

Prison Database

by Norman Heinzmann

1. Introduction

This project is about designing a database to support all core operations of a prison. It tracks current inmates, their personal details, program enrollment, cell assignments, and criminal history, their contact information via phone numbers, and any work or educational activities they participate in. Staff members (guards and administrative staff) are also stored, but with separate subtype tables, and their assignments to cells or departments are recorded. By fully normalizing the schema to 4NF, the design prevents data redundancy and ensures that complex business rules, like unique cell occupancy or scheduling guards, can be enforced reliably through transactions.

2. Database Use Cases

- **CRUD Operations:**

- *Create:*

- Add new prisoners along with their sentence records.
- Insert new staff members (guards or administrative staff).
- Create and assign cells (optionally with Storage).

- *Read:*

- Retrieve prisoner details and their sentence information.
- Display staff guard shift information.
- Query facility details including cell capacity or .

- *Update:*

- Modify prisoner records (sentence updates, adding program).
- Change cell capacity
- Adjust staff schedules

- *Delete:*

- Remove outdated or incorrect records while maintaining referential integrity
- Remove inmate specialization
- Remove attached storage

- **Possible Transactions:**

We want to add a new Inmate and give him his attributes, add his phone number to the Register, log his crimes and enroll him as a working- and educational-inmate. If one of these entries fail he can't make calls, is not registered in his work or can't participate in an educational course.

Because of that we need Transactions.

This example would look something like this:

```
BEGIN;
```

```
-- 1. Create the new inmate
```

```
INSERT INTO Inmate (FirstName, LastName, PostalCode, City, Street,  
BirthDate, ProgrammID, CellID)
```

```
VALUES ( 'Jonah', 'Richards', '10405', 'Berlin', 'Karl-Marx-Allee  
45', '1991-08-23', 3, 2)
```

```
RETURNING InmateID INTO :new_inmate_id;
```

```
-- 2. Record his phone number
```

```
INSERT INTO PhoneRegister (InmateID, PhoneNumber)
```

```
VALUES (:new_inmate_id, '+15551234567');
```

```
-- 3. Log the crime he has committed
```

```
-- (assume CrimeID = 4 is not used until now)
```

```
INSERT INTO HasCommitted (InmateID, CrimeID, CommittedDate)
```

```
VALUES (:new_inmate_id, 4, CURRENT_DATE);
```

```
-- 4. Enroll him in a work assignment (e.g., Kitchen Helper)
```

```
INSERT INTO WorkingInmate (InmateID, JobName, Hours,  
JobDescription )
```

```
VALUES ( :new_inmate_id, 'Bakery Assistant', 16.0, 'Helps prepare  
and package baked goods');
```

