:09:01"
               }
}
```



JSON (Beispiel)

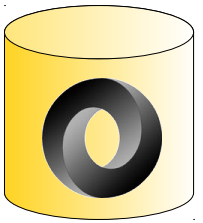
Diverse meiner Sensoren
→ https://github.com/boerge42/nodemcu_scripts

```
{
   "heap":"23216", "temperature":"30.8",
   "humidity":"41.7", "unixtime":"1533665227",
   "node_name":"esp8266-9982412", "node_alias":"Bad",
   "node_type":"dht22", "readable_ts":"2018\/08\/07 20:07:07"
}

{
   "heap":"24354", "temperature":"29.6",
   "humidity":"51.4", "unixtime":"1533665242",
   "pressure_rel":"1023.7", "drewpoint":"14.5",
   "node_name":"esp8266-1283417", "node_alias":"Garage",
   "node_type":"bme280", "readable_ts":"2018\/08\/07 20:07:42"
}
```



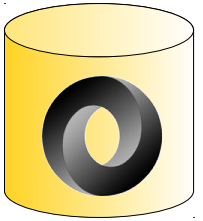
JSON (...für Programmierer..)

- bei einfachen, überschaubaren JSON-Objekten → „zu Fuß“ erzeugen
- JavaScript: JSON-Objekte entsprechen (fast) JavaScript-Objekten!
- diverse Bibliotheken für die gängigen Programmier-sprachen (zum Erzeugen, Parsen, Manipulieren von JSON-Objekten)



SQL → kennt ja jeder...

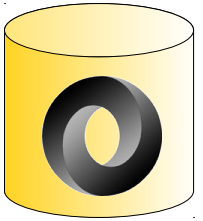
- SQL = **S**tructured **Q**uery **L**anguage
- standardisierte Datenbanksprache zum:
  - Erstellen von relationalen Datenstrukturen
    → CREATE
    → DROP
    → ALTER
    → INDEX
    → TRIGGER
    → ..
  - Abfragen und Bearbeiten der strukturiert abgelegten Daten
    → INSERT, DELETE, UPDATE
    → SELECT, VIEW
    → ...
- → RTFM



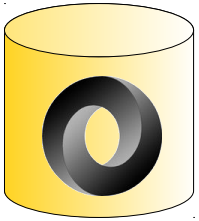
SQL-DBs mit JSON-Support

- „SQL-JSON“ ist (noch kein) SQL-Standard
- SQL-DBs mit JSON-Support:
  - **SQLite**
  - MariaDB
  - MySQL
  - PostgreSQL
  - Oracle
  - MS-SQL
  - ...

JSON-Extension für SQLite



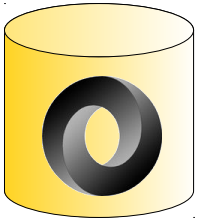
- SQLite akzeptiert JSON-Objekte nach RFC-7159
- Der JSON-Parser ist rekursiv programmiert → es ist eine Verschachtelungstiefe <= 2000 zulässig
- Es sind Funktionen zum:
  → Erzeugen
  → Auslesen
  → Manipulieren
  von JSON-Objekten implementiert, welche alle mit den entsprechenden SQL-Befehlen/-Funktionen kombinierbar sind



JSON-Extension für SQLite

Tabelle für JSON-Daten erzeugen (Beispiel):

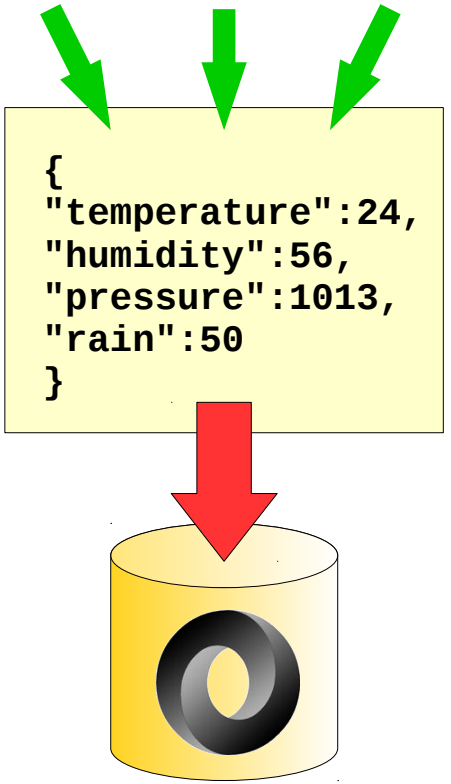
```
create table t (j json);
```

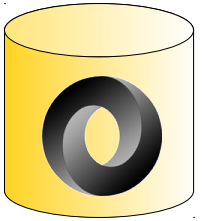


JSON-Extension für SQLite

Funktionen zum Erzeugen/Formatieren von JSON-Objekten:

- json()
- json_object()
- json_array()
- json_quote()

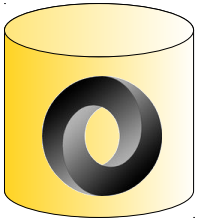




JSON-Extension für SQLite

Tabelle mit JSON-Daten befüllen (einzelne Werte/Arrays):

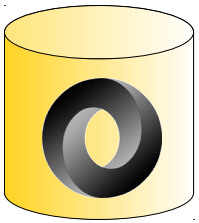
```
# einzelne Werte
insert into t values('{"a":1}');
insert into t values (json_object('a', 2));

