Here’s a simple taxi model with exactly Customer, Location, and Drive (a drive goes FROM one location TO another).

Model (labels + relationships)

Nodes

```
(:Customer {customerId, name, email})
(:Location {locationId, name, city})
(:Drive {driveId, requestedAt, distanceKm, fare, currency})
```

Relationships

```
(c:Customer) -[:TOOK]-> (d:Drive)
(d:Drive) -[:FROM]-> (l1:Location)
(d:Drive) -[:TO]->   (l2:Location)
```

Create the model

```
// Uniqueness constraints
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;


// Sample nodes
MERGE (c1:Customer {customerId:"C001"}) SET c1.name="Alice",
c1.email="alice@example.com";
MERGE (c2:Customer {customerId:"C002"}) SET c2.name="Bob",
c2.email="bob@example.com";

MERGE (l1:Location {locationId:"L100"}) SET l1.name="Vienna Hbf",
l1.city="Vienna";
MERGE (l2:Location {locationId:"L200"}) SET l2.name="Stephansplatz",
l2.city="Vienna";
MERGE (l3:Location {locationId:"L300"}) SET l3.name="Schönbrunn",
l3.city="Vienna";

MERGE (d1:Drive {driveId:"D9001"})
SET d1.requestedAt=datetime("2026-01-20T10:15:00"), d1.distanceKm=4.2,
d1.fare=12.30, d1.currency="EUR";

MERGE (d2:Drive {driveId:"D9002"})
SET d2.requestedAt=datetime("2026-01-21T09:05:00"), d2.distanceKm=6.8,
d2.fare=19.10, d2.currency="EUR";


// Relationships (customer + from/to locations)
MATCH (c1:Customer {customerId:"C001"}), (d1:Drive {driveId:"D9001"}),
      (l1:Location {locationId:"L100"}), (l2:Location {locationId:"L200"})
MERGE (c1)-[:TOOK]->(d1)
```

```
MERGE (d1)-[:FROM]->(l1)
MERGE (d1)-[:TO]->(l2);

MATCH (c2:Customer {customerId:"C002"}), (d2:Drive {driveId:"D9002"}),
      (l2:Location {locationId:"L200"}), (l3:Location {locationId:"L300"})
MERGE (c2)-[:TOOK]->(d2)
MERGE (d2)-[:FROM]->(l2)
MERGE (d2)-[:TO]->(l3);
```

Cypher Querys

Show all nodes and relations

```
MATCH (n)-[r]->(m)
RETURN n, r, m;
```

Show all nodes and relations (include isolated)

```
MATCH (n)
OPTIONAL MATCH (n)-[r]-(m)
RETURN n, r, m
LIMIT 200;
```

Show all customers took a ride from Vienna Hbf

```
MATCH (c:Customer)-[:TOOK]->(d:Drive)-[:FROM]->(l:Location)
WHERE l.name = "Vienna Hbf"
RETURN DISTINCT c
ORDER BY c.customerId;
```

Show all top customers including the number of rides

```
MATCH (c:Customer)-[:TOOK]->(:Drive)
RETURN c.customerId AS customerId, c.name AS name, count(*) AS rides
ORDER BY rides DESC
LIMIT 10;
```

```
MATCH (c:Customer)
OPTIONAL MATCH (c)-[:TOOK]->(:Drive)
RETURN c.customerId AS customerId, c.name AS name, count(*) AS rides
ORDER BY rides DESC;
```

Customers who did a ride from Vienna Hbf also did a ride from ...

```
MATCH (c:Customer)-[:TOOK]->(:Drive)-[:FROM]->(:Location {name:"Vienna Hbf"})
MATCH (c)-[:TOOK]->(:Drive)-[:FROM]->(other:Location)
WHERE other.name <> "Vienna Hbf"
RETURN other.name AS alsoFrom, count(DISTINCT c) AS customers
ORDER BY customers DESC
LIMIT 10;
```

Which customers did a ride to ALL destinations

```
MATCH (c:Customer)
WHERE NOT EXISTS {
  MATCH (l:Location)<-[:TO]-(:Drive)          // all destination locations
  WHERE NOT EXISTS {
    MATCH (c)-[:TOOK]->(:Drive)-[:TO]->(l)    // customer reached that
destination
```

```
  }
}
RETURN c.customerId AS customerId, c.name AS name
ORDER BY customerId;
```