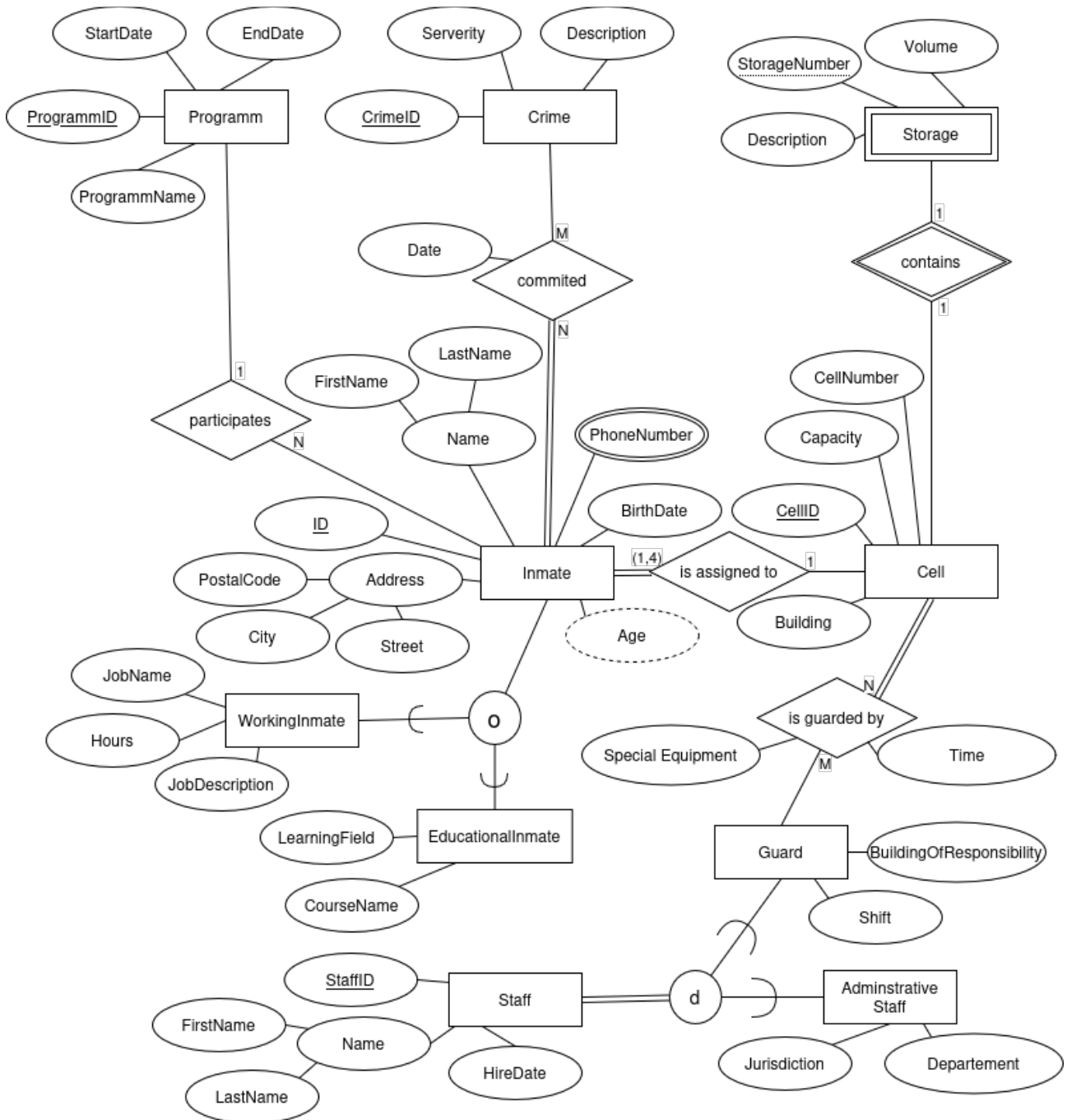
```
-- 5. Enroll him in an educational course
```

```
INSERT INTO EducationalInmate (InmateID, CourseName,  
LearningField)
```

```
VALUES (:new_inmate_id, 'Basic Literacy', 'Reading');
```

```
COMMIT;
```

