1a.

- Customer number is unique identifier for each customer
- Part number is unique identifier for each part
- Cage code is unique identifier for each shelf
- We see many-to-one relationship between parts and cage codes, since you can store many items on the same shelf
- Employee doesn't define a relationship between employee and customer
- Date and time define a specific moment of the order time

b.

The original form table is in unnormalised form, as we have repeating groups. The easiest methodology is to define customers into a separate table.

customerNumber (PK)	customerName	customerType
HG54587	Jeff Peterson	Customer

Where customerNumber is the primary key and no foreign keys.

Our next step is to uniquely identify parts into a separate table.

c.

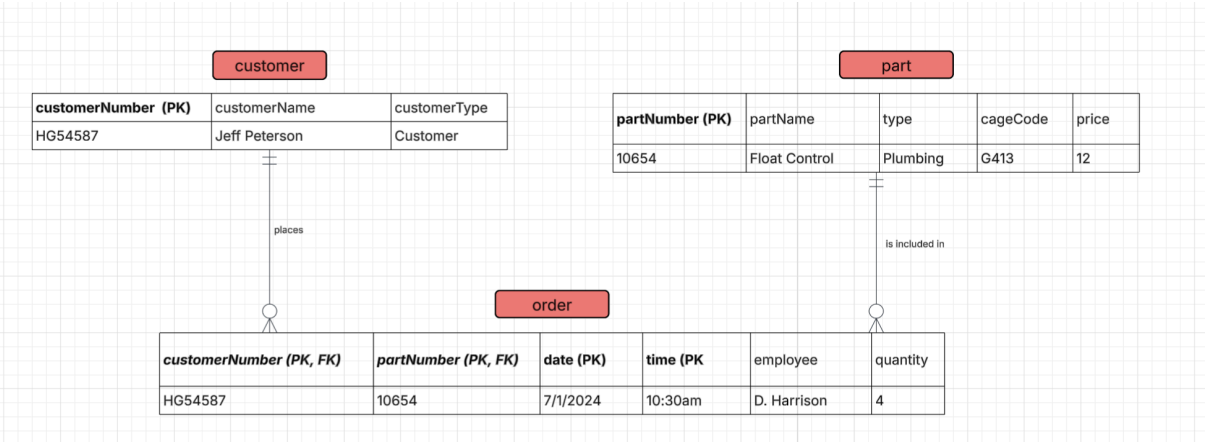
partNumber (PK)	partName	type	cageCode	price
10654	Float Control	Plumbing	G413	12

We do this to separate them into tables and avoid having repetitive groups. Also, it's easy to visualize this since we don't have many tables. Hence, our last table is order details

customerNumber (PK, FK)	***partNumber (PK, FK)***	**date (PK)**	**time (PK)**	employee	quantity
HG54587	10654	7/1/2024	10:30am	D. Harrison	4

d.

Thus, in the customer table we have customerNumber as PK, partNumber is PK in part table, and then we have a composite key: customerNumber, partNumber, date, time that uniquely identifies order details. customerNumber is related to the customer table, partNumber to part. Below is the EDR using LucidApp. This table resolves many to many relationships between customer and part with event.



2a.

- Staff number uniquely identifies each therapist
- Patient number uniquely identifies each patient
- Due to Richard Levin (his entry on the table) therapists can work in different branches but only on one branch at any specific day.
- Branch number uniquely identifies the branch
- Patients can have multiple appointments
- When I make normalized tables, I explicitly assume that an appointment is uniquely identified by patient number, staff number and appointment date (which serves as a key to know the branch from the table with staff assignments.

The given data is already in 1NF form since all attributes are atomic and no repeating groups. What we see first is to make separate tables for staff and patients.

staffId (PK)	therapistName
S1011	Fred Smith

patientId (PK)	patientName
P100	Lily White

Then, we see that we need to make a table to determine the daily assignment of different therapists to different branches. So then we can use the appointment date as a composite key in the table for appointment.

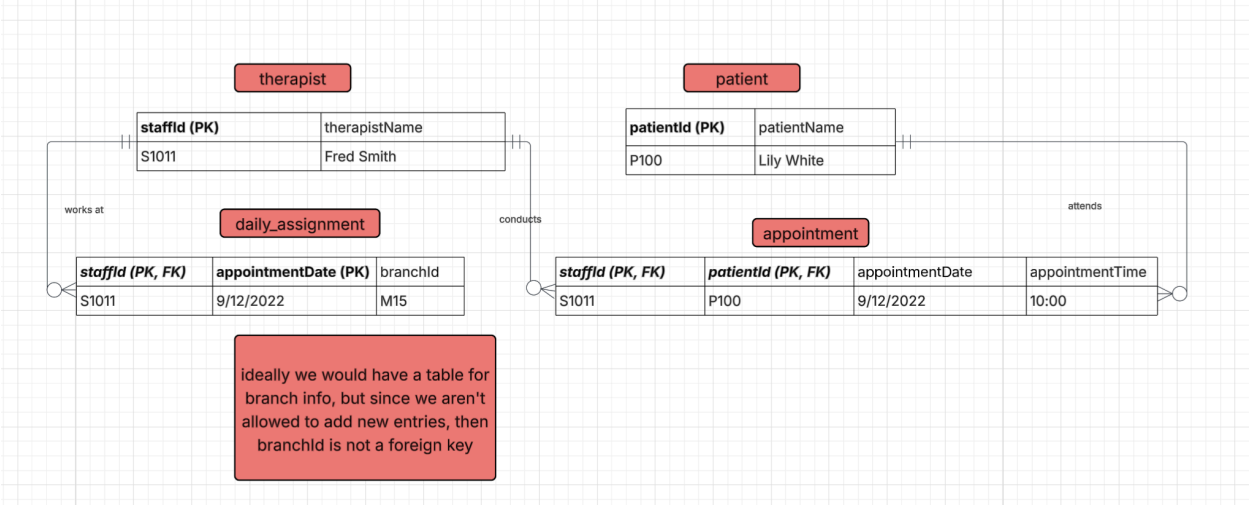
staffId (PK, FK)	**appointmentDate (PK)**	branchId
S1011	9/12/2022	M15

So we identify the daily assignment for each therapist.

And lastly we can do appointment details with a separate table to identify unique relationships.

staffId (PK, PK)	***patientId (PK, FK)***	**appointmentDate (PK)**	appointmentTime
S1011	P100	9/12/2022	10:00

To achieve 3NF, we removed all transitive dependencies.



Above I identified all relationships between tables, and thus we have 3NF.

3a.

- eNo is unique identifier for each employee
- contractId is unique identifier for each contract
- evenNo is unique identifier for each event
- One contract is for one event, but one event can have multiple contracts
- Each employee can work on multiple contracts.

Since we don’t have many entries, we can see that the table is already in 1NF since no repeating groups and attributes are atomic.

First, to achieve 2NF we must remove partial dependencies with employee name and id, and second to identify contracts with events.

eNo (PK)	eName
1135	Smith J.

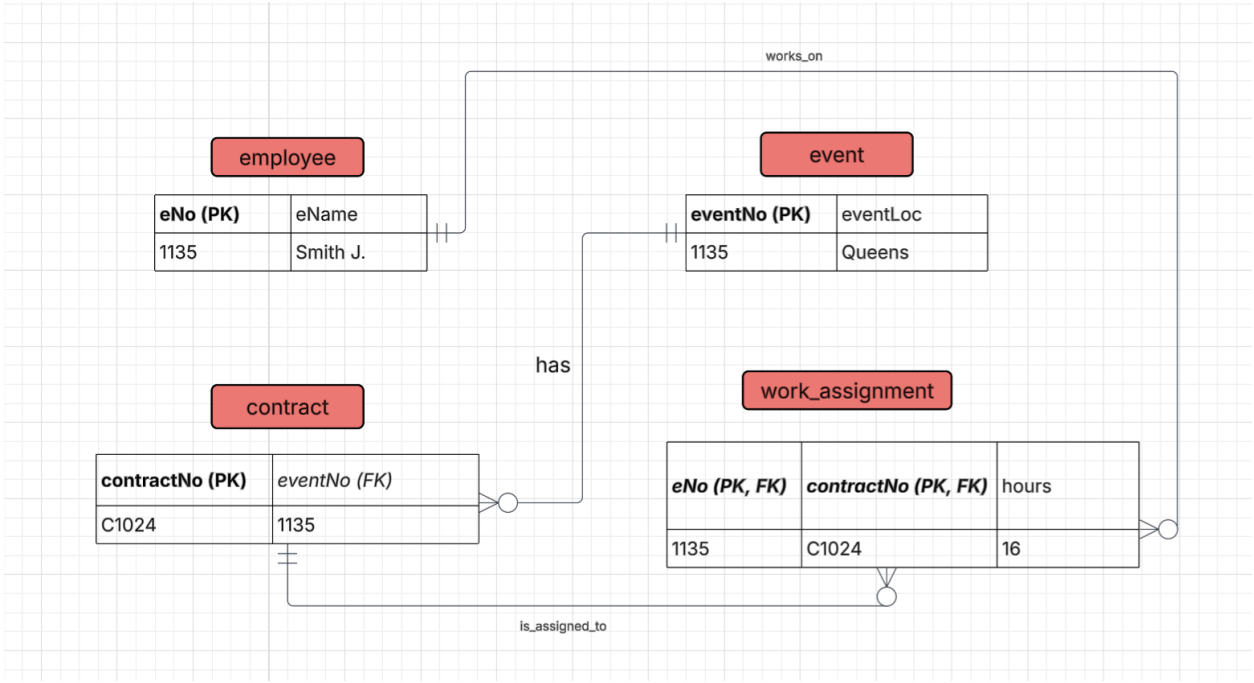
eventNo (PK)	eventLoc
1135	Queens

contractNo (PK)	*eventNo (FK)*
C1024	1135

Once we connect these tables to bring them into 3NF form, we also need to make the table with work assignment.

eNo (PK, FK)	***contractNo (PK, FK)***	hours
1135	C1024	16

Here is the EDR from lucid.



We have employee to contract (one to many), event to contract (one to many), contract to work assignment (one to many), and the work assignment table is used to resolve many-to-many relationship between employee and contracts and associate them.

For all tables, the process is the same. First, we bring them to 1NF to make sure that each row is unique, value cells have only one value, and attributes are atomic. After that we need to resolve partial dependencies for each table and create separate tables for them (as to avoid repetition and potential

anomalies). For example, we create separate tables with staff, customer, patient, etc, to store individual info. To get them to 3NF, we resolve transitive dependencies, meaning we make sure that no non-key attribute depends on another non-key attribute, and thus we normalize the data in 3NF form.