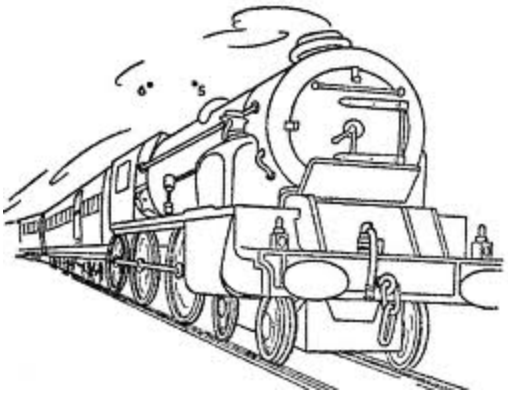
Übung Datenbanken: Views

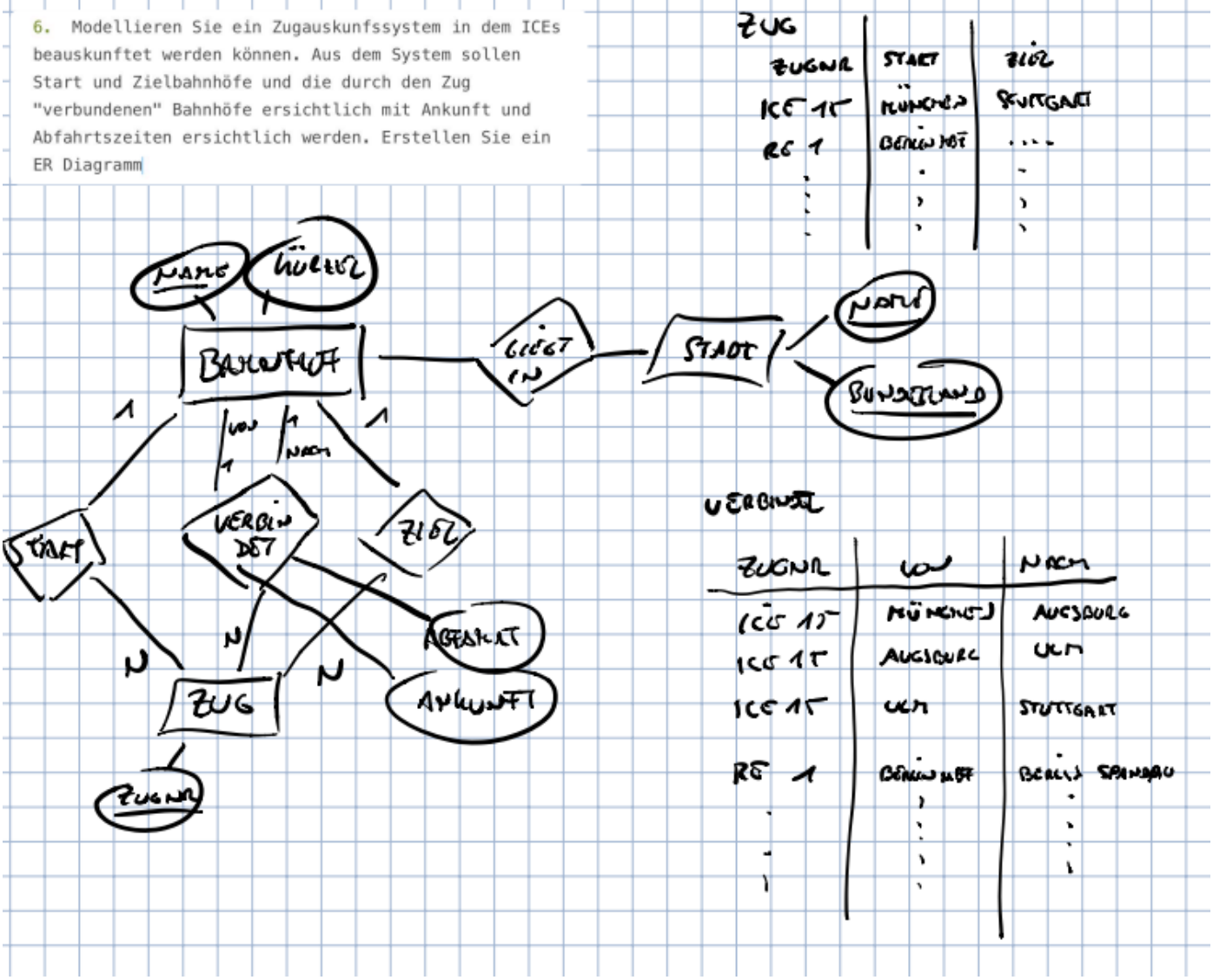


In [1]:
```
%%sh

#rm -f opendata.sql zugauskunft.db
#wget -qOopendata.sql 'https://nextcloud.th-deg.de/s/Kc7m486Z3KbZyB4/download?path=%2F&fi
#cat ./opendata.sql | sqlite3 zugauskunft.db

wget -qObahn-opendata.db 'https://nextcloud.th-deg.de/s/Kc7m486Z3KbZyB4/download?path=%2F
cp bahn-opendata.db zugauskunft.db
```

In [2]:
```
# Hier ist nur Code zum Initialisieren der Umgebeung, bitte gehen Sie weiter, es gibt nich

# Keine langen Fehlermeldungen
import sys
