كلية العلوم

قسـم الانظمة الطبية الذكية

Lec 7 & 8

Databases in Healthcare — SSMS Implementation (PK / FK / Constraints)

Subject: Electronic Health Records

Level: fourth

Lecturer: Asst. Lecturer Qusai AL-Durrah

Why Data Integrity Is a Clinical Imperative

In healthcare, a data error is not merely an IT inconvenience — it can directly harm a patient. An EHR system that stores duplicate patient records, mislinks a lab result to the wrong encounter, or accepts an impossible heart-rate value can trigger wrong diagnoses, inappropriate medications, or missed alerts.

Accurate

Every field stores the correct value — no outliers, no garbage data allowed through the front door.

Consistent

The same clinical fact looks identical across every table, every query, every report.

Traceable

Every record carries a timestamp and a chain of identifiers back to the originating encounter and patient.

Secure

Structure itself is a security control — invalid relationships are rejected before they are stored.

Core Principle: A database is the backbone of every EHR — it controls structure, integrity, and querying. Bad database design equals wrong clinical decisions.

Learning Outcomes

By the end of this lecture you will be able to:

01

Explain Relational Concepts

Describe tables, entities, and relationships in a healthcare context — and why the relational model fits clinical data.

02

Define PK, FK & Constraints

State the purpose of primary keys, foreign keys, and each core constraint type, with a clinical rationale for each.

03

Create a Database in SSMS

Write and execute T-SQL to create a named database, select it as active, and scaffold multiple linked tables.

04

Apply All Constraint Types

Implement NOT NULL, UNIQUE, CHECK, DEFAULT, and FK constraints with correct T-SQL syntax inside a CREATE TABLE statement.

05

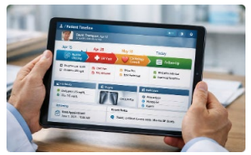
Validate with Test Inserts

Run INSERT statements that prove constraints work — including deliberate violations — and interpret SSMS error messages.

Why Databases Matter in Healthcare

Prevent Duplicate Patients

Two records for the same person = split clinical history. A UNIQUE constraint on a national ID stops this automatically at the database level.



Preserve Clinical Meaning Over Time

Visits, lab results, and medications must remain correctly linked across months or years of care — foreign keys make this guarantee structural, not procedural.

Support Evidence-Based Queries

Clinicians and analysts need to filter, join, and aggregate data (e.g., "all diabetic patients over 60 with an HbA1c above 8"). Only a well-structured relational database enables this reliably.

Enforce Rules Automatically

Constraints move validation logic from the application layer into the database engine — rules apply regardless of which app, script, or user inserts data.

The Relational Model: Healthcare Entities

In a relational database, every real-world concept is represented as a table (entity), and every link between concepts is represented by a foreign key (relationship). The six core entities of an EHR map directly onto clinical reality:

Core Entities (Tables)

- Patient — the person receiving care
- Encounter — a single clinical visit or admission
- Diagnosis — a coded clinical finding recorded at an encounter
- MedicationOrder — a drug prescribed during an encounter
- LabOrder — a laboratory test requested at an encounter
- LabResult — the numerical or textual result of a lab order

