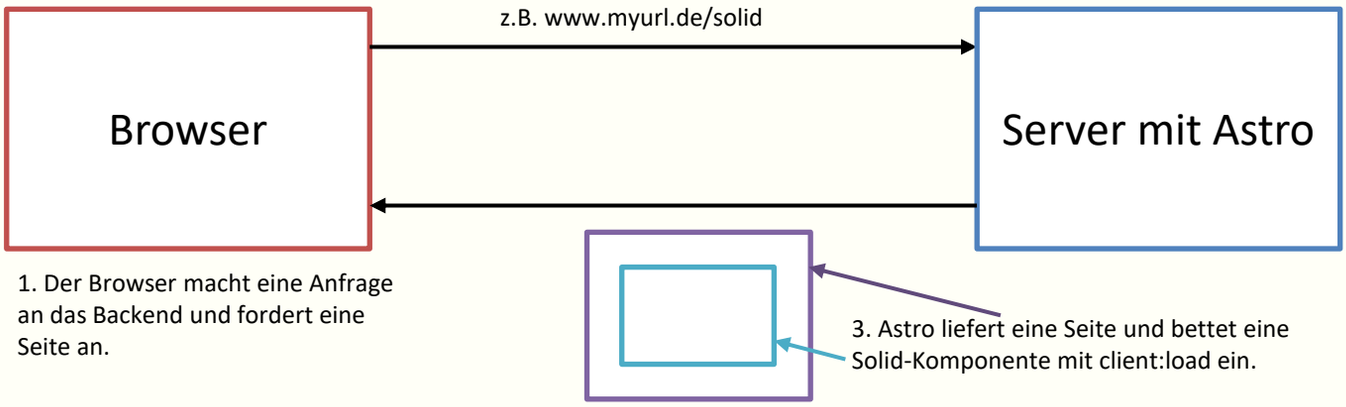
HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

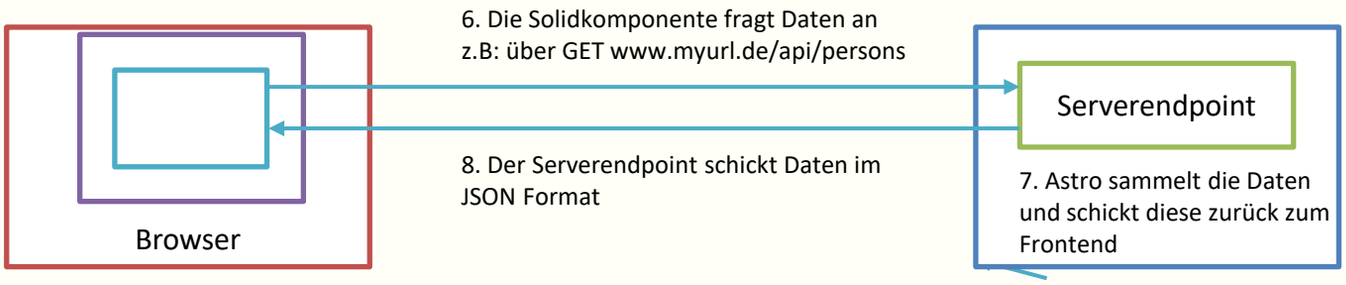
REST, Interaktion Frontend-Backends

Prof. Dr.-Ing. Thorsten Weiss

Version 1.2

Grundlegendes Konzept





Es gibt 3 Möglichkeiten, Daten in die Anfrage zu packen.

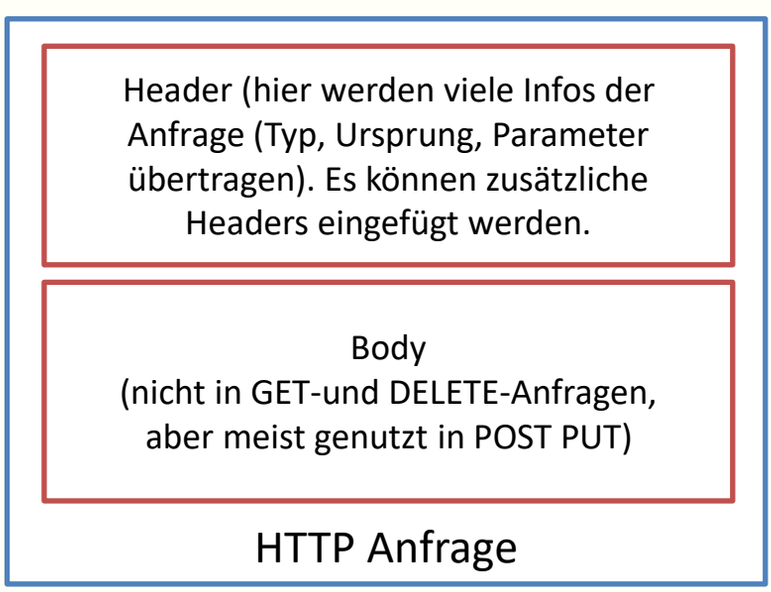
1. In die URL (params)

www.myurl.de/api/?a=1;b=2;c=3 ← Variante mit Variablennamen

www.myurl.de/api/1/2/3 ← Variante mit Routen

2. Headers

3. Body



Kommentiertes Beispiel:
Frontend: 2_solidPersonSimple
Backend: /api/personsimple.ts

