3NF Justification Report – Healthcare Admissions Schema

1. Overview

The original healthcare_dataset.csv can be viewed as a single wide table that mixes together:

- **Patient information** – name, age, gender, blood type
- **Doctor information** – doctor name
- **Hospital information** – hospital name
- **Insurance information** – provider name
- **Medical condition** – condition name
- **Visit / admission details** – admission date, discharge date, room number, admission type, billing amount
- **Lab test results** – test result status
- **Medication information** – medication name

Conceptually, each row represents a hospital admission/visit, but it repeats the same patient, doctor, hospital, insurance, condition, and medication details for every admission of that patient. This leads to the classic anomalies:

- **Update anomaly** – If a patient's blood type or gender is corrected, or a hospital/insurance provider/doctor is renamed, the change must be made in many different rows.
- **Insert anomaly** – We cannot store a new patient, doctor, hospital, insurance provider, or medical condition until there is at least one admission row for them.
- **Delete anomaly** – If we delete the last admission of a patient, we lose all information about that patient, their doctor, or the associated insurance if it only appeared in that row.

To remove these problems and achieve Third Normal Form (3NF), we decompose the wide table into the following relations (entities):

- Patient(patient_id, name, age, gender, blood_type)
- Doctor(doctor_id, doctor_name)
- Hospital(hospital_id, hospital_name)
- InsuranceProvider(insurance_id, provider_name)
- MedicalCondition(condition_id, condition_name)
- Medication(medication_id, medication_name)

- Admission(admission_id, patient_id, doctor_id, hospital_id, insurance_id, condition_id, date_of_admission, discharge_date, room_number, admission_type, billing_amount)
- TestResult(test_result_id, admission_id, result)

Here, every row in the original CSV corresponds to exactly one row in Admission, but the descriptive data about patients, doctors, hospitals, etc., is now stored once in separate tables and referenced via foreign keys.

---

2. Functional Dependencies

2.1 Patient

Primary Key: patient_id **FD:** patient_id → name, age, gender, blood_type Each patient_id uniquely identifies exactly one patient and all their demographic attributes.

2.2 Doctor

Primary Key: doctor_id **FD:** doctor_id → doctor_name

2.3 Hospital

Primary Key: hospital_id **FD:** hospital_id → hospital_name

2.4 Insurance Provider

Primary Key: insurance_id **FD:** insurance_id → provider_name

2.5 Medical Condition

Primary Key: condition_id **FD:** condition_id → condition_name

2.6 Medication

Primary Key: medication_id **FD:** medication_id → medication_name

2.7 Admission / Visit

Primary Key: admission_id **FD:** admission_id → patient_id, doctor_id, hospital_id, insurance_id, condition_id, date_of_admission, discharge_date, room_number, admission_type, billing_amount admission_id uniquely identifies a specific visit of a particular patient to a hospital under a doctor, with a specific condition, insurance coverage, dates, and billing information.

2.8 Test Result

Primary Key: test_result_id **FD:** test_result_id → admission_id, result Every test result row corresponds to one admission and records a single categorical outcome (Normal, Abnormal, or Inconclusive).

3. Normal Form Analysis

We now analyze each relation for keys, full functional dependency, and absence of transitive dependencies. Like in the previous report, we check that every relation is in BCNF, and therefore also in 3NF.

3.1 Patient(patient_id, name, age, gender, blood_type)

- **Candidate key(s):** {patient_id}
- All attributes (name, age, gender, blood_type) depend fully on patient_id.
- The key is a single attribute, so there are no partial dependencies.
- There are no non-key attributes that determine other non-key attributes inside this table.
- **Conclusion:** Patient is in BCNF and thus also in 3NF.

3.2 Doctor(doctor_id, doctor_name)

- **Candidate key(s):** {doctor_id}
- doctor_name is fully functionally dependent on doctor_id.
- No partial or transitive dependencies.
- **Conclusion:** Doctor is in BCNF/3NF.

3.3 Hospital(hospital_id, hospital_name)

- **Candidate key(s):** {hospital_id}
- hospital_name depends only on hospital_id.
- No partial or transitive dependencies.
- **Conclusion:** Hospital is in BCNF/3NF.

3.4 InsuranceProvider(insurance_id, provider_name)

- **Candidate key(s):** {insurance_id}
- provider_name is fully dependent on the key.
- No partial or transitive dependencies.
- **Conclusion:** InsuranceProvider is in BCNF/3NF.

3.5 MedicalCondition(condition_id, condition_name)

- **Candidate key(s):** {condition_id}
- condition_name depends fully on condition_id.

- No other attributes, so no anomalies.
- **Conclusion:** MedicalCondition is in BCNF/3NF.

3.6 Medication(medication_id, medication_name)

- **Candidate key(s):** {medication_id}
- medication_name is fully dependent on medication_id.
- No partial or transitive dependencies.
- **Conclusion:** Medication is in BCNF/3NF.

3.7 Admission(admission_id, patient_id, doctor_id, ...)

- **Candidate key(s):** {admission_id}
- All other attributes are fully functionally dependent on admission_id.
- The key is a single attribute, so there are no partial dependencies.
- Dependencies such as patient_id → name exist only in their respective tables and are enforced through foreign keys, not as dependencies among non-key attributes inside Admission. Therefore, no transitive dependencies occur within this table.
- **Conclusion:** Admission is in BCNF/3NF.

3.8 TestResult(test_result_id, admission_id, result)

- **Candidate key(s):** {test_result_id}
- admission_id and result both depend on test_result_id.
- No other attributes; the key is a single attribute; there are no partial or transitive dependencies.
- **Conclusion:** TestResult is in BCNF/3NF.

---

4. How the Decomposition Removes Anomalies

By splitting the original wide admission CSV into entity tables plus the fact table Admission, we eliminate the anomalies that existed in the denormalized design.

4.1 Update Anomaly

Patient data (name, age, gender, blood type) is stored only once per patient in **Patient**. Doctor names live only in **Doctor**. Updating any of these attributes now requires changing exactly one row instead of many repeated rows in the admissions fact table.

4.2 Insert Anomaly

We can insert a new patient into **Patient** even if they have not yet been admitted. We can also insert a new hospital, doctor, or insurance provider into their own tables without creating a dummy admission. Thus, valid reference data can be stored independently of transactional admission events.

4.3 Delete Anomaly

Deleting an admission row from **Admission** does not remove the corresponding patient, doctor, hospital, insurance provider, or medical condition. The entity data survives as long as its row exists in its own table.

4.4 Referential Integrity

We also maintain foreign key constraints to keep the database consistent:

- Admission.patient_id → Patient.patient_id
- Admission.doctor_id → Doctor.doctor_id
- Admission.hospital_id → Hospital.hospital_id
- Admission.insurance_id → InsuranceProvider.insurance_id
- Admission.condition_id → MedicalCondition.condition_id
- TestResult.admission_id → Admission.admission_id

These constraints ensure that every admission refers to valid existing entities and preserves semantics while enabling efficient analytic joins.

---

5. Conclusion

Starting from a single wide admissions CSV, we identified the core entities – Patient, Doctor, Hospital, Insurance Provider, Medical Condition, Medication, and the transactional tables Admission and TestResult. We then:

- Defined the functional dependencies and primary keys for each relation.
- Ensured that in every relation, all non-key attributes are fully dependent on the key, with no partial or transitive dependencies.
- Showed how the decomposition eliminates classic update, insert, and delete anomalies present in the original denormalized design.
- Preserved the original meaning of the data using foreign keys, while enabling flexible analytics.