ipython = get_ipython()

def exception_handler(exception_type, exception, traceback):
    print("%s: %s" % (exception_type.__name__, exception), file=sys.stderr)

ipython._showtraceback = exception_handler

# Lade die Erweiterung, damit wir SQL Befehle nutzen können
%reload_ext sql
%sql sqlite:///zugauskunft.db
```

Über das Importieren von Daten

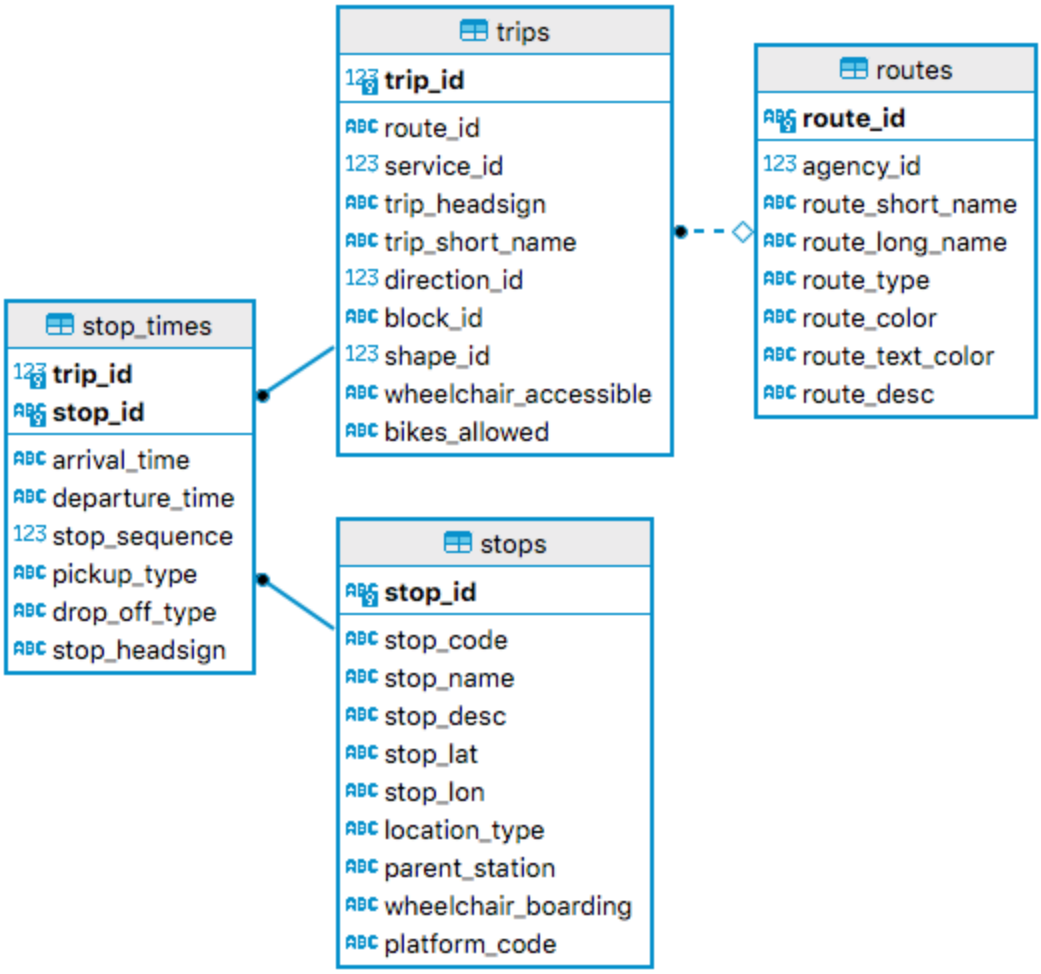
Wir erinnern uns an die Aufgabenstellung aus dem Übungsblatt. Wir verwenden hier folgende Modellierung.