REST / JSON

- REST (**Representational State Transfer)** ist ein Programmiervorgehen / **Regelwerk** zum Austausch von Daten zwischen einem Client und einem Server.
- Der Datenaustausch ist **zustandslos**. Jede Anfrage muss alle Informationen enthalten, die zum Abarbeiten vom Server nötig sind.
- REST arbeitet **asynchron** und wird dabei häufig mit AJAX (asynchroner Datenaustausch zwischen Browser und Server) in Verbindung gebracht. Eine Anwendung kann also parallel mehrere Anfragen an den Server schicken.

- Bei WebApps ist die Idee, funktionierende Anwendungen vollständig im Browser zu laden.
- **Zustandsänderungen, Eingaben etc. die auf dem Server gespeichert oder geladen** werden sollen, werden mit **kleinen Datenpaketen** ausgetauscht. Dadurch muss die Seite nicht ständig neu geladen werden, sondern nur kleine Datenpakete. Einzelne Informationen auf der Seite werden aktualisiert.
- Falls die WebApp keine Kommunikation mit dem Server benötigt, wird sie einmalig geladen und es erfolgt keine weitere Kommunikation. Beispiel: App, die °C in Fahrenheit umrechnet oder Meter in Inch → dies kann ohne Serverkommunikation umgesetzt werden.

Übermittlung der Daten via JSON (ggf. XML oder binär (z.B. für Bilder etc.)

Serverseitiges Skript/Service,
z.B. PHP, Java, C#,...

Unidirectional:
iPhone lädt Daten hoch.
iPhone fragt an und lädt Daten herunter (REST)

Bidirectional:
Upload/Download (Sockets)

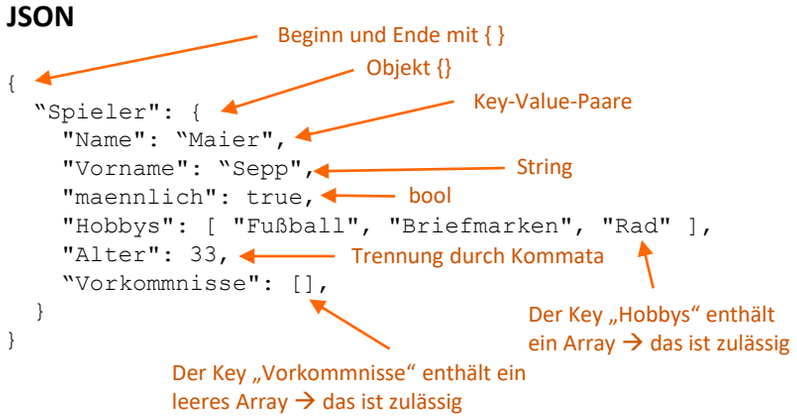
- Das übliche Austauschformat ist HTML in Verbindung mit **JSON**. Vor allem in der Microsoft-Welt wird oftmals mit XML gearbeitet.
- **REST unterscheidet 4 Anfragetypen:**

GET	POST	PUT	DELETE
Datenabrufen	Neuer Datensatz ablegen	Datensatz aktualisieren.	Datensatz löschen

Get kann sowohl ganze Seiten liefern als auch JSON Pakete

- Im Javascript (JS)-Code im Client werden Anfragen gestartet.
- Nach **asynchronem Empfang** werden die Ergebnisse im JS verarbeitet

JSON



JSON - Folgende Datentypen sind zulässig:

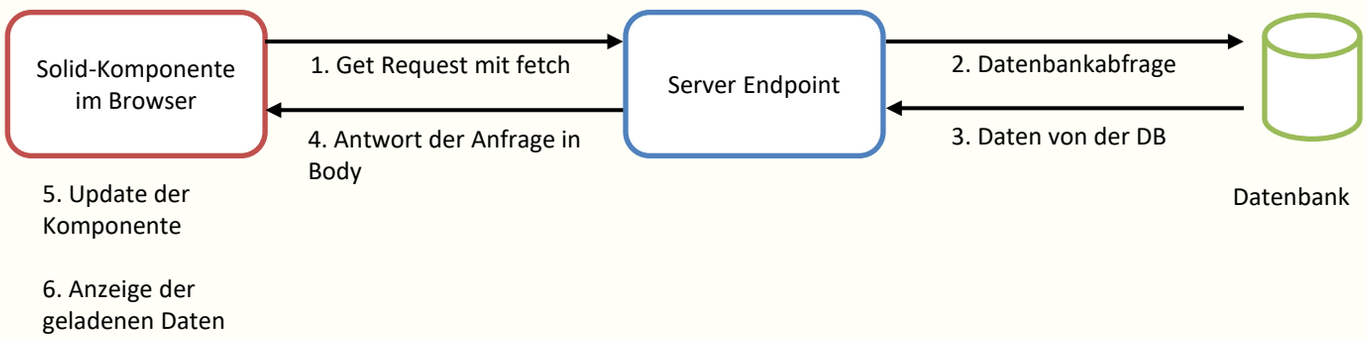
- **Nullwert:** null
- **Boolscher Wert:** true/false
- **Zahl:** double / int
- **String:** Beginnt und endet mit ".
- **Array:** beginnt mit [und endet mit].
  - Elemente werden durch Kommata getrennt.
  - Arrays können Werte oder Objekte gleichen oder verschiedenen Typs im selben Array enthalten.
  - Auch leere Arrays sind erlaubt.
- **Objekt** beginnt mit { und endet mit }.
  - Enthält eine ungeordnete Liste von Eigenschaften, die durch Kommata getrennt werden.
  - Eigenschaften bestehen aus einem Schlüssel (ein String) und einem Wert (Key-Value)
  - Schlüssel müssen eindeutig sein → nur einmal enthalten.
  - Objekte ohne Eigenschaften („leere Objekte“) sind zulässig.

