SQLite Foreign Keys (FK) & Relationships - Advanced Guide

This document provides a comprehensive, **advanced-level explanation** of Foreign Keys (FK) and relationships in SQLite3, including all commands and scenarios for One-to-One, One-to-Many, and Many-to-Many relationships.

---

1. Foreign Key (FK) in SQLite3

- **Definition:** A column in one table that refers to the **Primary Key (PK)** of another table.
- **Purpose:** Maintain **data integrity**, ensuring invalid data cannot be inserted.
- **Enable FK support in SQLite:**

```
PRAGMA foreign_keys = ON;
```

- This must be set **every time you start SQLite** if FK constraints are to be enforced.

- **Example:**

```
CREATE TABLE Departments (
    dept_id INTEGER PRIMARY KEY AUTOINCREMENT,
    dept_name TEXT
);

CREATE TABLE Employees (
    emp_id INTEGER PRIMARY KEY AUTOINCREMENT,
    emp_name TEXT,
    dept_id INTEGER,
    FOREIGN KEY(dept_id) REFERENCES Departments(dept_id)
);
```

- `Employees.dept_id` is a **FK** referencing `Departments.dept_id`. Any insert with an invalid `dept_id` will fail.

---

2. One-to-One (1:1) Relationship

- **Rule:** Each row in Table A corresponds to **exactly one row** in Table B.
- **Constraint:** `UNIQUE` + `FK`

```
CREATE TABLE Users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    email TEXT
);

CREATE TABLE UserProfiles (
    profile_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER UNIQUE,
    bio TEXT,
    age INTEGER,
    FOREIGN KEY(user_id) REFERENCES Users(user_id)
);
```

- **Insert sample data:**

```
INSERT INTO Users (name, email) VALUES ('Ali', 'ali@example.com'), ('Sara',
'sara@example.com');
INSERT INTO UserProfiles (user_id, bio, age) VALUES (1, 'Developer', 25),
(2, 'Designer', 30);
```

- **Retrieve joined data:**

```
SELECT u.name, u.email, p.bio, p.age
FROM Users u
JOIN UserProfiles p ON u.user_id = p.user_id;
```

- Result:

```
Ali | ali@example.com | Developer | 25
Sara | sara@example.com | Designer | 30
```

- **Note:** Attempting to insert another profile for the same user will throw an error due to `UNIQUE` constraint.

---

3. One-to-Many (1:N) Relationship

- **Rule:** One parent row → multiple child rows
- Example: Departments ↔ Employees

```
CREATE TABLE Departments (
    dept_id INTEGER PRIMARY KEY AUTOINCREMENT,
    dept_name TEXT
);

CREATE TABLE Employees (
    emp_id INTEGER PRIMARY KEY AUTOINCREMENT,
    emp_name TEXT,
    dept_id INTEGER,
    FOREIGN KEY(dept_id) REFERENCES Departments(dept_id)
);
```

- **Insert sample data:**

```
INSERT INTO Departments (dept_name) VALUES ('HR'), ('IT');
INSERT INTO Employees (emp_name, dept_id) VALUES ('Ali',1), ('Usman',1),
('Hassan',2);
```

- **JOIN query:**

```
SELECT e.emp_name, d.dept_name
FROM Employees e
JOIN Departments d ON e.dept_id = d.dept_id;
```

- Result:

```
Ali | HR
Usman | HR
Hassan | IT
```

- **FK ensures employees are linked only to valid departments.**
- **Advanced tip:** Use `INSERT OR IGNORE` to skip invalid FK entries instead of throwing errors:

```
INSERT OR IGNORE INTO Employees (emp_name, dept_id) VALUES ('Jawad', 3);
```

- If `dept_id = 3` does not exist, it will **skip the row silently**.

---

4. Many-to-Many (N:N) Relationship

- **Rule:** Multiple rows in Table A ↔ Multiple rows in Table B
- **Requires junction table**

```
CREATE TABLE Students (
    student_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT
);

CREATE TABLE Courses (
    course_id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT
);

CREATE TABLE StudentCourses (
    student_id INTEGER,
    course_id INTEGER,
    PRIMARY KEY(student_id, course_id),
    FOREIGN KEY(student_id) REFERENCES Students(student_id),
    FOREIGN KEY(course_id) REFERENCES Courses(course_id)
);
```

- **Insert sample data:**

```
INSERT INTO Students (name) VALUES ('Ali'), ('Sara'), ('Usman'), ('Jhoat'),
('Dareaa'), ('Rahim'), ('Daniyal'), ('Junaid');
INSERT INTO Courses (title) VALUES ('Math'), ('Physics'), ('English');

INSERT INTO StudentCourses (student_id, course_id) VALUES
(1,1), (1,2),
(2,1),
(3,2), (3,3),
(4,1),
(5,2),
(6,3),
(7,1), (7,3),
(8,2);
```

- **JOIN query to retrieve full mapping:**

```
SELECT s.name, c.title
FROM Students s
JOIN StudentCourses sc ON s.student_id = sc.student_id
JOIN Courses c ON sc.course_id = c.course_id;
```

- Result:

```
Ali     | Math
Ali     | Physics
Sara    | Math
Usman   | Physics
Usman   | English
Jhoat   | Math
Dareaa  | Physics
Rahim   | English
Daniyal | Math
Daniyal | English
Junaid  | Physics
```

- **Key point:** Adding new students or courses requires updating `StudentCourses` to reflect relationships, otherwise JOIN query won’t show them.

---

5. Summary Table of Relationships

Relationship	Example	FK Usage / Notes
One-to-One	Users ↔ UserProfiles	UNIQUE + FK ensures 1:1
One-to-Many	Departments ↔ Employees	FK links parent to multiple children
Many-to-Many	Students ↔ Courses	Junction table + FK, multiple ↔ multiple

---

6. Advanced Tips

1. **Always enable FK constraints:** `PRAGMA foreign_keys = ON;`
2. **Use AUTOINCREMENT for PKs** to automatically generate IDs.
3. **Use INSERT OR IGNORE** to handle invalid FK inserts gracefully.
4. **Junction tables** are essential for N:N relationships.
5. **JOIN queries only display linked data**, ensure junction table is updated for new relationships.
6. **Data integrity** is maintained automatically by FK constraints.

This document now serves as a **full advanced-level reference** for SQLite3 Foreign Keys and all types of relationships, including commands, examples, and best practices.