Eine Übung lebt ja davon, dass man ein paar Daten zum rumspielen hat. Daher habe ich Daten habe aus einem bundesweiten Opendataportal genommen. Wir sollten damit in der Lage sein echte Verbindungen nachzuvollziehen. Ich habe den Datensatz auf die Waldbahn beschränkt. Der Datensatz selbst ist im CSV Format (!), welches aber von SQLite importiert werden kann.

Die Daten selbst habe ich Ihnen bereits gefiltert und importiert (selbst der bayernweite Datensatz ist recht sperrig), nun geht es darum einige Views zu erstellen.

Die Quellstruktur der GTFS Daten ist wie folgt abgebildet:



Aufgabe 1

Aufgabe 1.1

Betrachten wir erst einmal die gegebenen Daten. Wir beginnen mit der `routes` Tabelle, in der alle Routen, die von einem Betreiber (Agency) bedient werden abgelegt sind. Der vorliegende Datensatz beschränkt sich auf die Waldbahn, welche die Agency ID 10964 hat. Fragen Sie daher alle Daten aus dieser Tabelle ab.

In [3]:
```
%sql SELECT * FROM routes;
```

```
 * sqlite:///zugauskunft.db
Done.
```

Out[3]:

route_id	agency_id	route_short_name	route_long_name	route_type	route_color	route_text_color	route_desc
161801_2	10964	WBA4		2			
161800_2	10964	WBA3		2			
161799_2	10964	WBA2		2			
161798_2	10964	WBA1		2			

Aufgabe 1.2

Die Routen selbst sind scheinbar also nicht sehr spannend. Trips sind dann die einzelnen Fahrten entlang der Routen, die gefahren werden. Also das, was man den Fahrplan nennt. Fragen Sie doch mal die ersten 10 Fahrten der WBA2 ab.

In [4]:
```
%sql SELECT * FROM trips WHERE route_id = '161799_2' LIMIT 10;
```

```
 * sqlite:///zugauskunft.db
Done.
```

Out[4]:

route_id	service_id	trip_id	trip_headsign	trip_short_name	direction_id	block_id	shape_id	wheelchair_acce
161799_2	28763	1003122256	Zwiesel (Bay)		0		18225	
161799_2	4751	1003122242	Zwiesel (Bay)		0		18225	
161799_2	28765	1003122255	Zwiesel (Bay)		0		18225	
161799_2	28765	1003122254	Zwiesel (Bay)		0		18225	
161799_2	100	1003122253	Zwiesel (Bay)		0		18225	
161799_2	28765	1003122252	Zwiesel (Bay)		0		18225	
161799_2	28765	1003122251	Zwiesel (Bay)		0		18225	
161799_2	28765	1003122250	Zwiesel (Bay)		0		18225	
161799_2	100	1003122249	Zwiesel (Bay)		0		18225	
161799_2	28765	1003122248	Zwiesel (Bay)		0		18225	

Aufgabe 1.3

Hier fehlt einiges an Information, nicht einmal die Abfahrten finden wir. Die finden wir in den `stop_times`. Fragen Sie dort doch mal den trip 1003122256 ab.