Astro - Endpoints

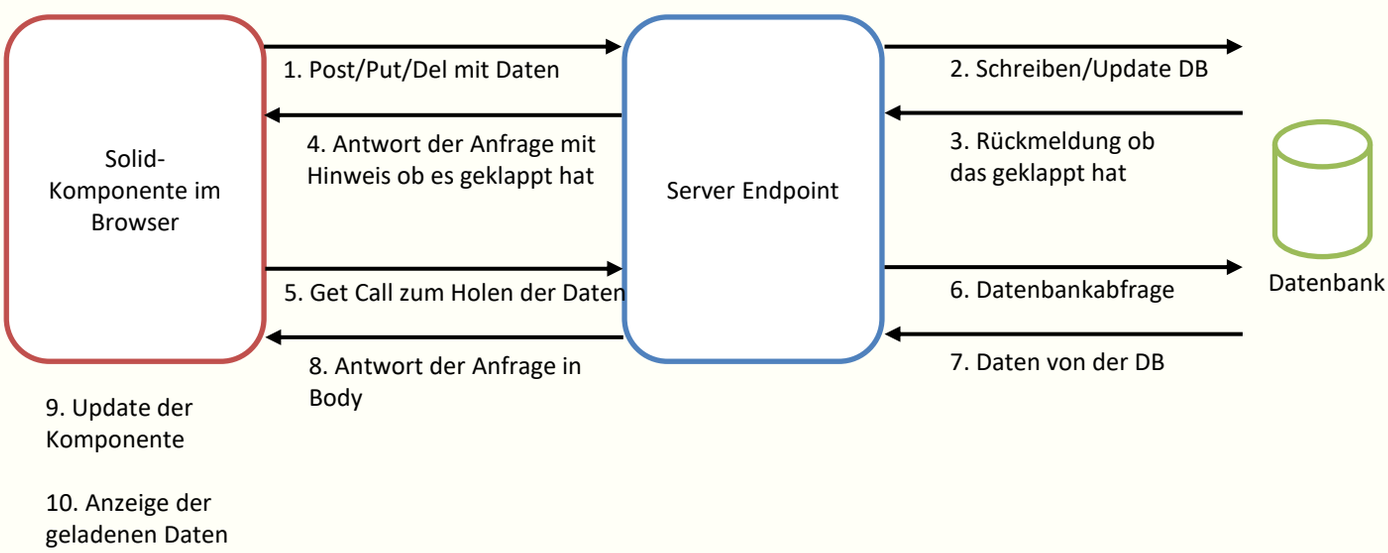
- Siehe Video in Moodle

Cloudservice

- **Ablauf eines Get-Calls:**



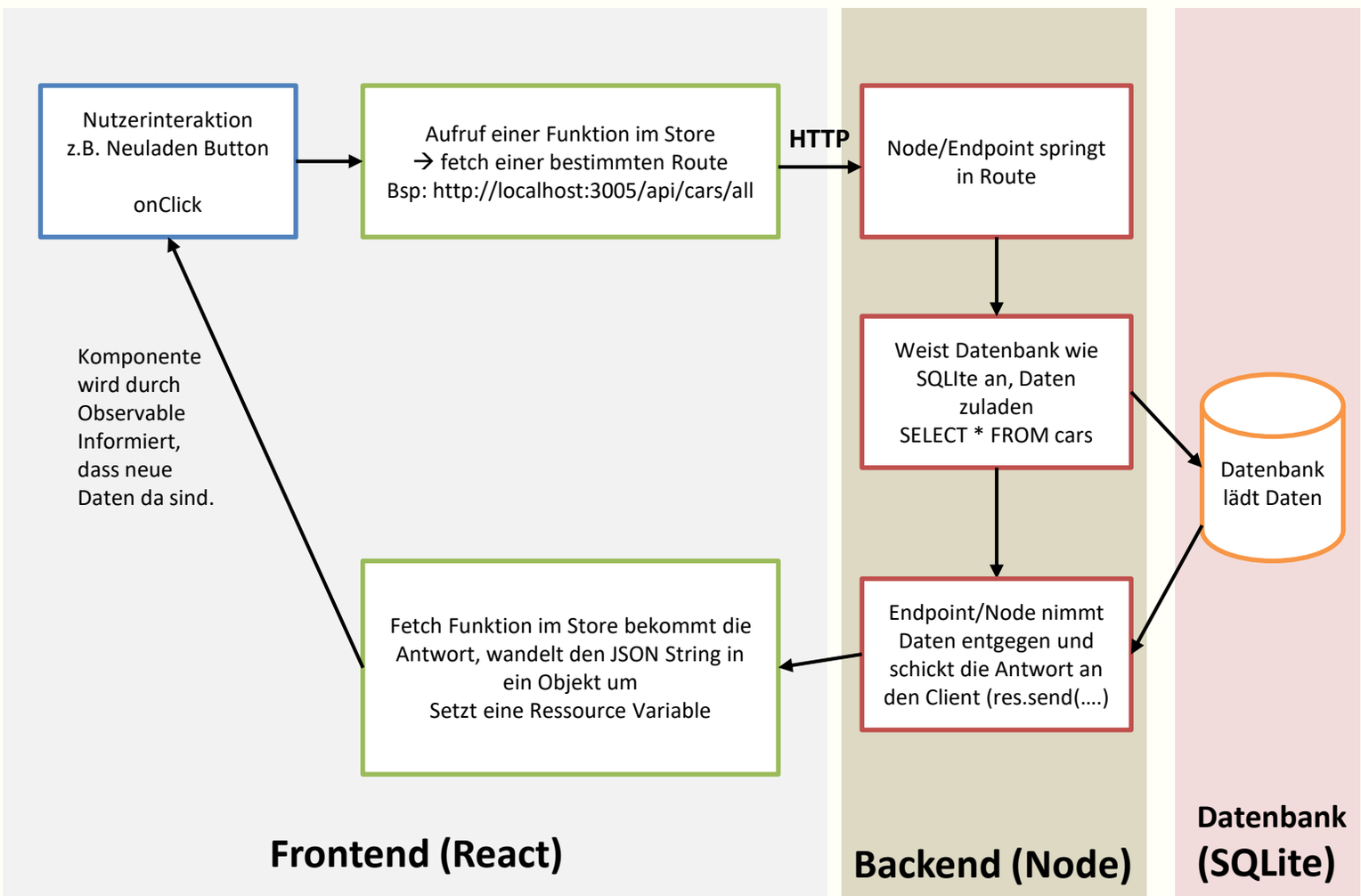
- **Ablauf eines Post/Put/Delete-Calls**



Cloud Computing mit React / Node

Eine Datenaktion im Cloudcomputing durchläuft folgende Phasen

- Nutzerinteraktion in der Komponente triggert einen http-Call (fetch)
- Der Server bearbeitet die Anfrage (get, post, put, delete) und führt eine Aktion auf der Datenbank aus (select, insert, update, delete)
- Danach werden die Daten an den Client zurückgesendet.
- Die Komponenten, die sich für die Daten „interessieren“ können die Anzeige updaten.



Siehe Inventory App

Datenbanken – SQLite

- SQLite ist eine leichtgewichtige Datenbank, die zur Speicherung von wenigen Daten lokal geeignet ist. Für große Datenmengen sollte ein Datenbankserver wie mySQL/postgresSQL, Oracle etc. genutzt werden.
- Installation:

```
npm i sqlite3 --save
```

- Verbindung zur Datenbank herstellen in routes:

```
let db = new sqlite3.Database(dbPath, (err) => {
  if (err) {
    console.error(err.message);
  }
  initDatabase();
  console.log('Connected to the cars database.');
});
```

Verbindung zu SQLite

```
function initDatabase() {
  // eine Tabelle erzeugen, falls diese noch nicht existiert.
  let query = "CREATE TABLE IF NOT EXISTS cars ( id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL DEFAULT '',
            type TEXT NOT NULL DEFAULT '',
            fin TEXT NOT NULL DEFAULT '',
            powerKw INTEGER NOT NULL DEFAULT 0)";
  db.run (query);
}
```