# Arrays
insert into t values ('{"b":[1,2,3,4]}');
insert into t values (json_object('b', '[4,5,6,7]'));
insert into t values (json_object('b', json(' [8 , 9 , 10] ')));
insert into t values (json_object('b', json_array(42,43,44)));

# Ergebnis
sqlite> select * from t;
{"a":1}
{"a":2}
{"b":[1,2,3,4]}
{"b":"[4,5,6,7]"}
{"b":[8,9,10]}
{"b":[42,43,44]}
```



JSON-Extension für SQLite

Tabelle mit JSON-Daten befüllen (mehrere Werte):

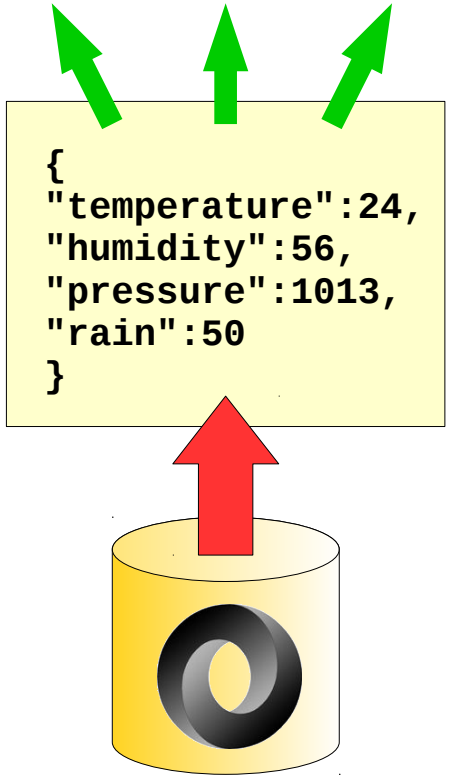
```
# einzelner Wert und ein Array
insert into t values(json_object('a', 42, 'b', json_array(11,22,33)));

# dito, aber ein Member im Array wieder ein JSON-Objekt
insert into t values(
    json_object(
        'a', 42,
        'b', json_array(11,22,33,json_object('c', 77))
    )
);

