Database Schema DDL and Normalization

Domain Definitions

```
-- Domain Definitions
CREATE DOMAIN GenderType AS VARCHAR(10)
    CHECK (VALUE IN ('Male', 'Female', 'Other'));

CREATE DOMAIN PriorityType AS VARCHAR(10)
    CHECK (VALUE IN ('Low', 'Medium', 'High', 'Critical'));

CREATE DOMAIN StatusType AS VARCHAR(20)
    CHECK (VALUE IN ('Open', 'In Progress', 'Resolved'));

CREATE DOMAIN PermissionType AS VARCHAR(10)
    CHECK (VALUE IN ('Read', 'Write'));
```

Table Definitions

Employee Table

```
-- Table: Employee
CREATE TABLE Employee (
    employeeid SERIAL PRIMARY KEY,
    firstname VARCHAR(100) NOT NULL,
    lastname VARCHAR(100) NOT NULL,
    gender GenderType NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    address TEXT,
    CHECK (email LIKE '%_@_%._%')
);
```

Projects Table

```
-- Table: Projects
CREATE TABLE Projects (
    projectid SERIAL PRIMARY KEY,
    projectname VARCHAR(255) NOT NULL,
    description TEXT,
    start_date DATE NOT NULL,
    end_date DATE,
    CHECK (end_date IS NULL OR end_date >= start_date)
);
```

Issues Table

```
-- Table: Issues
CREATE TABLE Issues (
    issueid SERIAL PRIMARY KEY,
    projectid INT NOT NULL REFERENCES Projects(projectid) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    status StatusType DEFAULT 'Open',
    priority PriorityType DEFAULT 'Medium'
);
```

IssueHistory Table

```
-- Table: IssueHistory
CREATE TABLE IssueHistory (
    historyid SERIAL PRIMARY KEY,
    issueid INT NOT NULL REFERENCES Issues(issueid) ON DELETE CASCADE,
    create_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    solution_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

IssueSolution Table

```
-- Table: IssueSolution
CREATE TABLE IssueSolution (
    solutionid SERIAL PRIMARY KEY,
    issueid INT NOT NULL REFERENCES Issues(issueid) ON DELETE CASCADE,
    employeeid INT NOT NULL REFERENCES Employee(employeeid) ON DELETE CASCADE,
    details TEXT NOT NULL
);
```

Roles Table

```
-- Table: Roles
CREATE TABLE Roles (
    roleid SERIAL PRIMARY KEY,
    rolename VARCHAR(100) UNIQUE NOT NULL
);
```

Permissions Table

```
-- Table: Permissions
CREATE TABLE Permissions (
    permissionname PermissionType PRIMARY KEY
);
```

EmployeeRoles Table

```
-- Table: EmployeeRoles
CREATE TABLE EmployeeRoles (
    employeeid INT NOT NULL REFERENCES Employee(employeeid) ON DELETE CASCADE,
    roleid INT NOT NULL REFERENCES Roles(roleid) ON DELETE CASCADE,
    PRIMARY KEY (employeeid, roleid)
);
```

EmployeePermissions Table

```
-- Table: EmployeePermissions
```

```
CREATE TABLE EmployeePermissions (
    employeeid INT NOT NULL REFERENCES Employee(employeeid) ON DELETE CASCADE,
    permissionname PermissionType NOT NULL REFERENCES Permissions(permissionname) ON DEL
    PRIMARY KEY (employeeid, permissionname)
);
```

EmployeeProjects Table

```
-- Table: EmployeeProjects
CREATE TABLE EmployeeProjects (
    employeeid INT REFERENCES Employee(employeeid) ON DELETE SET NULL,
    projectid INT NOT NULL REFERENCES Projects(projectid) ON DELETE CASCADE,
    PRIMARY KEY (employeeid, projectid)
);
```

Comments Table

```
-- Table: Comments
CREATE TABLE Comments (
    commentid SERIAL PRIMARY KEY,
    issueid INT NOT NULL REFERENCES Issues(issueid) ON DELETE CASCADE,
    comment_text TEXT NOT NULL,
    date_posted TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Attachments Table

```
-- Table: Attachments
CREATE TABLE Attachments (
    attachmentid SERIAL PRIMARY KEY,
    issueid INT REFERENCES Issues(issueid) ON DELETE SET NULL,
    file_path VARCHAR(255) NOT NULL,
    uploaded_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CHECK (file_path LIKE '%.%' AND LENGTH(file_path) > 4)
);
```

Functional Dependencies

Functional dependencies describe the relationship between attributes in a table. For each functional dependency $A \rightarrow B$, the value of attribute(s) A determines the value of attribute(s) B.

Table	Functional Dependencies
Employee	employeeid → {firstname, lastname, gender, email, address}
Projects	projectid → {projectname, description, start_date, end_date}
Issues	issueid → {projectid, title, description, status, priority}
IssueHistory	historyid → {issueid, create_date, solution_date}
IssueSolution	solutionid → {issueid, employeeid, details}
Roles	roleid → {rolename}
Permissions	permissionname → {}
EmployeeRoles	{employeeid, roleid} → {}
EmployeePermissions	{employeeid, permissionname} → {}
EmployeeProjects	{employeeid, projectid} → {}
Comments	commentid → {issueid, comment_text, date_posted}
Attachments	attachmentid → {issueid, file_path, uploaded_date}

Normalization

Normalization is the process of organizing attributes in a relational database to reduce redundancy and avoid undesirable characteristics such as update anomalies. We will now describe the normal forms and evaluate the schema for each of them.

First Normal Form (1NF)

A table is in First Normal Form (1NF) if it meets the following conditions:

- All attributes have atomic (indivisible) values.
- Each record has a unique identifier (Primary Key).

Evaluation: All tables in the schema are in 1NF because they have atomic values and primary keys.

Second Normal Form (2NF)

A table is in Second Normal Form (2NF) if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key. There should be no partial dependencies.

Evaluation:

- In the table `EmployeeRoles`, the composite primary key (`employeeid, roleid`) determines the table's non-key attributes. Therefore, it is in 2NF.
- Similarly, other tables with composite keys such as `EmployeePermissions` and `EmployeeProjects` satisfy 2NF.

Third Normal Form (3NF)

A table is in Third Normal Form (3NF) if it is in 2NF and all non-key attributes are non-transitively dependent on the primary key. That is, there should be no transitive dependencies between non-key attributes.

Evaluation:

- In the `Employee` table, attributes such as `firstname`, `lastname`, `email`, and `address` depend directly on `employeeid`, and there are no transitive dependencies.
- The `Roles` and `Permissions` tables also do not have any transitive dependencies.

Boyce-Codd Normal Form (BCNF)

A table is in Boyce-Codd Normal Form (BCNF) if it is in 3NF and every determinant is a candidate key.

Evaluation: All tables in the schema satisfy BCNF because every functional dependency has a determinant that is a candidate key.

Conclusion

The provided schema is well-normalized up to the Fifth Normal Form (5NF). The functional dependencies are straightforward, and all tables comply with 1NF, 2NF, 3NF, BCNF.