In [5]:

```
%sql SELECT * FROM stop_times WHERE trip_id = 1003122256;
```

```
 * sqlite:///zugauskunft.db
Done.
```

Out[5]:

trip_id	arrival_time	departure_time	stop_id	stop_sequence	pickup_type	drop_off_type	stop_heads
1003122256	06:17:00	06:17:00	de:09276:5025_G	4	0	0	
1003122256	06:11:00	06:11:00	de:09276:80503	3	0	0	
1003122256	06:00:00	06:00:00	de:09276:80504	1	0	0	
1003122256	05:58:00	05:58:00	de:09276:80505	0	0	0	
1003122256	06:08:00	06:08:00	de:09276:81827	2	0	0	

Aufgabe 1.4

Unbestreitbar spannendere Informationen. Anstatt nun in der letzten Tabelle, der `stops` den Namen der Bahnhöfe aufzulösen wollen wir nun eine View `fahrplan` erstellen, die uns den Fahrplan möglichst übersichtlich präsentiert. Erstellen Sie nun einen Join, über alle Tabellen und geben Sie folgende Informationen zurück:

- Name der Route konkateniert mit der anzeige des Trips
- ID des Trips
- Ankunftszeit am Halt
- Abfahrtszeit
- Der wievielte Halt auf der Strecke ist das (sequence)
- Name des Halts

Sortieren Sie nach trip_id und stop_sequence.

In [6]:

```
%%sql

CREATE VIEW IF NOT EXISTS fahrplan AS
SELECT
    routes.route_short_name || " " || trips.trip_headsign as name,
    stop_times.trip_id,
    stop_times.arrival_time,
    stop_times.departure_time,
    stop_times.stop_sequence,
    stops.stop_name,
    stops.stop_id
from
    stop_times
JOIN stops
    USING(stop_id)
JOIN trips
    USING(trip_id)
JOIN routes
    USING(route_id)
ORDER BY
    trip_id,
    stop_sequence;

SELECT * FROM fahrplan LIMIT 3;
```

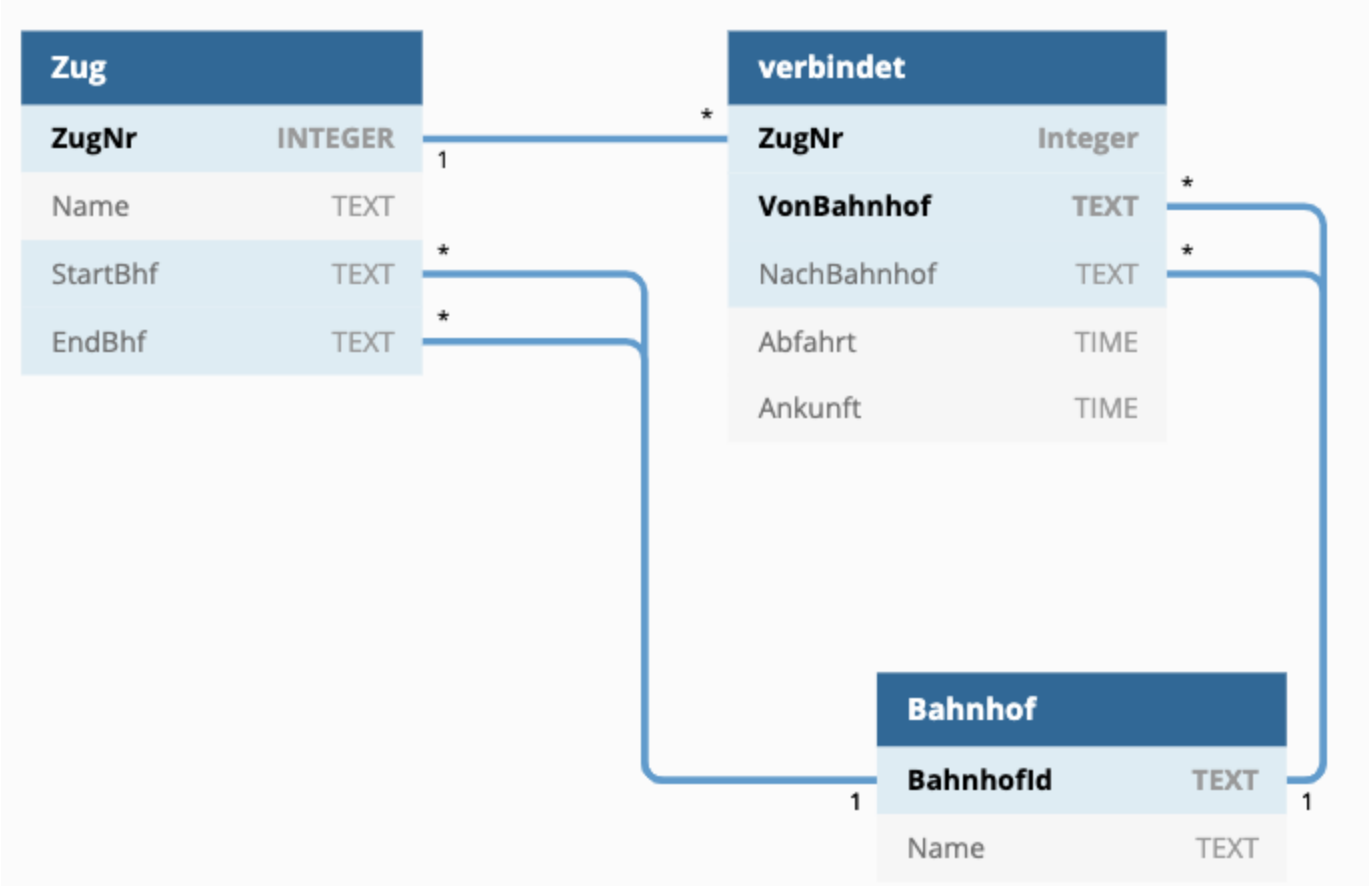
```
 * sqlite:///zugauskunft.db
Done.
Done.
```