# Ergebnis
select * from t;
{"a":42,"b":[11,22,33]}
{"a":42,"b":[11,22,33,{"c":77}]}
```



JSON-Extension für SQLite

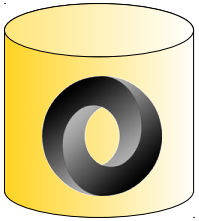
Funktionen zum Abfragen von JSON-Daten:

- json_extract()
- json_array_length()
- json_type()
- json_valid()



PATH-Argument:

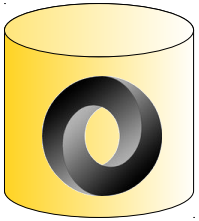
- `**$**...`: referenziert JSON-Objekt (siehe folgende Beispiele)



JSON-Extension für SQLite

Abfrage von JSON-Daten (einfache Werte):

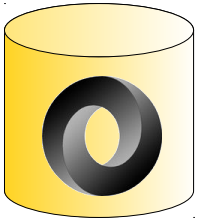
```
# Inhalt der Tabelle
select * from t1;
{"a":1,"b":2,"c":3}
{"a":4,"b":5,"c":5}
{"a":7,"b":8,"c":9}

# Wert a abfragen
select json_extract(j, '$.a') from t1;
1
4
7

# Mittelwert/Maximum von c
select avg(json_extract(j, '$.c')), max(json_extract(j, '$.c')) from t1;
5.66666666666667|9
```



JSON-Extension für SQLite

Abfrage von JSON-Daten (...mit Bedingung, Sortierung):

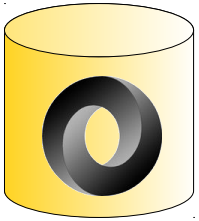
```
# Inhalt der Tabelle
select * from t1;
{"a":1,"b":2,"c":3}
{"a":4,"b":5,"c":5}
{"a":7,"b":8,"c":9}

# mit Bedingung
select json_extract(j, '$.c') from t1 where json_extract(j, '$.c')>=5;
5
9

# Sortierung
select json_extract(j, '$.c') from t1 order by json_extract(j, '$.c') desc;
9
5
3
```



JSON-Extension für SQLite

Abfrage von JSON-Daten („fehlende“ Werte):

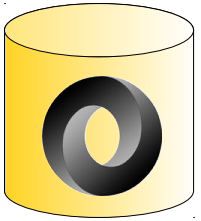
```
# Inhalt der Tabelle
sqlite> select * from t1;
{"a":1,"b":2,"c":3}
{"a":4,"b":5,"c":5}
{"a":7,"b":8,"c":9}
{"a":7,"b":8,"c":9,"d":10}
{"a":7,"b":8,"c":9,"d":42}

# Werte von d
select json_extract(j, '$.d') from t1 where json_extract(j, '$.d') <> '';
10
42

# Mittelwert von d
select avg(json_extract(j, '$.d')) from t1;
26.0
```



JSON-Extension für SQLite

Abfrage von JSON-Daten (Arrays):

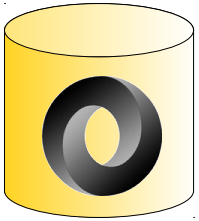
```
# Inhalt der Tabelle
select * from t2;
{"b":[1,2,3,4]}
{"b":[8,9,10]}
{"b":[42,43,44]}

# Array-Werte abfragen
select json_extract(j, '$.b[0]'), json_extract(j, '$.b[3]') from t2;
1|4
8|
42|
```



JSON-Extension für SQLite

Abfrage von JSON-Daten (ein paar „Anregungen“...):

```
# Inhalt der Tabelle
select * from t3;
{"b":[1,2,3,4]}
{"b":"[4,5,6,7]"}
{"b":[8,9,10]}
{"b":[42,43,44]}
{"b":42}

# json_type() (object, array, integer, real, text, true, false, null, NULL)
select json_extract(j, '$.b') from t3 where json_type(j,'$.b') = 'array';
[1,2,3,4]
[8,9,10]
[42,43,44]

# json_array_length()
select json_extract(j, '$.b') from t3 where json_array_length(j,'$.b') > 3;
[1,2,3,4]
```

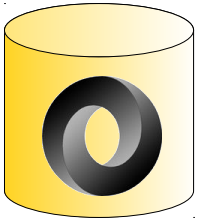


JSON-Extension für SQLite

Zuviel Schreibarbeit? Statements zu unübersichtlich?
→ **VIEWs** definieren

