Übungszettel: Taxi-Graph (Neo4j + Arrows)

Aufgabe 1 – Text → Diagramm (4 Nodes, 4 Relations)

Angabe
Erstelle in **Arrows** (oder auf Papier) ein Diagramm aus folgenden **4 Nodes** und **4 Relationships**. Achte darauf, dass **Properties** dort stehen, wo sie angegeben sind.

Nodes (4)
```
(:Customer {customerId:"C001", name:"Alice", email:"alice@example.com"})
(:Drive {driveId:"D9001", requestedAt: datetime("2026-01-20T10:15:00"),
distanceKm: 4.2, fare: 12.30, currency:"EUR"})
(:Location {locationId:"L100", name:"Vienna Hbf", city:"Vienna"})
(:Location {locationId:"L200", name:"Stephansplatz", city:"Vienna"})
```

Relationships (4)
```
(Customer)-[:REQUESTED {at: datetime("2026-01-20T10:15:00"), passengers:
1}]->(Drive)
(Customer)-[:TOOK {paymentMethod:"CARD", tip: 2.00}]->(Drive)
(Drive)-[:FROM {pickupTime: time("10:20:00")}]->(Location L100)
(Drive)-[:TO {dropoffTime: time("10:35:00")}]->(Location L200)
```

Hinweis: REQUESTED und TOOK sind **zwei verschiedene Beziehungstypen** (das ist erlaubt), aber sie sollten **unterschiedliche Bedeutung/Properties** haben (keine redundanten gleichen Fakten doppelt speichern).

Aufgabe 2 – Diagramm → Cypher (Konvertierung)

Angabe
Gegeben ist folgendes Diagramm (als Textbeschreibung). Schreibe **Cypher**, das das Diagramm in Neo4j erstellt.

Nodes
```
Node A: (:Customer {customerId:"C002", name:"Bob",
email:"bob@example.com"})
Node B: (:Drive {driveId:"D9002", requestedAt: datetime("2026-01-
21T09:05:00"), distanceKm: 6.8, fare: 19.10, currency:"EUR"})
Node C: (:Location {locationId:"L200", name:"Stephansplatz",
city:"Vienna"})
Node D: (:Location {locationId:"L300", name:"Schönbrunn", city:"Vienna"})
```

Relationships
```
(Customer C002)-[:TOOK]->(Drive D9002)
(Drive D9002)-[:FROM]->(Location L200)
(Drive D9002)-[:TO]->(Location L300)
(Customer C002)-[:RATED {stars: 4, comment:"smooth ride"}]->(Drive D9002)
```

Aufgabe:
- Lege (falls sinnvoll) **Constraints** an.
- Erstelle alle Nodes per MERGE.
- Erstelle alle Relationships per MERGE.

Lösung (Cypher)

```
// Constraints (optional aber empfohlen)
CREATE CONSTRAINT customer_id_unique IF NOT EXISTS
FOR (c:Customer) REQUIRE c.customerId IS UNIQUE;

CREATE CONSTRAINT location_id_unique IF NOT EXISTS
FOR (l:Location) REQUIRE l.locationId IS UNIQUE;

CREATE CONSTRAINT drive_id_unique IF NOT EXISTS
FOR (d:Drive) REQUIRE d.driveId IS UNIQUE;

// Nodes
MERGE (c:Customer {customerId:"C002"})
SET c.name="Bob", c.email="bob@example.com";

MERGE (d:Drive {driveId:"D9002"})
SET d.requestedAt = datetime("2026-01-21T09:05:00"),
    d.distanceKm  = 6.8,
    d.fare        = 19.10,
    d.currency    = "EUR";

MERGE (from:Location {locationId:"L200"})
SET from.name="Stephansplatz", from.city="Vienna";

MERGE (to:Location {locationId:"L300"})
SET to.name="Schönbrunn", to.city="Vienna";

// Relationships
MATCH (c:Customer {customerId:"C002"}),
      (d:Drive {driveId:"D9002"}),
      (from:Location {locationId:"L200"}),
      (to:Location {locationId:"L300"})
MERGE (c)-[:TOOK]->(d)
MERGE (d)-[:FROM]->(from)
MERGE (d)-[:TO]->(to)
MERGE (c)-[:RATED {stars: 4, comment:"smooth ride"}]->(d);
```