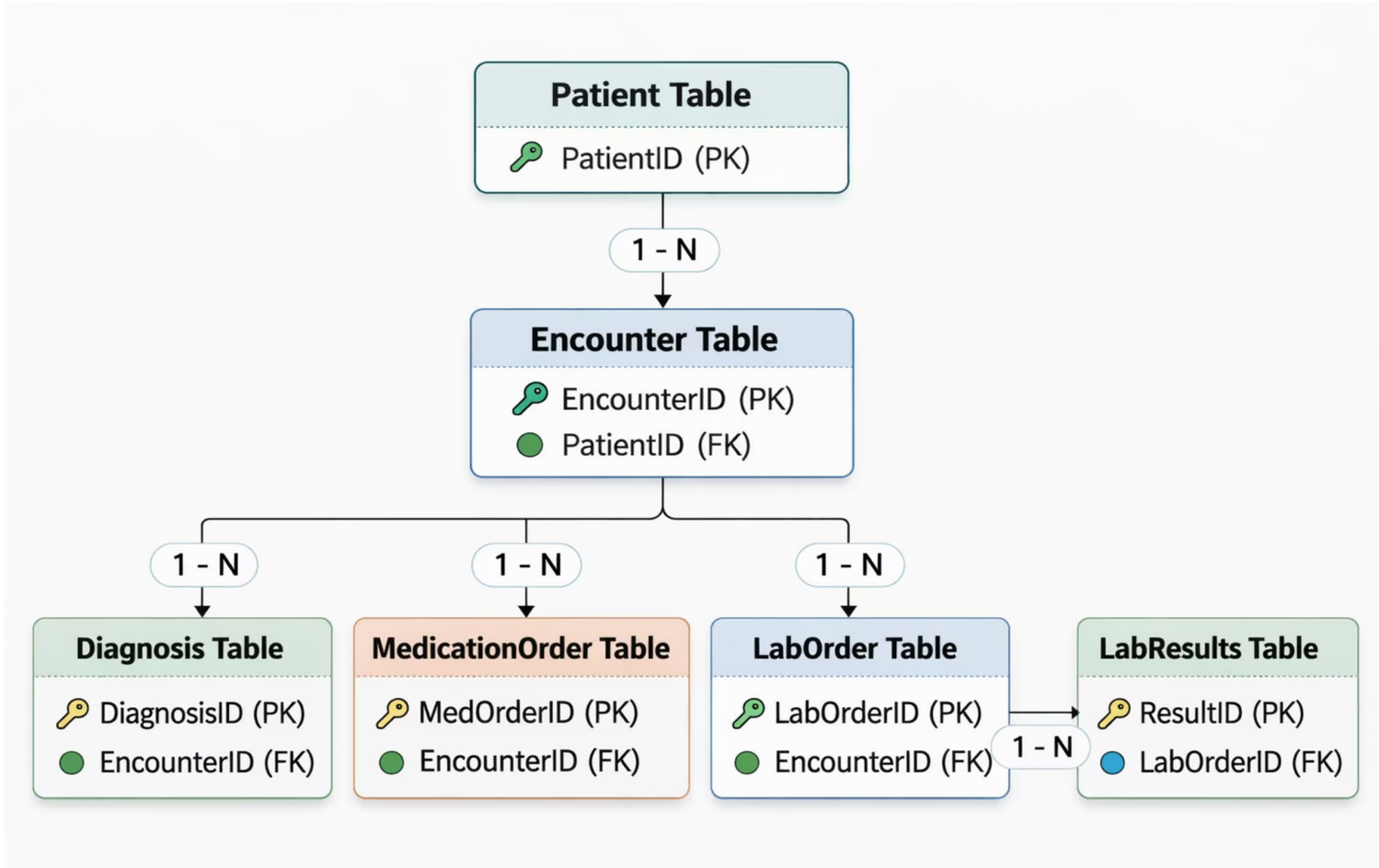
Relationships (Foreign Keys)

- Patient 1 → many Encounters — one patient can have many visits
- Encounter 1 → many Diagnoses — one visit can yield several diagnoses
- Encounter 1 → many MedicationOrders — multiple drugs per visit
- Encounter 1 → many LabOrders — multiple tests per visit
- LabOrder 1 → many LabResults — one order, potentially multiple result rows

Rule of thumb: if you see "one → many," you need a foreign key from the many side pointing to the one side.

Entity-Relationship Diagram: Mini-EHR Schema

The diagram below shows the four tables you will implement in SSMS today. Every arrow represents a foreign-key relationship. Read each arrow as: "this table's FK must match an existing PK in the parent table."



This is the schema you will implement step-by-step in Part D. Notice that Patient is the root every other table traces its ancestry back to a patient record through the chain of foreign keys.

Primary Key (PK): The Row's Unique Identity

What a PK Must Guarantee

- Uniqueness: No two rows in the table can share the same PK value — ever.
- Non-null: A PK column can never be left empty. SQL Server enforces this automatically.
- Stability: Once assigned, a PK value should never change — other tables store it as a reference.
- Single-purpose: The PK identifies the row; it carries no clinical meaning itself (prefer surrogate integers over meaningful codes).

Why not use FullName as a PK? Names are not unique (two patients named "Ali Hassan"), can change (marriage, transliteration), and are slow to join on. Always prefer a system-generated integer.