3. ER Model



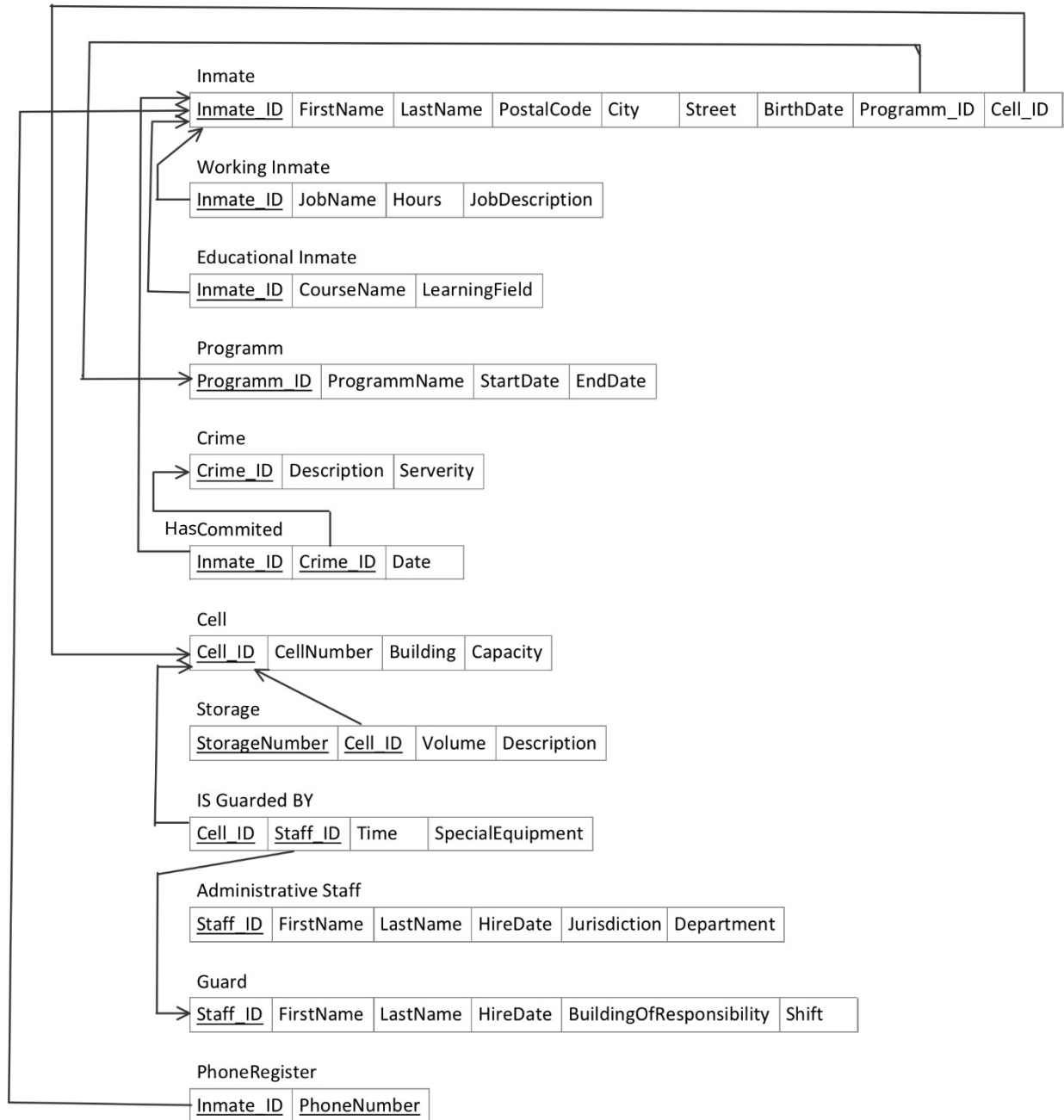
This is the ER-Model, it shows every Entity and every Relation in as the conceptual Schema.

The core entity is the inmate, which has various attributes, like multiple phone numbers. Also the inmates have the choice to work in the prison or take part in an educational course. Additionally every inmate can take part in one programm.

There is an entity for cells, which can contain one storage unit in form of a locker or a cabinet. Each Inmate has committed at least one crime and each inmate is assigned to exactly one cell. Multiple Inmates can commit the same crime and the crimes have the attribute Severity, which shows how grave it was, like "Infraction, Misdemeanor, Felony or Capital Crime".

Each cell can house between 1 and 4 inmates. Every cell must be guarded by one or more guards. The other type of Staff in the prison is the administrative staff, which manages operations and resources.

4. Relational Model



From ER-Model to the relational Model some changes have been made to satisfy 1NF.

The generic Staff table dropped because its members participate fully in exactly one subtype Guard or AdministrativeStaff, so no standalone supertype is needed.

Also the PhoneNumber attribute has been moved into a PhoneRegister lookup-table.

All composite address fields (PostalCode, City, Street) and name parts (FirstName, LastName) were flattened into separate columns. The 1:N relationship from inmate to programm is shown with a foreign key. The N:M relationship from inmate to crime is represented with a new lookup-table “committed” with both foreign key, which form a primary key. The 1:1 relationship from Cell to storage is represented in the storage table with the cell_id and StorageNumber as a composite primary key, because the StorageNumber on its own is just a weak primary key. The N:M relationship from cell to guard is represented with a lookup-table “isGuardedBy” with both foreign key, which form a primary key.