Out[6]:

Name	trip_id	arrival_time	departure_time	stop_sequence	stop_name	stop_id
WBA1 Plattling	1003122052	05:30:00	05:30:00	0	Deggendorf Hbf	de:09271:5500
WBA1 Plattling	1003122052	05:35:00	05:35:00	1	Pankofen	de:09271:80502
WBA1 Plattling	1003122052	05:39:00	05:39:00	2	Plattling	de:09271:4445

Aufgabe 2

Betrachten Sie nun die im folgenden angegebene Zielstruktur:



Erstellen Sie eine View `Bahnhof` die wie abgebildet die beiden Attribute BahnhofId und Name hat.

In [7]:
```
%%sql
CREATE VIEW IF NOT EXISTS Bahnhof AS
SELECT stop_id, stop_name
FROM stops;
SELECT * FROM Bahnhof;
```

```
 * sqlite:///zugauskunft.db
Done.
Done.
```

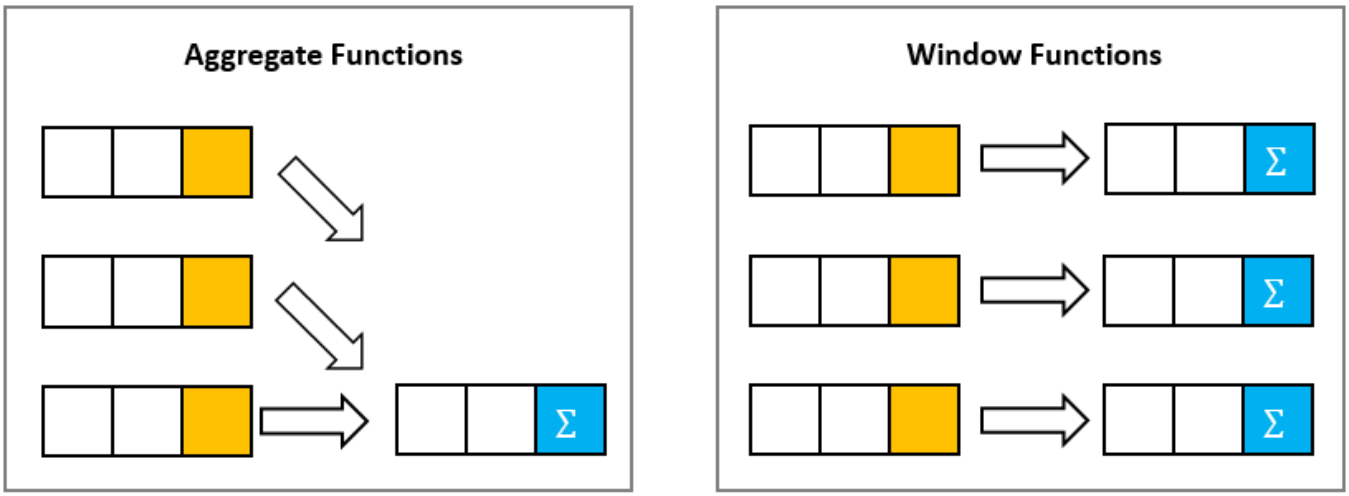
Out[7]:

