1. Q

Foreign Key Constraint is nothing, but a rule used in DBMS to link one relation (a table) with another relation. It has certain rules to follow, as such that the values in foreign key field of one relation must match with the valid values in the primary ley field of the other relation, to ensure the tables and the relations remain consistent, and error free.

For example:
Students table stores information about students.

Students (sid CHAR(20) , name CHAR (30) , login CHAR(20) , age INTEGER, gpa REAL)
Primary key: sid (student ID)

Enrolled table: stores information about which students are enrolled in courses.

Enrolled (studid: string, cid: string, gTade: string)
Foreign key: studid (student ID from the Enrolled table, referring to the sid in Students).

This constraint ensures that any studid in the Enrolled table must correspond to an existing sid in the Students table. For example, inserting a record into Enrolled with a studid value that doesn't exist in Students violates the constraint, and the DBMS will reject this insertion.

Significance of Foreign Key Constraints:

1. Foreign key constraints are significant as they ensure referential integrity in a database, which is crucial for maintaining consistency and accuracy of data. Without such constraints, the database could contain invalid relationships, leading to inaccurate or misleading data. Here are a few reasons why foreign key constraints are important:
2. Prevents Invalid Data: For example, if I try to insert a student ID into the Enrolled table that doesn't exist in the Students table, the foreign key constraint will prevent this, ensuring that only legitimate students can enroll in courses.
3. When a record in the referenced table (Students) is deleted or updated, the foreign key constraint ensures the dependent records in the referencing table (Enrolled) are either updated, deleted, or the action is restricted to prevent breaking the relationship.
4. Improved Data Integrity: By enforcing relationships between tables, foreign key constraints help ensure that related records always match valid, existing entities. This prevents orphaned records or references to non-existent records.

Referential Integrity:
Referential integrity means, that every foreign key value in the referencing table (e.g., Enrolled) must correspond to a valid primary key value in the referenced table (e.g., Students). This ensures that relationships between records in the database are preserved.

For example, in the context of the Students and Enrolled tables:

1. If a student (with sid=53666) exists in the Students table and is referenced in the Enrolled table, deleting that student will break the foreign key constraint. The database system can prevent the deletion or take specific actions like cascading the delete to remove the corresponding Enrolled record.
2. If we have a Partner field in the Students table. The student does not yet have a partner or the partner is unknown, a null value can be used in the partner field, which is valid for foreign keys but not allowed in primary key fields.

Example of Foreign Key Constraint Violation:
If we try to insert the tuple (55555, Art104, A) into Enrolled, it will violate the foreign key constraint because there is no student with sid=55555 in the Students table. Similarly, deleting a student with sid=53666 who is enrolled in a course (recorded in Enrolled) would also violate the foreign key constraint unless proper actions like cascading deletes or restricting the deletion are in place.

Foreign key constraints play a critical role in maintaining referential integrity between tables, ensuring that relationships between data remain accurate and consistent and error free.

2.Q

Given relation schema as follows,

Students(sid: string, name: string, login: string, age: integer, gpa: real)

Faculty(fid: string, fname: string, sal: real)

Courses(cid: string, cname: string, credits: integer)

Rooms(rno: integer, address: string, capacity: integer)

Enrolled(sid: string, cid: string, grade: string)

Teaches (fid: string, cid: string)

Meets In (cid: string, rno: integer, time: string)

1.Answer,

Firstly, we have to learn about the PK's and FK's of this schema, only then we can define the Foreign Key Constraints,

Students: sid (PK)

Faculty: fid ((PK)

Courses: cid (PK)

Rooms: rno (PK)

Enrolled Relation

FK1: (sid) REFERENCES Students(sid)

Enrolled.sid → Students.sid

Every sid in the Enrolled table must match a valid sid from the Students table, to ensure that only legitimate students can be enrolled in courses.

FK2: (cid) REFERENCES Courses(cid)

Enrolled.cid → Courses.cid

Every cid in the Enrolled table must match a valid cid from the Courses table to ensure that students are only enrolled in valid courses.

Teaches Relation

FK1: (fid) REFERENCES Faculty(fid)

Teaches.fid → Faculty.fid

Every fid in the Teaches table must match a valid fid from the Faculty table to make sure that only valid faculty members can teach the courses.

FK2: (cid) REFERENCES Courses(cid)

Teaches.cid → Courses.cid

Every cid in the Teaches table must match a valid cid from the Courses table such that the courses being taught are valid courses listed in the Courses table.

Meets In Relation

FK1: (cid) REFERENCES Courses(cid)

Meets_In.cid → Courses.cid

Every cid in the Meets_In table must match a valid cid from the Courses table so, that only valid courses have scheduled meeting times and rooms.

FK2: (rno) REFERENCES Rooms(rno)

Meets_In.rno → Rooms.rno

Every rno in the Meets_In table must match a valid rno (room number) from the Rooms table. This ensures that courses are scheduled in valid rooms.

2.Answer,

We can introduce the Check constraint, since the question has mentioned it neither to be a primary key nor a foreign key.

Ex:

1. In Students relation,
   The age, which is not a PK, we need to have students who are age 15 or above in our college/university (for ex) then
   Alter Table Students
   Check (Age >=15);

2. In Courses relation,
   The credits which is not a PK, the credit must not be zero to the courses that are being taught.
   Alter Table Courses
   Check (Credits IS NOT NULL);

3. In Rooms Relation,
   The capacity of rooms which is not a PK, the size of room is in a way that it must fit more than 0 students.
   Alter Table Rooms
   Check (capacity > 0);

3. Q

Using SQL,

PATIENT

```
CREATE TABLE Patient (

   Patient_ID INT PRIMARY KEY,

   Name VARCHAR(255),

   SSN VARCHAR(11) UNIQUE,

   Sex CHAR(1),
   Age INT,

   DOB DATE,

   Address VARCHAR(255),

   ZipCode VARCHAR(10),

   InsuranceInfo VARCHAR(255)

);
```

DOCTOR

```
           CREATE TABLE Doctor (
              DOC_ID INT PRIMARY KEY,
              SSN VARCHAR(11) UNIQUE,
              Name VARCHAR(255),
              Specialty VARCHAR(255),
              YearExp INT
           );
```

DRUGS

```
CREATE TABLE Drugs (

   TradeName VARCHAR(255) PRIMARY KEY,


   Formula VARCHAR(255),

   Dosage INT,

   Price DECIMAL(10, 2),

   CautionNote TEXT,

   PharmaName VARCHAR(255),  -- Foreign key to Pharmaceutical Company

   TempStorage VARCHAR(255),

   Quantity INT,

   ExpiryDate DATE,

   FOREIGN KEY (PharmaName) REFERENCES PharmaCompany(Name) ON DELETE CASCADE

);
```

PHARMACOMPANY

```
CREATE TABLE PharmaCompany (

   Name VARCHAR(255) PRIMARY KEY,
   PhoneNum VARCHAR(15),
   Address VARCHAR(255),
   Zipcode VARCHAR(10)
);
```

PHARMACY

```
CREATE TABLE Pharmacy (
    PharmacyName VARCHAR(255) PRIMARY KEY,
    Address VARCHAR(255),
    Zipcode VARCHAR(10),
    PhoneNum VARCHAR(15)
);
```

PRESCRIPTION

```
CREATE TABLE Prescription (
    Prescription_ID INT PRIMARY KEY,
    Patient_ID INT,
    Doctor_ID INT,
    TradeName VARCHAR(255),  -- Foreign key to Drugs table
    Date DATE,
    Quantity INT,
    FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID),
    FOREIGN KEY (Doctor_ID) REFERENCES Doctor(DOC_ID),
    FOREIGN KEY (TradeName) REFERENCES Drugs(TradeName)
);
```

SUPERVISOR

```
CREATE TABLE Supervisor (
    Supervisor_ID INT PRIMARY KEY,
    Name VARCHAR(255),
    Contract_ID INT, -- Foreign key to Contract table
    FOREIGN KEY (Contract_ID) REFERENCES Contract(ContractID)
);
```

RELATIONSHIPS:

1. **PrimaryDoc (Doctor-Patient Relationship)**

```
CREATE TABLE PrimaryDoc (
  Patient_ID INT,
  Doctor_ID INT,
  PRIMARY KEY (Patient_ID),
  FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID),
  FOREIGN KEY (Doctor_ID) REFERENCES Doctor(DOC_ID)
);
```

2. **Prescribes (Doctor-Patient-Drugs-Prescription Relationship)**

```
CREATE TABLE Prescribes (
  Patient_ID INT,
  Doctor_ID INT,
  TradeName VARCHAR(255), -- Foreign key to Drugs table
  Prescription_ID INT, -- Foreign key to Prescription table
  PRIMARY KEY (Patient_ID, Doctor_ID, TradeName),
  FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID),
  FOREIGN KEY (Doctor_ID) REFERENCES Doctor(DOC_ID),
  FOREIGN KEY (TradeName) REFERENCES Drugs(TradeName),
  FOREIGN KEY (Prescription_ID) REFERENCES Prescription(Prescription_ID)
);
```

3. **Sells (Drugs-Pharmacy Relationship)**

```
CREATE TABLE Sells (
  PharmacyName VARCHAR(255),
  TradeName VARCHAR(255),
  Price DECIMAL(10, 2),  -- Price can vary by pharmacy (if required)
  PRIMARY KEY (PharmacyName, TradeName),
  FOREIGN KEY (PharmacyName) REFERENCES Pharmacy(PharmacyName),
  FOREIGN KEY (TradeName) REFERENCES Drugs(TradeName)
);
```

4. **Contract (Pharmacy-PharmaCompany Relationship**

```
CREATE TABLE Contract (
  ContractID INT PRIMARY KEY,
  PharmacyName VARCHAR(255),
  PharmaName VARCHAR(255),
  ContractText TEXT,
    StartDate DATE,
    EndDate DATE,
    FOREIGN KEY (PharmacyName) REFERENCES Pharmacy(PharmacyName),
    FOREIGN KEY (PharmaName) REFERENCES PharmaCompany(Name)
);
```

5. **Supervises (Supervisor-Contract Relationship)**

```
CREATE TABLE Supervises (
    Contract_ID INT,
    Supervisor_ID INT,
    PRIMARY KEY (Contract_ID),
    FOREIGN KEY (Contract_ID) REFERENCES Contract(ContractID),
    FOREIGN KEY (Supervisor_ID) REFERENCES Supervisor(Supervisor_ID)
);
```

Reference: The Database Management textbook by Ramakrishnan &Gehrke