NORMALISATION WORKSHEET

From 0NF → 1NF → 2NF → 3NF

Instructions only — no answers included

This worksheet will guide you through ALL stages of normalisation using the FactoryFlow dataset.
Follow the steps carefully and use the hints to help you.

SECTION 1 — Understanding 0NF (Zero Normal Form)

You will start with a messy table in **0NF**.
Your job is to clean it and break it into proper relational tables.

Task 1 — Spot the Problems

Look at the 0NF table.
Make notes about anything that looks messy or confusing.

Hints:

- Does any cell contain more than one thing?
- Are different types of information mixed together?
- Are any lists hidden inside a single cell?

SECTION 2 — Moving from 0NF to 1NF

Your goal in 1NF is to make the data tidy and consistent.

Task 2 — Break the data into smaller tables

Look for groups of information that repeat.
These usually become their own tables.

Hints:

- Photos often repeat
- Messages repeat
- Status updates repeat

- A fault may contain several separate details

Task 3 — Make each value atomic

Rewrite the data so each cell holds **one value only**.

Hints:

- One photo per row
- One message per row
- One status update per row

Task 4 — Add Primary Keys

Every table needs a Primary Key (PK).

Hints:

- Create a new ID if necessary (e.g. PhotoID, StatusID)
- A PK must be unique for each record

Task 5 — Add Foreign Keys

Use a Foreign Key to link tables.

Hints:

- Anything connected to a fault should contain FaultID
- A FK should appear in the "child" table

SECTION 3 — Moving from 1NF to 2NF

Your goal in 2NF is to fix problems caused by composite keys.

Task 6 — Check for composite keys

A composite key is when a table uses **two fields together** to uniquely identify a row.

Hints:

- If you find a composite key, ask:
  "Does every field depend on BOTH parts of this key?"
- If the answer is "no", the table is not in 2NF

Task 7 — Fix partial dependencies

If a table breaks the rule, turn part of the composite key into its own field.

Hints:

- Usually you fix this by creating a new ID
- Keep the original link as a Foreign Key

SECTION 4 — Moving from 2NF to 3NF

Your goal in 3NF is to remove transitive dependencies.

Task 8 — Look for chained dependencies

A transitive dependency is when a field depends on something **other than** the Primary Key.

Hints:

- Look for any field that describes *another* field
- Ask: “Does this piece of information belong in a different table?”

Task 9 — Create new tables for these dependencies

If you find data that doesn’t depend directly on the table’s PK, move it to its own table.

Hints:

- Customer information often belongs in a separate table
- Replace it with a new Foreign Key

SECTION 5 — Final Checks

When you think you're finished:

Task 10 — Check each table

Ask yourself:

- Does each table have a primary key?
- Does every field depend directly on that key?
- Is every cell atomic (one thing only)?

- Are all relationships shown with foreign keys?
- Does each table represent ONE type of thing only?

SECTION 6 — Data Dictionary

Once you have finished normalising your data into 1NF, 2NF, and 3NF, your next challenge is to produce a **full data dictionary** for the final set of tables.

A data dictionary explains *exactly* what each field in the database means.

This task helps you understand your structure more clearly — and helps you check if your design makes sense.

1. Field Name

The exact name you will use in SQL.

2. Description

A clear explanation (in your own words) of what this field stores.

3. Data Type (SQL-ready)

Choose suitable SQL data types such as:

- VARCHAR(n)
- INT
- DATE
- DATETIME
- BOOLEAN / BIT
- TEXT

Tip:
Choose a length that makes sense (e.g., VARCHAR(50), not VARCHAR(999)).

4. Key Type

Say whether the field is:

- **PK** (Primary Key)
- **FK** (Foreign Key) — and state which table it links to
- **None**

5. Validation Rules

These rules help SQL enforce the correct data. Include things like:

- **NOT NULL** (field must not be empty)
- **UNIQUE** (all values must be different)
- **CHECK()** constraints
- Allowed formats (e.g., must be text only, must be a date, cannot be negative)

Tip:
A primary key should ALWAYS be **NOT NULL** and **UNIQUE**.

6. Example Value

Give a sensible example (not a full record set).

What Your Data Dictionary Should Look Like

(Students fill it in — do not provide answers.)

Table Name: ______________________

Field Name	Description	Data Type	Key	Validation Rules	Example
…	…	…	…	…	…
…	…	…	…	…	…

Repeat for EVERY table in your 3NF design.

Hints to Help You Make Good Choices

• A field that identifies a record should normally be **INT** or **VARCHAR** and set as **Primary Key**.
• Names and descriptions are usually **VARCHAR**.
• Dates use **DATE** or **DATETIME**.
• Long text (e.g., messages) can use **TEXT**.
• Foreign keys must use the **same data type** as the primary key they link to.
• Validation makes the database more reliable, for example:

- NOT NULL for important fields

- CHECK(LENGTH(field) < 60) for limits
- CHECK(field >= 0) for numerical ranges

Success Criteria

You have completed this extension when:

✓ Each table has a full data dictionary
✓ All data types are suitable for SQL
✓ Validation rules are included and make sense
✓ Every PK and FK is correctly labelled
✓ The data dictionary could be used to write complete CREATE TABLE statements

Now write the SQL....

Write the **SQL CREATE TABLE** code for ONE of your tables
using the information from your data dictionary.

EXTENSION TASK (Optional)

Draw an ERD (Entity Relationship Diagram) showing:

- All tables
- All primary keys
- All foreign keys
- The relationships between them