5. Normalization Steps

List of non-trivial dependencies

| | |
|----------------------------|--|
| Inmate_ID → | FirstName, LastName, PostalCode, City, Street, BirthDate , Programm_ID, Cell_ID |
| Inmate_ID → | JobName, Hours, JobDescription |
| Inmate_ID → | CourseName, LearningField |
| Crime_ID → | Description, Severity |
| (Inmate_ID, Crime_ID) → | Date |
| Cell_ID → | CellNumber, Building, Capacity |
| (Cell_ID, StorageNumber) → | Description, Volume |
| (Cell_ID, Staff_ID) → | Time, SpecialEquipment |
| Staff_ID → | FirstName, LastName, HireDate, Department, Jurisdiction |
| Staff_ID → | FirstName, LastName, HireDate, Shift, BuildingOfResponsibility |
| Programm_ID → | ProgrammName, StartDate, EndDate |

The first normal form (**1NF**) is achieved by removing the multi-valued attribute PhoneNumber from the Inmate relation and placing it into its own lookup table—PhoneRegister(Inmate_ID, PhoneNumber). By doing so, every attribute in every relation is atomic, and there are no nested or repeating groups.

To verify second normal form (**2NF**), one must ensure there are no partial dependencies—no non-key attribute should depend on only a part of a composite primary key. Examining each relationship with a composite key—namely (Inmate_ID, Crime_ID), (Cell_ID, StorageNumber), and (Cell_ID, Staff_ID)—reveals that neither Inmate_ID nor Crime_ID alone can determine Date, nor can Cell_ID alone determine Description and Volume, nor can Cell_ID alone determine Time and SpecialEquipment. Because no attribute in any of those composite relationships depends solely on a subset of the key, the schema already conforms to 2NF.

For third normal form (**3NF**), we must check for transitive dependencies: a situation where a non-key attribute depends on another non-key attribute that in turn depends on a key. The only dependency that connects to another entity is Inmate_ID → Name, Address, Programm_ID, Cell_ID. Since Programm_ID and Cell_ID are foreign keys referencing other tables, we need to ensure that neither Name nor Address is transitively determined by Programm_ID or Cell_ID. A review of the non-trivial dependency list shows that neither Name nor Address appears on the left side of any dependency; thus, no transitive dependency exists and the schema satisfies 3NF.

To confirm Boyce–Codd normal form (**BCNF**), every non-trivial functional dependency must have a determinant that is a superkey. Scanning the list of dependencies, we see that `Inmate_ID`, `Crime_ID`, `Cell_ID`, `Staff_ID`, and `Programm_ID` are all primary keys or components thereof, and composite keys such as (`Inmate_ID`, `Crime_ID`), (`Cell_ID`, `StorageNumber`), and (`Cell_ID`, `Staff_ID`) are also primary keys for their respective relations. Each of these determinants is therefore a superkey, and no functional dependency violates the BCNF condition. As a result, the entire schema is already in BCNF.

Finally, fourth normal form (**4NF**) prohibits the presence of non-trivial multivalued dependencies (MVDs) in a relation. Since the multi-valued attribute has been relocated to a lookup table and because no remaining relation has an attribute that can take on multiple independent values of another non-key attribute, there are no MVDs present. Consequently, the schema complies with 4NF.

In conclusion, after minor adjustments—specifically relocating `PhoneNumber` into its own lookup table—the relational model is in full compliance with 1NF, 2NF, 3NF, BCNF, and 4NF. It exhibits no partial or transitive dependencies, no determinant outside of a superkey, and no non-trivial multivalued dependencies, thereby fulfilling the highest standards of normalization.

6. Highlights

In the following I want to underline some features of the database, which makes it useful. One of its standout features is the use of role-based subtyping for staff management. By separating general staff information into a main “`Staff`” table and using subtypes like “`Guard`” and “`AdministrativeStaff`”, the database avoids data duplication while allowing detailed role-specific attributes. This design provides flexibility and clarity in managing different categories of personnel.

The schema is fully normalized up to the Fourth Normal Form, which ensures high data quality by eliminating redundancy and preventing update, insert, and delete anomalies. Multivalued attributes, such as phone numbers, are handled properly using lookup tables like “`PhoneRegister`”, further contributing to the clean and structured design.

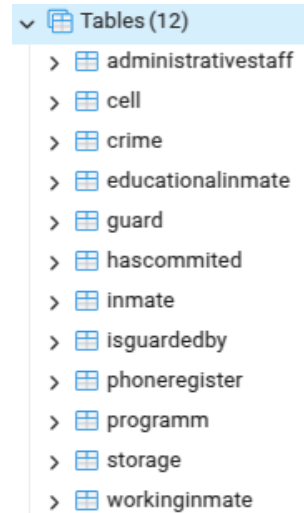
Another key feature is the database’s support for complex relationships and atomic transactions. Tables such as “`HasCommitted`” and “`IsGuardedBy`” are modeled with composite primary keys and enforce meaningful connections between inmates, crimes, cells, and guards.