Healthcare Implications

Without a true PK, two records for different patients can receive the same identifier — their entire clinical histories merge. This is a patient safety event, not just a data quality issue.

Implementation in T-SQL

```
-- IDENTITY generates the PK value
-- automatically — no application code needed
PatientID INT IDENTITY(1,1)
  CONSTRAINT PK_Patients PRIMARY KEY
```

IDENTITY(1,1) means: start at 1, increment by 1 each insert. SQL Server assigns the value — you never supply it manually.

Foreign Key (FK): Encoding Clinical Relationships

A Foreign Key is a column in a child table whose value must match an existing Primary Key value in a parent table. It is the mechanism by which the relational model encodes real-world relationships — and it is enforced automatically by the database engine at every INSERT, UPDATE, and DELETE.

Prevents Orphan Records

Without an FK, you could insert an Encounter for `PatientID = 999` even if no such patient exists. The FK rejects this insert with a constraint violation error — the record never reaches storage.

Guarantees Referential Integrity

Every clinical event (diagnosis, medication order, lab) is guaranteed to be anchored to a real, existing encounter. You can always trace a result back to a visit, and a visit back to a patient.

T-SQL Syntax

The FK is declared at the bottom of a `CREATE TABLE` statement as a named constraint, making it easy to identify in error messages and to drop/modify later:

```
CONSTRAINT FK_Encounters_Patients
 FOREIGN KEY (PatientID)
 REFERENCES dbo.Patients(PatientID)
```

PK vs. FK: Side-by-Side Comparison

Dimension	Primary Key (PK)	Foreign Key (FK)
Purpose	Uniquely identifies each row in its own table	Links a row in a child table to a row in a parent table
Location	Lives in the parent (referenced) table	Lives in the child (referencing) table
Null allowed?	Never — SQL Server enforces NOT NULL automatically	Optional: if NULL is allowed, the row is simply unlinked
Uniqueness	Must be unique across all rows in the table	Can repeat — many encounters can share the same PatientID
Duplicate values	Rejected — constraint violation error	Allowed and expected (one patient → many encounters)
Healthcare example	`Patients.PatientID` identifies one unique patient record	`Encounters.PatientID` links every visit back to that patient
Referential rule	Other tables reference this column	Value must already exist in the parent's PK column

What Are Constraints?

Definition

A constraint is a rule embedded inside the database schema that the engine enforces automatically on every write operation (INSERT, UPDATE, DELETE). No application code, trigger, or manual check is needed — the database engine refuses any operation that would violate a constraint.

Why Not Validate in the Application?

- Multiple apps, scripts, and interfaces write to the same DB
- Application-layer validation can be bypassed or forgotten
- Database-level constraints apply universally, 24/7, regardless of source
- Constraint violations produce structured, loggable error codes

In an EHR Context

Constraints are the database's "guardrails." They represent clinical rules encoded as schema logic:

- A patient's date of birth cannot be missing
- A heart rate of –5 or 900 cannot be stored
- An encounter cannot exist without a parent patient
- The same national ID cannot appear twice

Each of these is a one-line SQL constraint — not pages of application code.

Core Constraint Types: Definitions & Healthcare Examples

NOT NULL

Definition: The column must always contain a value; an empty (NULL) insert is rejected.

Syntax: Add `NOT NULL` after the data type in the column definition.

Healthcare use: `DateOfBirth DATE NOT NULL` — you cannot register a patient without a birth date, as it drives age-based clinical rules.

UNIQUE

Definition: No two rows may store the same value in this column (NULLs are typically allowed and treated as distinct).

Syntax: `CONSTRAINT UQ_TableName_Column UNIQUE` after the column definition.

Healthcare use: `CONSTRAINT UQ_Patients_NationalID UNIQUE` on NationalID — prevents registering the same person twice under different records.

CHECK

Definition: Accepts a row only if a Boolean expression evaluates to TRUE. Rows where the expression is FALSE are rejected; NULL evaluates to UNKNOWN (typically passes).

Syntax: `CONSTRAINT CK_Table_Column CHECK (expression)`

Healthcare use: `CHECK (Sex IN ('M','F'))` — enforces a controlled vocabulary; `CHECK (HeartRate BETWEEN 20 AND 250)` — rejects physiologically impossible values.

DEFAULT

Definition: Automatically supplies a value when the INSERT statement omits the column. Does not prevent NULL if NULL is explicitly inserted.

Syntax: `CONSTRAINT DF_Table_Column DEFAULT value`

Healthcare use: `DEFAULT SYSDATETIME()` on `CreatedAt` — every record is automatically stamped with its creation time, supporting audit trails without requiring the inserting app to supply a timestamp.

FOREIGN KEY

Definition: Enforces referential integrity — the value in the child column must match an existing PK value in the parent table, or be NULL (if the column allows NULL).

Syntax: `CONSTRAINT FK_Child_Parent FOREIGN KEY (col) REFERENCES dbo.Parent(PK_col)`

Healthcare use: `FK_Encounters_Patients` — guarantees every encounter record belongs to a real, registered patient.

Constraint Design: Critical Thinking

Knowing how to write constraints is only half the skill — knowing when to apply them, and how strictly, is the design judgment that separates good schemas from brittle ones.

✅ DO: Constrain Safety-Critical Fields

Apply NOT NULL to every field that a clinical rule or downstream query depends on. Apply CHECK to numeric ranges that have known physiological limits (heart rate, SpO_2, weight). Apply UNIQUE to true business identifiers (MRN, NationalID).

⚠️ CAUTION: Don't Over-Constrain Early

Real-world healthcare data is messy. Phone numbers come in dozens of formats; addresses vary internationally; some fields are genuinely optional at intake. An overly strict CHECK on a phone number format will block legitimate registrations. Start with essential constraints; tighten iteratively.

🎯 Strategy: Use Realistic Ranges

A CHECK constraint on heart rate should allow `BETWEEN 20 AND 250` — not a "perfect" narrow band. Critically ill patients and measurement artifacts fall at the edges. Design for real clinical populations, not textbook examples.

Setting Up SSMS: Step-by-Step

Before You Write Any SQL

Follow these four steps every session to avoid the most common beginner mistake — running a script against the wrong database.

1 Open SSMS → Connect to Database Engine

Use Windows Authentication or supply your SQL Server credentials. Confirm the server name in the connection dialog matches your local instance (e.g., `localhost` or `.\SQLEXPRESS`).

2 Open a New Query Window

Click New Query in the toolbar, or press `Ctrl + N`. This opens a blank T-SQL editor pane connected to your server.

3 Run the Script in the Correct Order

Execute in sequence: (1) CREATE DATABASE, (2) USE, (3) CREATE TABLE statements, (4) INSERT test data. Running out of order causes "object not found" errors.

4 Save Your Script

Press `Ctrl + S` and name the file descriptively (e.g., `EHR_DB_Lecture7.sql`). You will reuse and submit this file as your homework artifact.

Key SSMS Concepts

- GO: A batch separator — it signals SSMS to send everything written above it to SQL Server as one execution batch. `CREATE DATABASE` must be in its own batch, so always follow it with `GO`.
- USE EHR_DB: Switches the active database context. All subsequent statements run against `EHR_DB` rather than the default `master` database.
- Execute selection: Highlight a portion of the script and press `F5` to run only that portion — useful for testing one table at a time.
- Error messages: Read the red error text carefully. It always tells you which constraint was violated, which table, and which column — this is your primary debugging tool.
- Object Explorer: Expand your database in the left panel to see tables, columns, and constraints visually after creation.

Tip: execute the script section by section (highlight → F5) rather than all at once. This makes it far easier to identify which statement caused an error.

Create the Database

The first two lines of every SSMS project are always the same: create the database container, then select it as the active context. Every subsequent statement runs inside this database.

```
-- ============================================================
-- STEP 1: Create the EHR database
-- CREATE DATABASE creates a new, empty database on the server.
-- The name EHR_DB will appear in Object Explorer on the left.
-- ============================================================
CREATE DATABASE EHR_DB;
GO
-- GO is a batch separator. It tells SSMS to send the CREATE DATABASE
-- statement to SQL Server before continuing. CREATE DATABASE MUST
-- be in its own batch — running it with other statements causes errors.

-- ============================================================
-- STEP 2: Switch context to the new database
-- Without USE, all your CREATE TABLE statements would run against
-- the default "master" database — a system database you should
-- never modify. Always USE your database before writing tables.
-- ============================================================
USE EHR_DB;
GO
```

CREATE DATABASE

Allocates a new database with default data and log files on disk. The database appears immediately in Object Explorer.

GO

Not a T-SQL keyword — it is an SSMS command. It flushes the current batch to the server. Required after CREATE DATABASE.

USE EHR_DB

Changes the active database for this connection. All subsequent DDL (CREATE TABLE) and DML (INSERT) run inside EHR_DB.

Patient Table: PK, UNIQUE, CHECK, DEFAULT

The `Patients` table is the root of the entire schema. Every constraint type is demonstrated here — study each annotation carefully before moving to child tables.

```
-- ============================================================
-- CREATE TABLE dbo.Patients
-- "dbo" is the default schema in SQL Server (database owner).
-- Always prefix table names with dbo. for clarity.
-- ============================================================
CREATE TABLE dbo.Patients
(
 -- IDENTITY(1,1): SQL Server automatically generates the PK value.
 -- Start = 1, Increment = 1. You NEVER insert this column manually.
 -- CONSTRAINT PK_Patients PRIMARY KEY: names the constraint so that
 -- SSMS shows a meaningful name in errors and Object Explorer.
 PatientID INT IDENTITY(1,1)
 CONSTRAINT PK_Patients PRIMARY KEY,

 -- UNIQUE constraint: two patients cannot share the same NationalID.
 -- NULL is allowed (some patients may not have a national ID yet),
 -- and SQL Server treats each NULL as distinct (no UNIQUE violation).
 NationalID NVARCHAR(20) NULL
 CONSTRAINT UQ_Patients_NationalID UNIQUE,

 -- NOT NULL: FullName must always be provided. An INSERT that omits
 -- FullName or passes NULL is rejected immediately.
 FullName NVARCHAR(100) NOT NULL,

 -- DATE type: stores only the date portion (no time). NOT NULL because
 -- date of birth drives age calculations in clinical decision support.
 DateOfBirth DATE NOT NULL,

 -- CHAR(1): single character. CHECK enforces a controlled vocabulary —
 -- only 'M' or 'F' are accepted. Any other value raises error 547.
 Gender CHAR(1) NOT NULL
 CONSTRAINT CK_Patients_Gender  CHECK (Gender IN ('M', 'F')),

 -- NULL is fine for phone: not every patient has one at registration.
 Phone NVARCHAR(20) NULL,

 -- DATETIME2: high-precision date+time column (up to 7 decimal places).
 -- DEFAULT SYSDATETIME(): if the INSERT omits CreatedAt, SQL Server
 -- automatically fills it with the current server timestamp.
 -- This creates an automatic audit trail with zero application effort.
 CreatedAt DATETIME2 NOT NULL
 CONSTRAINT DF_Patients_CreatedAt
 DEFAULT SYSDATETIME()
);
```

Encounter Table: FK, CHECK on Type, DEFAULT

The `Encounters` table introduces the Foreign Key — the mechanism that ties every clinical visit back to a registered patient.

It also shows how CHECK constraints enforce controlled clinical vocabularies.

```
-- ============================================================
-- CREATE TABLE dbo.Encounters
-- Child of dbo.Patients via FK on PatientID.
-- Must be created AFTER dbo.Patients (parent must exist first).
-- ============================================================
CREATE TABLE dbo.Encounters
(
    -- Auto-generated surrogate PK for this encounter.
    EncounterID    INT          IDENTITY(1,1)
                   CONSTRAINT PK_Encounters PRIMARY KEY,

    -- FK column: stores the PatientID of the patient this visit belongs to.
    -- NOT NULL: every encounter MUST belong to a patient.
    PatientID      INT          NOT NULL,

    -- DEFAULT SYSDATETIME(): if no date is supplied, the current timestamp
    -- is recorded. Useful when the application inserts in real time.
    EncounterDate  DATETIME2    NOT NULL
                   CONSTRAINT DF_Encounters_Date
                   DEFAULT SYSDATETIME(),

    -- CHECK constraint enforces three allowed encounter types:
    -- OPD = Outpatient, ER = Emergency Room, IPD = Inpatient.
    -- Any other string (e.g., 'CLINIC') causes error 547.
    EncounterType  NVARCHAR(3)  NOT NULL
                   CONSTRAINT CK_Encounters_Type
                   CHECK (EncounterType IN ('OPD', 'ER', 'IPD')),

    -- Free-text clinical notes. NULL is allowed — clinicians may not
    -- always add notes at the time of the encounter creation.
    Notes          NVARCHAR(4000) NULL,

    -- FOREIGN KEY CONSTRAINT (declared at table level, after all columns):
    -- PatientID in this table MUST match an existing PatientID in dbo.Patients.
    -- If you try to insert PatientID = 999 and no such patient exists → error 547.
    -- If you try to DELETE a patient who has encounters → error 547 (cascade not set).
    CONSTRAINT FK_Encounters_Patients
        FOREIGN KEY (PatientID)
        REFERENCES dbo.Patients(PatientID)
);
```

Table creation order matters. A FK can only reference a table that already exists. Always create parent tables before child tables. Here: Patients first, then Encounters.

Diagnoses & MedicationOrders: More FKs + Date CHECK

These two tables both hang off `Encounters`. The `MedicationOrders` table introduces a cross-column CHECK constraint — verifying that `EndDate` is never earlier than `StartDate`.

```
-- ============================================================
-- TABLE 1: dbo.Diagnoses — child of dbo.Encounters
-- ============================================================
CREATE TABLE dbo.Diagnoses
(
 DiagnosisID INT IDENTITY(1,1)
 CONSTRAINT PK_Diagnoses PRIMARY KEY,

 -- FK to Encounters: a diagnosis MUST belong to a specific visit.
 EncounterID INT NOT NULL,

 -- Stores ICD-10 codes (e.g., 'E11', 'J18.9'). NOT NULL: a diagnosis
 -- record without a code has no clinical value.
 DiagnosisCode NVARCHAR(20) NOT NULL,
 DiagnosisText NVARCHAR(255) NOT NULL,

 -- Auto-timestamp: records when the diagnosis was entered.
 DiagnosedAt DATETIME2 NOT NULL
 CONSTRAINT DF_Diagnoses_Date
 DEFAULT SYSDATETIME(),

 CONSTRAINT FK_Diagnoses_Encounters
 FOREIGN KEY (EncounterID)
 REFERENCES dbo.Encounters(EncounterID)
);


-- ============================================================
-- TABLE 2: dbo.MedicationOrders — child of dbo.Encounters
-- ============================================================
CREATE TABLE dbo.MedicationOrders
(
 MedOrderID INT IDENTITY(1,1)
 CONSTRAINT PK_MedOrders PRIMARY KEY,

 EncounterID INT NOT NULL,
 DrugName NVARCHAR(100) NOT NULL,
 Dose NVARCHAR(50) NOT NULL,

 -- CHECK: only four allowed routes of administration.
 -- PO = by mouth, IV = intravenous, IM = intramuscular, SC = subcutaneous.
 Route NVARCHAR(2) NOT NULL
 CONSTRAINT CK_MedOrders_Route
 CHECK (Route IN ('PO', 'IV', 'IM', 'SC')),

 StartDate DATE NOT NULL,

 -- EndDate can be NULL (open-ended prescription / chronic medication).
 EndDate DATE NULL,

 -- CROSS-COLUMN CHECK: ensures EndDate is not before StartDate.
 -- "EndDate IS NULL" part: if EndDate is NULL (no end set), the check
 -- passes automatically — NULL means "still ongoing."
 -- If EndDate is provided, it must be >= StartDate.
 CONSTRAINT CK_MedOrders_Dates
 CHECK (EndDate IS NULL OR EndDate >= StartDate),

 CONSTRAINT FK_MedOrders_Encounters
 FOREIGN KEY (EncounterID)
 REFERENCES dbo.Encounters(EncounterID)
);
```

Test Inserts: Validating the Schema in SSMS

After creating all tables, run the following INSERTs in order. Each builds on the previous — you must have a patient before an encounter, an encounter before a diagnosis, and so on.

```
-- ============================================================
-- INSERT 1: Register a new patient.
-- We do NOT supply PatientID (IDENTITY handles it → will be 1).
-- We do NOT supply CreatedAt (DEFAULT SYSDATETIME() fills it in).
-- ============================================================
INSERT INTO dbo.Patients (NationalID, FullName, DateOfBirth, Sex, Phone)
VALUES ('IQ123456', 'Ali Hassan', '2002-05-10', 'M', '07701234567');

-- ============================================================
-- INSERT 2: Record a clinical encounter for PatientID = 1.
-- PatientID = 1 must already exist in dbo.Patients (FK check).
-- EncounterDate is omitted → DEFAULT fills current timestamp.
-- ============================================================
INSERT INTO dbo.Encounters (PatientID, EncounterType, Notes)
VALUES (1, 'OPD', 'Follow-up visit for diabetes management');

-- ============================================================
-- INSERT 3: Add a diagnosis to EncounterID = 1.
-- EncounterID = 1 must exist in dbo.Encounters (FK check).
-- DiagnosedAt omitted → DEFAULT fills timestamp automatically.
-- ============================================================
INSERT INTO dbo.Diagnoses (EncounterID, DiagnosisCode, DiagnosisText)
VALUES (1, 'E11', 'Type 2 Diabetes Mellitus');

-- ============================================================
-- INSERT 4: Prescribe a medication for EncounterID = 1.
-- Route 'PO' passes the CHECK (PO, IV, IM, SC are allowed).
-- EndDate is omitted (NULL) → chronic/open prescription.
-- CK_MedOrders_Dates passes because EndDate IS NULL.
-- ============================================================
INSERT INTO dbo.MedicationOrders (EncounterID, DrugName, Dose, Route,
StartDate)
VALUES (1, 'Metformin', '500 mg', 'PO', '2026-03-01');

-- ============================================================
-- VERIFY: Run these SELECT statements to confirm data is stored.
-- ============================================================
SELECT * FROM dbo.Patients;
SELECT * FROM dbo.Encounters;
SELECT * FROM dbo.Diagnoses;
SELECT * FROM dbo.MedicationOrders;
```

Deliberate Violations: Proving Constraints Work

The best way to confirm your constraints are active is to deliberately break them and observe the SSMS error. Run each statement below individually.

Each one should fail — if it succeeds, your constraint was not created correctly.

```
-- ============================================================
-- VIOLATION 1: FK violation — PatientID 999 does not exist.
-- Expected error 547: "The INSERT statement conflicted with
-- the FOREIGN KEY constraint FK_Encounters_Patients."
-- ============================================================
INSERT INTO dbo.Encounters (PatientID, EncounterType)
VALUES (999, 'OPD');   -- ERROR: no patient with ID 999

-- ============================================================
-- VIOLATION 2: CHECK violation — 'CLINIC' is not in ('OPD','ER','IPD').
-- Expected error 547: "conflicted with the CHECK constraint
-- CK_Encounters_Type."
-- ============================================================
INSERT INTO dbo.Encounters (PatientID, EncounterType)
VALUES (1, 'CLINIC');  -- ERROR: invalid encounter type

-- ============================================================
-- VIOLATION 3: CHECK violation — Sex 'X' not in ('M','F').
-- Expected error 547: "conflicted with the CHECK constraint
-- CK_Patients_Sex."
-- ============================================================
INSERT INTO dbo.Patients (FullName, DateOfBirth, Sex)
VALUES ('Test Patient', '2000-01-01', 'X');  -- ERROR: invalid sex value
```

```
-- ============================================================
-- VIOLATION 4: NOT NULL violation — FullName cannot be NULL.
-- Expected error 515: "Cannot insert the value NULL into column
-- 'FullName', table 'EHR_DB.dbo.Patients'."
-- ============================================================
INSERT INTO dbo.Patients (FullName, DateOfBirth, Sex)
VALUES (NULL, '2000-01-01', 'F');  -- ERROR: NULL not allowed

-- ============================================================
-- VIOLATION 5: Cross-column CHECK — EndDate before StartDate.
-- Expected error 547: "conflicted with the CHECK constraint
-- CK_MedOrders_Dates."
-- ============================================================
INSERT INTO dbo.MedicationOrders (EncounterID, DrugName, Dose, Route, StartDate,
EndDate)
VALUES (1, 'Aspirin', '100 mg', 'PO', '2026-03-10', '2026-03-01');  -- ERROR: EndDate <
StartDate
```

When a constraint violation occurs, no partial data is written. The entire INSERT is rolled back. Read error 547 messages carefully — they always name the constraint that was violated.

Common Mistakes & How to Avoid Them

❌ Using FullName as a Primary Key

Problem: Names are not unique (two patients named "Sara Ahmed"), can change over time (marriage, spelling correction), and are slow to join on as text. Fix: Always use `INT IDENTITY` as a surrogate PK. The name stays in a separate `FullName` column.

❌ Storing Multiple Medications in One Text Field

Problem: A column like `Medications NVARCHAR(500)` storing "Metformin 500mg, Aspirin 100mg" cannot be queried, filtered, or aggregated. You cannot find all patients on Metformin. Fix: One row per medication in `MedicationOrders`.

❌ Missing Foreign Keys → Orphan Clinical Records

Problem: Without an FK, you can insert a Diagnosis for `EncounterID = 500` that does not exist — a dangling record with no clinical context. Joins will silently return no results. Fix: Always declare FK constraints; let the engine enforce referential integrity.

❌ No CHECK Constraints → Garbage Values

Problem: Without `CHECK (HeartRate BETWEEN 20 AND 250)`, values of `-5`, `0`, or `9999` enter the database and corrupt analytics. Clinical decision support rules based on these values fire incorrectly. Fix: Add CHECK constraints for every numeric field with known physiological bounds.

❌ Creating Child Tables Before Parent Tables

Problem: If you run `CREATE TABLE dbo.Encounters` before `CREATE TABLE dbo.Patients`, SQL Server cannot resolve the FK reference and throws an error. Fix: Always create tables in dependency order: root parents first, leaf children last.

HOMEWORK

Homework Assignment: LabOrders & LabResults

You will extend the `EHR_DB` schema by adding two new tables that complete the laboratory workflow. Your implementation must follow every constraint principle covered in this lecture.

01

Create dbo.LabOrders (linked to Encounters)

Design a `LabOrders` table with: a surrogate PK (`LabOrderID INT IDENTITY`), a FK to `dbo.Encounters(EncounterID)` with NOT NULL, a `TestName NVARCHAR(100) NOT NULL`, an `OrderedAt DATETIME2` with a DEFAULT of `SYSDATETIME()`, and a `Priority` column with a CHECK constraint limiting values to `'ROUTINE'`, `'URENT'` , or `'STAT'`.

02

Create dbo.LabResults (linked to LabOrders)

Design a `LabResults` table with: a surrogate PK, a FK to `dbo.LabOrders(LabOrderID)` with NOT NULL, a `ResultValue NVARCHAR(50) NOT NULL`, a `Unit NVARCHAR(20) NULL`, a `ReferenceRange NVARCHAR(50) NULL`, a `ResultedAt DATETIME2` with DEFAULT, and a `Status` column with a CHECK constraint for `'PENDING'`, `'FINAL'`, or `'CORRECTED'`.

03

Test with INSERT Statements

Insert at least one `LabOrder` linked to an existing encounter, and at least one `LabResult` linked to that order. Also run at least two deliberate violation INSERTs (one FK violation, one CHECK violation) and paste the resulting SSMS error messages into your report.

04

Submit Two Deliverables

(a) A 1-page design rationale explaining each constraint choice — why each NOT NULL, CHECK, and DEFAULT was selected, with a clinical justification.
(b) A clean `.sql` script file that runs without errors on a fresh `EHR_DB` instance.

Lecture Summary: Key Takeaways

Databases Are a Patient Safety Tool

Structure, integrity rules, and relational design are not IT concerns — they directly determine whether clinical data can be trusted for care decisions.

PK = Identity, FK = Relationship

Every table needs a surrogate integer PK. Every inter-table link needs a FK. Together they prevent duplicate identities and orphan clinical records.

Constraints Are Automatic Guardrails

NOT NULL, UNIQUE, CHECK, DEFAULT, and FK constraints enforce data quality at the engine level — regardless of which application or user writes the data.

Name Every Constraint

Use the `CONSTRAINT CK_Table_Column` naming pattern. Named constraints appear in error messages, Object Explorer, and ALTER TABLE statements — making schemas maintainable.

Design Thoughtfully, Not Maximally

Constrain what matters for safety and querying. Start with essential rules; refine iteratively as you understand the real-world data you will receive.

Remember: Bad database design = wrong clinical decisions. Every constraint you add is a clinical safety net embedded permanently in the system architecture.

Thank you

Google Classroom