```
# Select mit json_...
select json_extract(j, '$.a') as a from t
where json_extract(j, '$.a') <> '';

# VIEW erzeugen
create view view_t as
select json_extract(j, '$.a') as a from t
where json_extract(j, '$.a') <> '';

# VIEW anwenden
select a from view_t;
```

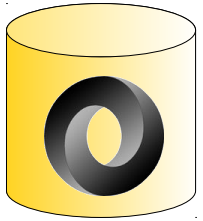
JSON-Extension für SQLite



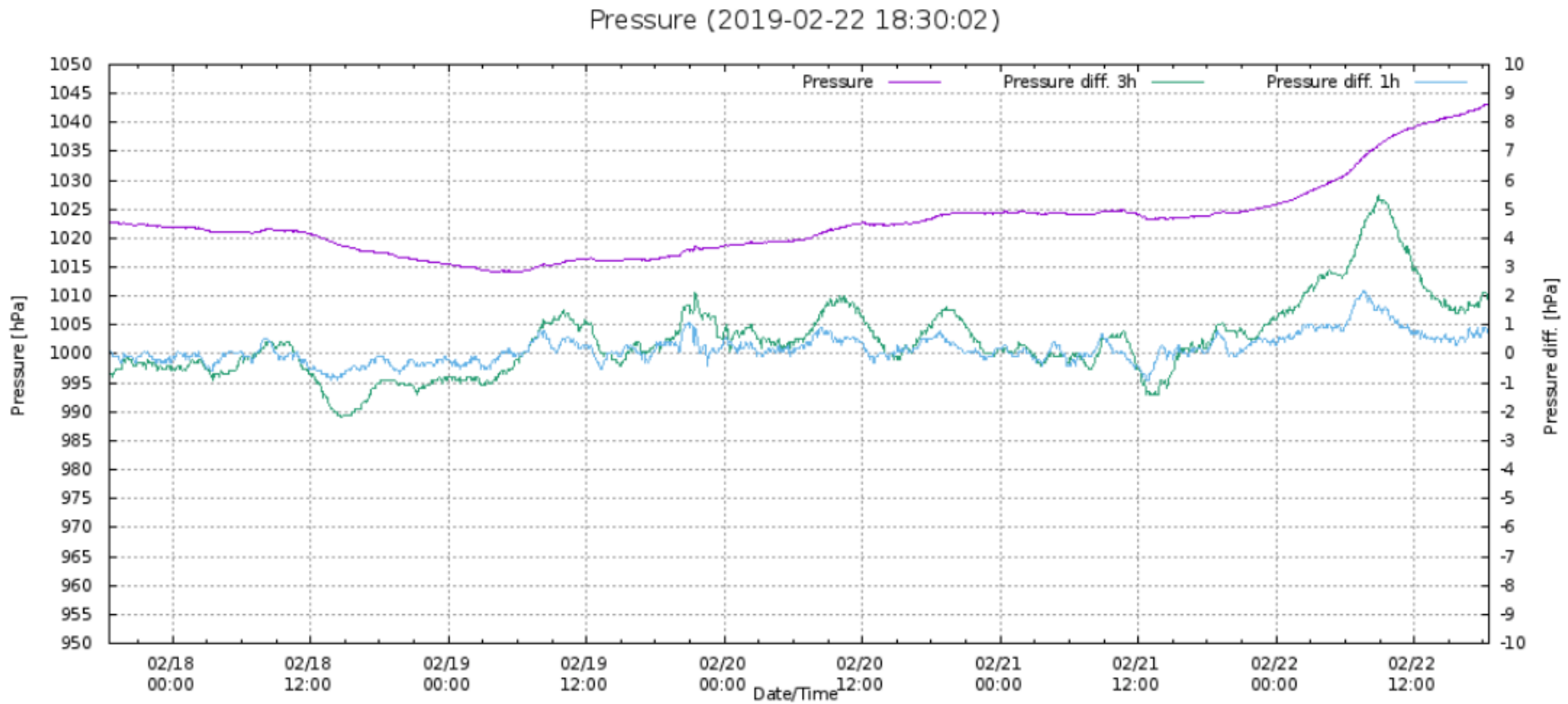
Inhalte von JSON-Objekten können auch Trigger auslösen:

```
# Tabelle pressure...
CREATE TABLE pressure (unixtime integer, nodename text, pressure real,
    pressure_diff3h real, pressure_diff1h real);

CREATE TRIGGER trigger_pressure after insert on sensors
when (json_extract(new.d, '$.pressure_rel') <> '')
begin
    insert into pressure values (
        json_extract(new.d, '$.unixtime'),
        json_extract(new.d, '$.node_name'),
        json_extract(new.d, '$.pressure_rel'),
        (select avg(pressure) from pressure
            where nodename=json_extract(new.d, '$.node_name') and
                unixtime/300*300=
                    ((json_extract(new.d, '$.unixtime')-10800)/300*300)),
        (select avg(pressure) from pressure
            where nodename=json_extract(new.d, '$.node_name') and
            unixtime/300*300=
                ((json_extract(new.d, '$.unixtime')-3600)/300*300))
);
```

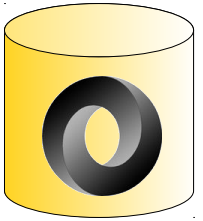
JSON-Extension für SQLite



Inhalte von JSON-Objekten können auch Trigger auslösen:



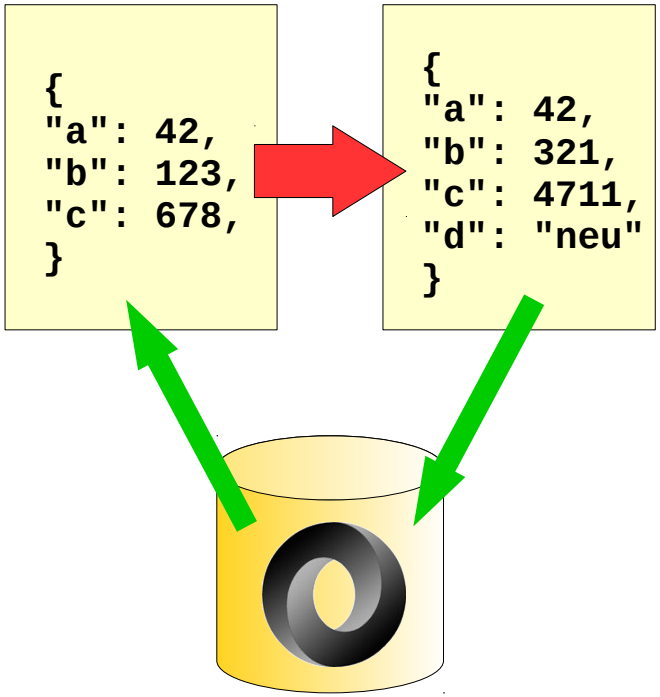

```
);
```

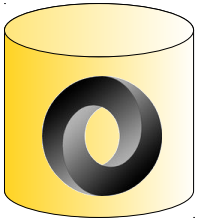


JSON-Extension für SQLite

JSON-Objekte in Tabelle manipulieren:

- json_insert()
- json_replace()
- json_set()
- json_patch()
- json_remove()



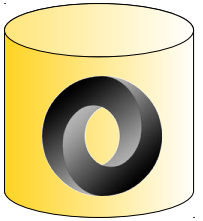


JSON-Extension für SQLite

JSON-Objekte in Tabelle manipulieren:

```
# json_insert(), json_replace(), json_set()
select json_insert('{"a":42}', '$.b', 43);
{"a":42,"b":43}

select json_replace('{"a":42}', '$.a', 23);
{"a":23}

select json_set('{"a":42}', '$.b', json_array(1,2,3,4));
{"a":42,"b":[1,2,3,4]}

# json_patch()
select json_patch('{"a":1,"b":2}','{"a":9,"b":null,"c":8}');
{"a":9,"c":8}


# json_remove()
select json_remove('{"b":[1,2,3,4,5,6]}', '$.b[4]', '$.b[2]');
{"b":[1,2,4,6]}
```

JSON-Extension für SQLite

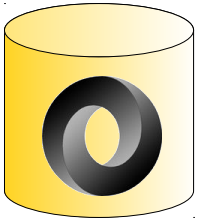


Weitere JSON-Funktionen:

- Umwandlung von Abfrageergebnissen in JSON-Objekte
  - json_group_array()
  - json_group_object()
- „Zeilenweises“ Abarbeiten von einzelnen JSON-Werten innerhalb eines JSON-Objektes:
  - json_each()
  - json_tree()

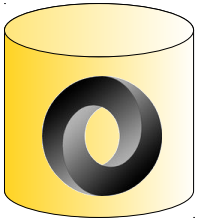
→ RTFM

JSON-Extension für SQLite



Performance:

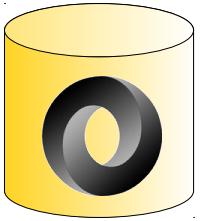
- JSON-Parser (sqlite) $\rightarrow$ 300MByte/s JSON-Text
- Natürlich kann man JSON-Objekte indizieren...!



JSON-Extension für SQLite

Performance-Experiment:

```
# JSON-Objekt
{
  "hostname" : "zerow", "uptime"   : "2 days, 2:34:21",
  "load"     : [ "0.32", "0.25", "0.27" ],
  "ram"      : { "free"  : "234832", "share" : "21912",
               "buffer": "39472", "total" : "492620" },
  "swap"     : { "free" : "102396", "total": "102396"  },
  "processes": "108",
  "time"     : { "unix"    : "1536080941",
               "readable": "2018\/09\/04 19:09:01" }
}

# Anzahl Datensaetze in Tabelle
select count(*) from computers;
1862728
```

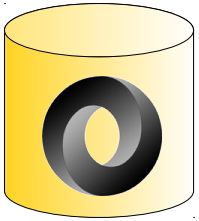


JSON-Extension für SQLite

Performance-Experiment (Ergebnisse):

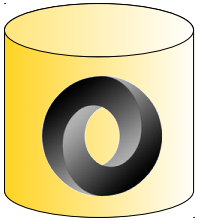
```
# Anzahl nach hostname gruppiert
select json_extract(d, '$.hostname'), count(*) from computers
    group by json_extract(d, '$.hostname');

# Verarbeitungszeit ohne Index:
# 0|0|0|SCAN TABLE computers
# 0|0|0|USE TEMP B-TREE FOR GROUP BY

==> 0m2,448s

# Index ueber hostname:
create index computers_hostname on computers
    (json_extract(d, '$.hostname'));

# 0|0|0|SCAN TABLE computers USING INDEX computers_hostname

==> 0m0,839s
```

JSON-Extension für SQLite



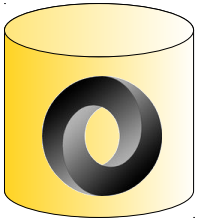
Performance-Experiment („Gegenprobe“):

```
# Umladen der JSON-Objekte in eine „normale“ SQL-Tabelle
create table computers1 (timestamp integer, hostname text, uptime text);

insert into computers1 select json_extract(d, '$.time.unix'),
  json_extract(d, '$.hostname'),json_extract(d, '$.uptime') from computers;

==> 0m2,000s

# Anzahl Eintraege nach hostname gruppiert:
select hostname, count(*) from computers1 group by hostname;

## ohne Index
==> 0m0,549s

## mit Index
create index computers1_hostname on computers1 (hostname);
==> 0m0,091s
```



Weiterführende Informationen

- https://json.org/
- https://sqlite.org/
- https://sqlite.org/json1.html



Fragen...?