The system also provides a strong foundation for both operational and analytical use. Daily activities like inmate tracking, staff scheduling, and program participation can be managed efficiently.

7. Actual usage and implementation of the Database

7.1 Creating every table, with the “Create Table” Command.

```
CREATE TABLE Inmate (  
    InmateID      SERIAL PRIMARY KEY,  
    FirstName     VARCHAR(50),  
    LastName      VARCHAR(50),  
    PostalCode    VARCHAR(20),  
    City          VARCHAR(50),  
    Street        VARCHAR(100),  
    BirthDate     DATE,  
    ProgrammID    INTEGER,  
    CellID        INTEGER,  
    FOREIGN KEY (ProgrammID) REFERENCES Programm(ProgrammID),  
    FOREIGN KEY (CellID) REFERENCES Cell(CellID)  
);
```



7.2 Second step is to insert our data with the “Insert Into” statement.

```
INSERT INTO Inmate (InmateID, FirstName, LastName, PostalCode,  
City, Street, BirthDate, Programm_ID, Cell_ID) VALUES  
(1, 'Samuel', 'Axelrod', '10115', 'Berlin', 'Spandauer Damm 10',  
'1985-06-12', 3, 15),  
(2, 'María', 'González', '20095', 'Hamburg', 'Kaiserstraße 42',  
'1992-11-03', NULL, 15);
```

7.3 After that we can create some Views to make the database easily useful for each usergroup:

7.3.1 View all cells

```
CREATE VIEW View_Cell_Inmates AS  
SELECT  
    c.CellID,  
    c.CellNumber,  
    c.Building,  
    ARRAY_AGG(i.FirstName || ' ' || i.LastName ORDER BY  
i.InmateID) AS InmateNames  
FROM  
    Cell c  
LEFT JOIN  
    Inmate i ON c.CellID = i.CellID  
GROUP BY  
    c.CellID,  
    c.CellNumber,  
    c.Building;
```

Now we can see each cell together with the inmates living there.

7.3.2 View all guards

```
CREATE VIEW View_Guard_Shifts AS
SELECT
    g.StaffID,
    g.FirstName || ' ' || g.LastName AS GuardName,
    igb.CellID,
    igb.ShiftTime,
    igb.SpecialEquipment
FROM
    Guard g
JOIN
    IsGuardedBy igb ON g.StaffID = igb.StaffID
ORDER BY
    igb.ShiftTime;
```

This would be very helpful for somebody controlling or scheduling the guards.

With this view you get the current guard assignment and their shifts.

An output would look like this:

| | staffid integer | guardname text | cellid integer | shifttime timestamp without time zone | specialequipment text |
|---|--------------------|-------------------|-------------------|--|--------------------------|
| 1 | 3 | Karim Diallo | 3 | 2025-06-01 14:00:00 | [null] |
| 2 | 5 | Rafael Costa | 5 | 2025-06-01 14:00:00 | Radio |
| 3 | 1 | Tobias Wolf | 1 | 2025-06-01 22:00:00 | Taser |
| 4 | 2 | Aylin Yilmaz | 2 | 2025-06-02 06:00:00 | Pepper Spray |
| 5 | 4 | Eva Schulz | 4 | 2025-06-02 22:00:00 | Handcuffs |

8. Summary

The creation of this prison database was a structured journey through the stages of professional relational design. We began by analyzing the real-world environment of a prison, identifying key entities such as inmates, staff, crimes, and cells. These were modeled using an EER diagram to capture all necessary relationships and complexities.

Next, we transformed the conceptual model into a relational schema, applying normalization rules up to 4NF to ensure data integrity, minimized redundancy, and supported efficient querying.

We implemented the database for realuse cases in postgresql and improved usability by creating external SQL views tailored for different user roles, such as monitoring cell occupancy or organizing guard shifts. The result is a clean, flexible, and well-structured system that reflects the operational needs of a modern prison.