3NF Justification Report

1. Overview

The original Kaggle dataset (Healthcare.csv) is a single wide table that mixes patient information, neighborhood information, and appointment events:

Patient: PatientId, Gender, Age, Scholarship, Hypertension, Diabetes, Alcoholism, Handicap

Neighborhood: Neighbourhood

Appointment: AppointmentID, ScheduledDay, AppointmentDay, SMS_received, No-show

Every appointment row repeats the same patient and neighborhood data. This causes all three classic anomalies:

Update anomaly – changing a patient's chronic condition (e.g., Hypertension) requires updating many rows.

Insert anomaly – we cannot store a new patient or neighborhood until that patient has at least one appointment.

Delete anomaly – if we delete the last appointment of a patient, we lose all information about that patient.

To remove these problems and reach at least Third Normal Form (3NF), we decomposed the raw table into three relations:

Neighborhood(neighborhood_id, name)

Patient(patient_id, external_patient_id, gender, age_at_registration, neighborhood_id, has_scholarship, has_hypertension, has_diabetes, has_alcoholism, handicap_level)

Appointment(appointment_id, patient_id, scheduled_datetime, appointment_datetime, sms_received, no_show)

2. Functional Dependencies

Based on the semantics of the data, we assume the following key functional dependencies (FDs):

Neighborhood

neighborhood_id → name

(Because we enforce UNIQUE(name), name → neighborhood_id also holds.)

Patient

patient_id → external_patient_id, gender, age_at_registration, neighborhood_id, has_scholarship, has_hypertension, has_diabetes, has_alcoholism, handicap_level

external_patient_id → patient_id, gender, age_at_registration, neighborhood_id, has_scholarship, has_hypertension, has_diabetes, has_alcoholism, handicap_level

(each external patient id is unique in the source data)

Appointment

appointment_id → patient_id, scheduled_datetime, appointment_datetime, sms_received, no_show

There are no meaningful FDs where a non-key attribute determines another non-key attribute within the same table. For example, neighborhood_id → name is enforced in the Neighborhood table, not inside Patient.

3. Normal Form Analysis

3.1 Neighborhood(neighborhood_id, name)

Candidate keys: {neighborhood_id}, {name}

All non-key attributes (name or neighborhood_id depending on which key you pick) are fully functionally dependent on the key.
There are no partial dependencies (keys are single attributes).
There are no transitive dependencies among non-key attributes.
The neighborhood is in BCNF and therefore also in 3NF.
3.2 Patient(patient_id, external_patient_id, gender, age_at_registration, neighborhood_id, has_scholarship, has_hypertension, has_diabetes, has_alcoholism, handicap_level)
Candidate keys: {patient_id}, {external_patient_id} (both uniquely identify a patient).
Every non-key attribute is fully dependent on the key:
e.g., patient_id → gender, age_at_registration, neighborhood_id, has_scholarship, …
There are no partial dependencies (again, keys are single attributes).
There are no transitive dependencies inside this table:
neighborhood_id → name exists, but name is stored in the Neighborhood table, not in Patient, so transitivity is handled via a foreign key relationship, not as a violation.
The patient is in BCNF and thus in 3NF.
3.3 Appointment(appointment_id, patient_id, scheduled_datetime, appointment_datetime, sms_received, no_show)
Candidate key: {appointment_id}
All other attributes (patient_id, scheduled_datetime, appointment_datetime, sms_received, no_show) depend directly on appointment_id.
No partial dependencies (key is a single attribute).
No transitive dependencies between non-key attributes.
Appointment is also in BCNF and 3NF.

4. How the Decomposition Removes Anomalies

Update anomaly fixed
Patient attributes (age, chronic conditions, scholarship) are stored once in Patient. Updating a patient's hypertension status now touches only one row. Neighborhood names are stored once in Neighborhood, so renaming a neighborhood affects just a single tuple.
Insert anomaly fixed
We can insert a new neighborhood into Neighborhood or a new patient into Patient even if they don't yet have an appointment. There is no longer a requirement to create a dummy appointment row first.
Delete anomaly fixed
Deleting an appointment from Appointment does not delete the corresponding patient or neighborhood. The entity data lives independently in Patient and Neighborhood.
Referential integrity
Foreign keys:
Patient.neighborhood_id → Neighborhood.neighborhood_id
Appointment.patient_id → Patient.patient_id
guarantee that every appointment refers to a valid patient, and every patient refers to a valid neighborhood.

5. Conclusion

Starting from a single denormalized CSV, we identified the core entities (Neighborhood, Patient, Appointment) and their functional dependencies. We decomposed the raw data into three relations such that:

Each relation has a clearly defined primary key.

All non-key attributes are fully dependent on the key, with no partial or transitive dependencies.

Classic update/insert/delete anomalies are eliminated.

Foreign keys preserve the original semantics and allow efficient joins for analytics (e.g., computing no-show rates by neighborhood).

Therefore, the final relational schema for the healthcare no-show project is in at least Third Normal Form (3NF), and in fact each relation also satisfies Boyce-Codd Normal Form (BCNF).