SQL-Query: Tabelle erzeugen, falls diese noch nicht existiert

Ausführen

- Zugriff mit SQL Befehlen

```
router.get('/all', function(req, res, next) {
  let query = "SELECT * FROM cars";
  db.all(query, [], (err, rows) => {
    if (err) {
      res.send( err.message );
    }
    console.log(rows);
    res.send( { cars: rows });
  });
});
```

Alle Elemente der Tabelle cars laden

Datenmodell – 1:1, 1:n (1)

- Für den Entwurf eines Datenmodells sollte genug Zeit und Sorgfalt investiert werden.
- Es hat sich gezeigt, dass eine **einheitliche Namensgebung** von Spalten und Tabellen sehr viel Ärger, Zeit und Missverständnisse spart. Daher sollten die Namenskonventionen zu Beginn festgelegt werden, bevor die erste Zeile programmiert wird.

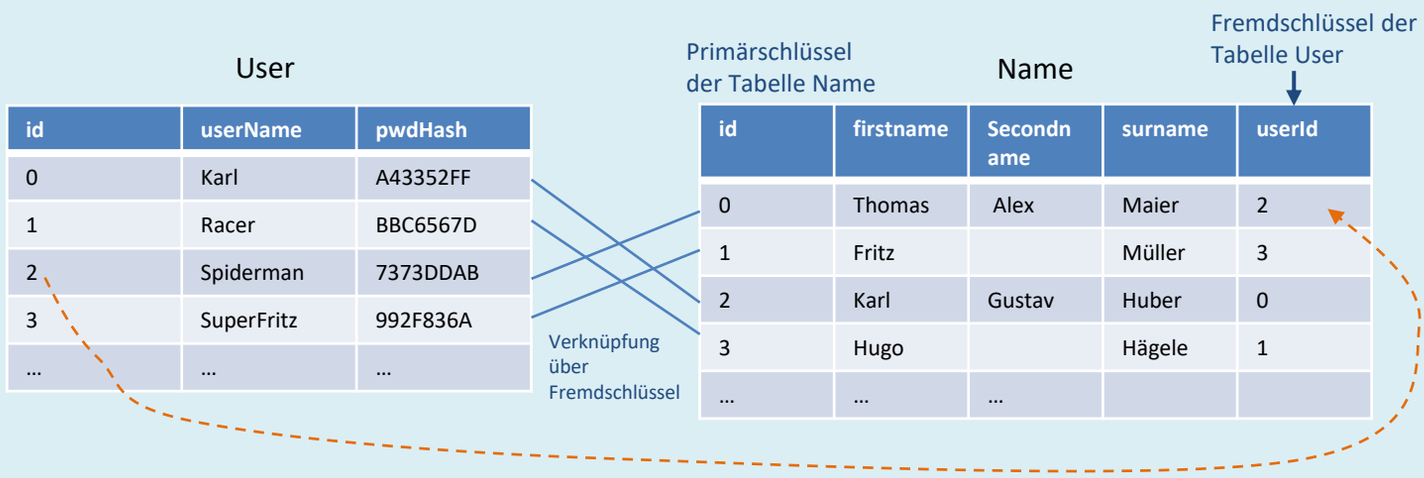
  Beispiele für Regeln:
  - Keine Underscores (MyTable statt my_table)
  - Alle Bezeichnungen auf Englisch
  - Alle Tabellennamen UpperCamelCase (MyTable)
  - Abkürzungen einheitlich (no = number of → noCalls, ms = measurement → msSensorTempC)
  - Alle Spaltennamen lowerCamelCase (userName, adress, noCalls)
  - Alle Spalten, die eine physikalische Größe beinhalten, die Einheit anhängen. (velWheelFrontMs: Geschwindigkeit vorne in m/s, sensorTempOuterC: Außentemperatur in Celsius).
  - Alle Primärschlüssel heißen „id“.
  - Alle Fremdschlüssel heißen tableNameId (partId, carId)
  - usw…

- Das Speichern und Abrufen von einfachen Tabellen ist auf den letzten Seiten gezeigt. Sequelize wandelt Tabellen in Javaskript Dateien direkt um.

Verknüpfungen von Tabellen

- Tabellen können verknüpft werden, um eine Beziehung (Relationship) darzustellen. Hierbei unterscheiden wir drei Verknüpfungsarten:

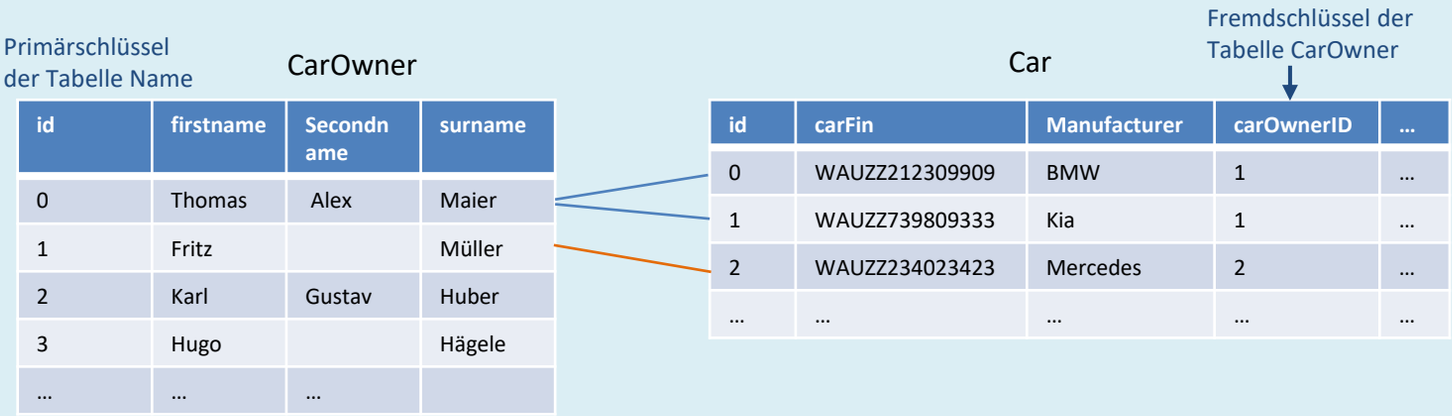
- **1:1 Verknüpfung:** Eine Zeile in einer Tabelle gehört zu einer Zeile in einer anderen Tabelle.
  Beispiel: Ein Tabelle User und eine Tabelle Name ist 1:1 verknüpft



id	userName	pwdHash
0	Karl	A43352FF
1	Racer	BBC6567D
2	Spiderman	7373DDAB
3	SuperFritz	992F836A
…	…	…

id	firstname	Secondname	surname	userId
0	Thomas	Alex	Maier	2
1	Fritz		Müller	3
2	Karl	Gustav	Huber	0
3	Hugo		Hägele	1
…	…	…		

- **1:n Verknüpfung:** Eine Zeile in einer Tabelle kann zu vielen Zeilen in einer anderen Tabelle gehören.
  Beispiel: Ein User kann Halter mehrerer Autos sein. Aber jedes Auto hat nur einen Halter:



id	firstname	Secondname	surname
0	Thomas	Alex	Maier
1	Fritz		Müller
2	Karl	Gustav	Huber
3	Hugo		Hägele
…	…	…	

id	carFin	Manufacturer	carOwnerID	…
0	WAUZZ212309909	BMW	1	…
1	WAUZZ739809333	Kia	1	…
2	WAUZZ234023423	Mercedes	2	…
…	…	…	…	…

Datenmodell – n:m, Bilder (2)

- **n:m Verknüpfung:** Eine Zeile in einer Tabelle gehört zu mehreren Zeilen in einer anderen Tabelle.
  Beispiel: Warenkorb in einem Shopsystem: Ein User kann mehrere Waren in seinem Einkaufswagen haben. Dieselbe Ware kann bei vielen Nutzern in einem Warenkorb liegen.
- In diesem Fall muss eine Zwischentabelle eingefügt werden, die diese Verknüpfung abbildet.

User

id	userName	pwdHash
0	Karl	A43352FF
1	Fritz	BBC6567D
2	Walter	7373DDAB
3	Francis	992F836A
…	…	…

UserArticle

id	idUser	idArticle
0	0	2
1	0	3
2	3	3
3	3	1
…	…	…

Primärschlüssel | Fremdschlüssel der Tabelle User | Fremdschlüssel der Tabelle Article

Article

id	name	price	orderNumber	…
0	shampoo	3.49	29308714	…
1	soap	1.99	12312312	…
2	showergel	2.99	33453252	…
3	aftershave	9.99	53345763	…
…	…	…		

Karl hat showergel und aftershave in seinem Warenkorb.
Francis hat aftershave und soap in seinem Warenkorb.
Fritz und Walter haben nichts in ihrem Warenkorb.

- Es können auch mehrere Verknüpfungen pro Tabelle vorhanden sein. So könnte die Tabelle User als 1:1 Verknüpfung mit einer Tabelle Payment 1:1 verknüpft sein, in der Zahlungsinfos enthalten sind. Weiterhin könnte der User 1:n mit Adress verknüpft sein, um mehrere Adressen pro Nutzer zu speichern.

- **Das Datenmodell mit Hilfe von Tabellen kann somit die gängigen Verknüpfungen abbilden. Datenmodelle werden daher so gestaltet, dass die Problemstellung mit diesen Tabellen abgebildet werden kann.**

Speichern von Massendaten wie Bildern

- Eine SQL Datenbank ist nicht dafür vorgesehen, Bilder, Dateien, sprich große Daten zu speichern.
- **Daher kann ein Link, ein Ort oder ein Hinweis auf einen Webspace angegeben wird.**
- Beispiel: Bilder einer Galerie sollen gespeichert werden:

Images

id	Name	Link	…
0	Rose	/img/rose.jpg	…
1	Tree	/img/tree.png	…
2	Flower	/img/flo.jpg	…
3	Leaf	/img/23814412.jpg	…
…	…	…	

SQL Queries

- SQL ist eine eigene Skript-Sprache für Datenbankabfragen. In den Seeds sind viele Beispiele zu finden.
- Abfragen sollten mit Parametern ausgeführt sein.

Mit Joins können Tabellen miteinander kombiniert werden.

SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE  c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE  c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING TABLES

```
CREATE TABLE t (
   id INT PRIMARY KEY,
   name VARCHAR NOT NULL,
   price INT DEFAULT 0
);
```
Create a new table with three columns

```
DROP TABLE t;
```
Delete the table from the database

```
ALTER TABLE t ADD column;
```
Add a new column to the table

```
ALTER TABLE t DROP COLUMN c;
```
Drop column c from the table

```
ALTER TABLE t ADD constraint;
```
Add a constraint

```
ALTER TABLE t DROP constraint;
```
Drop a constraint

```
ALTER TABLE t1 RENAME TO t2;
```
Rename a table from t1 to t2

```
ALTER TABLE t1 RENAME c1 TO c2;
```
Rename column c1 to c2

```
TRUNCATE TABLE t;
```
Remove all data in a table

USING SQL CONSTRAINTS

```
CREATE TABLE t(
   c1 INT, c2 INT, c3 VARCHAR,
   PRIMARY KEY (c1,c2)
);
```
Set c1 and c2 as a primary key

```
CREATE TABLE t1(
   c1 INT PRIMARY KEY,
   c2 INT,
   FOREIGN KEY (c2) REFERENCES t2(c2)
);
```
Set c2 column as a foreign key

```
CREATE TABLE t(
   c1 INT, c1 INT,
   UNIQUE(c2,c3)
);
```
Make the values in c1 and c2 unique

```
CREATE TABLE t(
 c1 INT, c2 INT,
 CHECK(c1> 0 AND c1 >= c2)
);
```
Ensure c1 > 0 and values in c1 >= c2

```
CREATE TABLE t(
   c1 INT PRIMARY KEY,
   c2 VARCHAR NOT NULL
);
```
Set values in c2 column not NULL

MODIFYING DATA

```
INSERT INTO t(column_list)
VALUES(value_list);
```
Insert one row into a table

```
INSERT INTO t(column_list)
VALUES (value_list),
       (value_list), ....;
```
Insert multiple rows into a table

```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```
Insert rows from t2 into t1

```
UPDATE t
SET c1 = new_value;
```
Update new value in the column c1 for all rows

```
UPDATE t
SET c1 = new_value,
    c2 = new_value
WHERE condition;
```
Update values in the column c1, c2 that match the condition

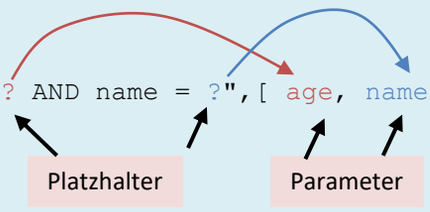
```
DELETE FROM t;
```
Delete all data in a table

```
DELETE FROM t
WHERE condition;
```
Delete subset of rows in a table

SQL Queries (Advanced)

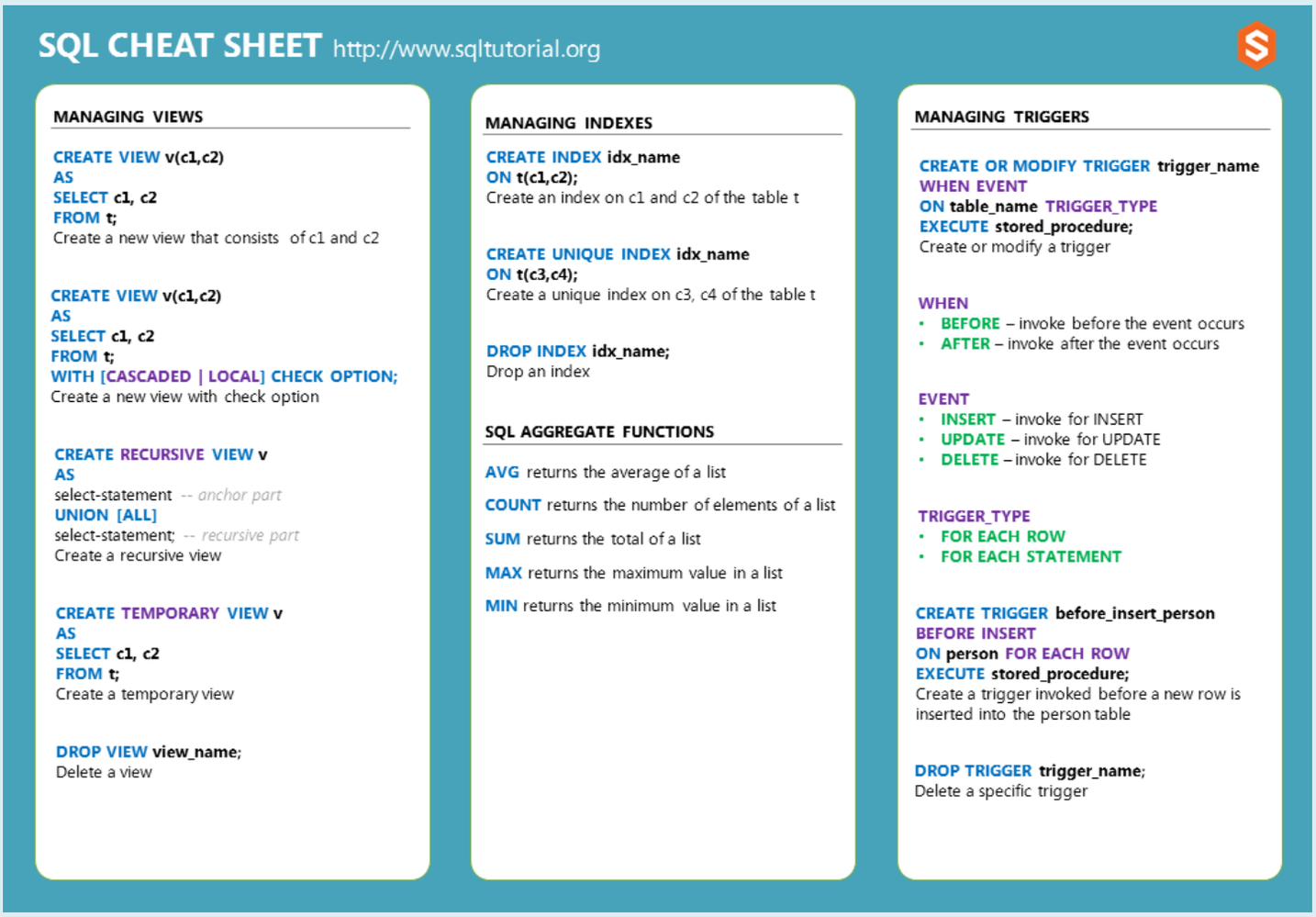
- SQL Queries immer mit Parameter ausführen anstatt mit String Operationen (verhindert viele SQL Injections)

```
let name = req.body.name || '';
let age = req.body.age || 0;
db.query("SELECT * FROM users WHERE age > ? AND name = ?",[ age, name ] …
```



Views

- Ein View ist eine Sicht auf eine andere Datenbank. Daten können so zwischen SQL-Datenbanken getauscht, bzw. miteinandern verknüpft werden.



SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING VIEWS

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
```
Create a new view that consists of c1 and c2

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
WITH [CASCADED | LOCAL] CHECK OPTION;
```
Create a new view with check option

```
CREATE RECURSIVE VIEW v
AS
select-statement -- anchor part
UNION [ALL]
select-statement; -- recursive part
```
Create a recursive view

```
CREATE TEMPORARY VIEW v
AS
SELECT c1, c2
FROM t;
```
Create a temporary view

```
DROP VIEW view_name;
```
Delete a view

MANAGING INDEXES

```
CREATE INDEX idx_name
ON t(c1,c2);
```
Create an index on c1 and c2 of the table t

```
CREATE UNIQUE INDEX idx_name
ON t(c3,c4);
```
Create a unique index on c3, c4 of the table t

```
DROP INDEX idx_name;
```
Drop an index

SQL AGGREGATE FUNCTIONS

AVG returns the average of a list

COUNT returns the number of elements of a list

SUM returns the total of a list

MAX returns the maximum value in a list

MIN returns the minimum value in a list

MANAGING TRIGGERS

```
CREATE OR MODIFY TRIGGER trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE
EXECUTE stored_procedure;
```
Create or modify a trigger

WHEN
- **BEFORE** – invoke before the event occurs
- **AFTER** – invoke after the event occurs

EVENT
- **INSERT** – invoke for INSERT
- **UPDATE** – invoke for UPDATE
- **DELETE** – invoke for DELETE

TRIGGER_TYPE
- **FOR EACH ROW**
- **FOR EACH STATEMENT**

```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
```
Create a trigger invoked before a new row is inserted into the person table

```
DROP TRIGGER trigger_name;
```
Delete a specific trigger