stop_id	stop_name
de:09271:4445	Plattling
de:09271:5500	Deggendorf Hbf
de:09271:80502	Pankofen
de:09271:80587	Grafling-Arzting
de:09272:4688	Spiegelau
de:09272:4976_G	Klingenbrunn
de:09272:5047_G	Grafenau
de:09272:80738	Großarmschlag
de:09272:80739	Rosenau (b Grafenau)
de:09276:10969	Lichtenthal
de:09276:16165	Patersdorf, Bahnhof
de:09276:1771	Gotteszell
de:09276:1773	Viechtach, Bahnhof/ZOB
de:09276:4726	Bayerisch Eisenstein

de:09276:5000	Regen
de:09276:5025_G	Zwiesel (Bay)
de:09276:5030_G	Zwieselau
de:09276:5090	Triefenried
de:09276:7955	Gumpenried
de:09276:7962	Ruhmannsfelden
de:09276:80503	Außenried
de:09276:80504	Böhmhof
de:09276:80505	Bodenmais
de:09276:80506	Frauenau
de:09276:80540	Teisnach
de:09276:80541	Teisnach Rohde&Schwarz
de:09276:80598	Schnitzmühle
de:09276:80783	Ludwigsthal
de:09276:80784	Bettmannsäge
de:09276:81827	Langdorf

Fensterfunktionen

Fensterfunktionen sind im SQL:2003 Standard definiert. Diese Funktionen beschreiben eine Form der Aggregation die mit einem einfachen `GROUP BY` nicht erreicht werden kann: Es werden zwar die Werte mehrerer Datensätze aggregiert, also auf einen Wert gebracht, ohne aber dabei alle weiteren Elemente des Ergebnisses zu zwingen das Selbe zu tun. Grafisch stellt sich das wie folgt dar:



Quelle: https://www.sqlitetutorial.net

Links zwingt das Aggregat alle weiteren Werte dazu ebenfalls aggregiert zu werden. Rechts ist das Ergebnis des Aggregats einfach ein Teil jedes Ergebnistupels.

Die gegebenen Daten sind offensichtlich nicht in dem Format, das wir brauchen würden, um alleine die Relation Zug zu erstellen. Alle Informationen sind zwar in `fahrplan` enthalten, nur ist hier jede Reihe ein Halt eines Zuges. Um Start- und Endbahnhof zu bestimmen müsste man nach `trip_id` gruppieren und in die erste und

letzte Zeile eines Zuges schauen könnten. Bis zur Standardtisierung der sog. Fensterfunktionen (window) waren dafür eine große Menge von Subselects nötig. Nun geht das deutlich einfacher.

Wir beschränken uns hier auf die Elemente, die wir zum Erreichen unseres Ziels brauchen. Der Aufbau einer Fensterfunktion ist wie folgt:

```
<AGGREGATFUNKTION> OVER([<PARTITION BY>] [<ORDER BY>] [<FRAMESPEC>])
```

- Eine Aggregatfunktion ist jede normale Aggregatfunkion, die wir im `GROUP BY` verwenden. Es gibt aber für Fenster eben auch neue Funktionen, z.b. `first_value(attribut), last_value(attribut)` oder zum Durchnumerieren `row_number()` . Eine Liste für SQLite finden Sie hier.
- `OVER()` leitet ein Fenster ein. Spezifizieren Sie zwischen den Klammern keine Partition, keine Sortierung und keinen Frame ist es ein Fenster über alle Daten
- `PARTITION BY` gruppiert nach dem gegebenen Attribut
- `ORDER BY` sortiert die Liste, ohne dass das Auswirkungen auf die Sortierung der Query hat
- Die Framespec erlaubt Ihnen anzugeben wie groß Ihr Fenster sein soll. Das sprengt hier den Rahmen

Aufgabe 3

Bringen wir nun also die Zug Relation in Form. Dafür brauchen wir eine Tabelle mit `zugnr, name, startbhf, endbhf` . Erstellen Sie also eine View anhand der Hinweise im Folgenden:

Hinweis: kopieren Sie diese Zelle und entwickeln Sie hier Ihre Query

```
%sql
SELECT
    DISTINCT trip_id as zugnr,
    as name -- TODO Hier erstellen wir eine Partition anhander der TRIP ID.
Von dieser Partition nehmen wir den ersten Wert des namen
    as startbhf -- TODO Hier erstellen wir eine Partition anhander der TRIP
ID. Von dieser Partition nehmen wir den ersten Wert der bahnhofs_id
    as endbhf -- TODO Hier erstellen wir eine Partition anhander der TRIP
ID. Von dieser Partition nehmen wir den letzten Wert der bahnhofs_id
FROM fahrplan
ORDER BY RANDOM()
LIMIT 5;
```

In [34]:

```
%%sql
DROP VIEW IF EXISTS zug;
CREATE VIEW IF NOT EXISTS zug AS
SELECT
    DISTINCT trip_id as zugnr,
    first_value(name) OVER(PARTITION BY trip_id) as name,
    first_value(stop_id) OVER(PARTITION BY trip_id) as startbhf,
    last_value(stop_id) OVER(PARTITION BY trip_id) as endbhf
FROM fahrplan;

SELECT * FROM zug ORDER BY RANDOM() LIMIT 5;
```

```
 * sqlite:///zugauskunft.db
Done.
Done.
Done.
```

Out[34]:

zugnr	name	startbhf	endbhf
1003122251	WBA2 Zwiesel (Bay)	de:09276:80505	de:09276:5025_G

1003122240	WBA2 Bodenmais	de:09276:5025_G	de:09276:80505
1003122194	WBA1 Bayerisch Eisenstein	de:09271:4445	de:09276:4726
1003122283	WBA4 Viechtach, Bahnhof/ZOB	de:09276:1771	de:09276:1773
1003122120	WBA1 Plattling	de:09271:5500	de:09271:4445

Aufgabe 4

Abschließend folgt noch die finale View. Jeder Abschnitt soll einzeln aufgelistet werden. Dazu brauchen wir die Zugnummer, die ID des *von* Bahnhofs, die ID des *zu* Bahnhofs, Abfahrts und Ankunftszeit.

Das Vorgehen ist ähnlich wie oben, nur das wir für den nächsten Halt ja in die nächste Ergebniszeile schauen wollen. Das macht die `lead()` funktion für uns. Versuchen wir es erst einmal in einem SELECT:

Hinweis: kopieren Sie diese Zelle und entwickeln Sie hier Ihre Query

```
%%sql

SELECT
    *,
    AS next_stop_name, -- stop_name Analog aufgabe 3 nur dass wir lead
verwenden
    AS next_stop_id -- stop_id Analog aufgabe 3 nur dass wir lead verwenden
FROM
    fahrplan;
```

In [45]:

```
%%sql

DROP VIEW IF EXISTS verbindet;

CREATE VIEW verbindet AS
SELECT
    trip_id as zugnr,
    stop_name,
    stop_id,
    LEAD(stop_name) OVER(PARTITION BY trip_id ORDER BY trip_id, stop_sequence) AS next_sto
    LEAD(stop_id) OVER(PARTITION BY trip_id ORDER BY trip_id, stop_sequence) AS next_stop_
FROM
    fahrplan;
```

```
 * sqlite:///zugauskunft.db
Done.
Done.
[]
```

Out[45]: