In [ ]:
```
import sqlite3

conn = sqlite3.connect('test1.db')
```

In [2]:
```
def clenup(conn, emp, dept):
    cursor = conn.cursor()
    cursor.execute('PRAGMA foreign_keys = OFF;')
    cursor.execute('BEGIN DEFERRED TRANSACTION;')
    cursor.execute('DROP TABLE IF EXISTS {};'.format(emp))
    cursor.execute('DROP TABLE IF EXISTS {};'.format(dept))
    conn.commit()
    cursor.close()
```

In [3]:
```
def displayDB(conn):
    import pandas as pd
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' ORDER BY name;
    table_names = cursor.fetchall()

    # Check if there are any tables
    if not table_names:
        print("No tables found in the database.")
        return

    for table in table_names:
        df = pd.read_sql_query(f"SELECT * FROM {table[0]}", con=conn)
        print(f"Table: {table[0]}")
        print(df)
        print("\n")
    cursor.close()
```

In [4]:
```
def handle_error(conn, callback, *args, **kwargs):
    try:
        cursor = conn.cursor()
        cursor.execute('PRAGMA foreign_keys = ON;')
        callback(cursor, *args, **kwargs)
        cursor.connection.commit()
    except sqlite3.DatabaseError as e:
        error_message = str(e)
        print(f"Error: {error_message}")
        cursor.connection.rollback()
    except ValueError as e:
        error_message = str(e)
        print(f"Value Error: {error_message}")
    else:
        print("Operation completed successfully.")
    finally:
        cursor.close()
```

In [5]:
```
# define basic RUD operations
def insert_emp(cursor, eno, ename, birtyday, level, position, salary, dno):
    cursor.execute('''
        INSERT INTO Emp (eno, ename, birtyday, level, position, salary, dno)
        VALUES (?, ?, ?, ?, ?, ?, ?)
    ''', (eno, ename, birtyday, level, position, salary, dno))
    print("Record inserted successfully.")

def insert_dept(cursor, dno, dname, budget, manager):
    cursor.execute('''
        INSERT INTO Dept (dno, dname, budget, manager)
        VALUES (?, ?, ?, ?)
```

```
    ''', (dno, dname, budget, manager))
    print("Record inserted successfully.")

def update_emp(cursor, eno, ename=None, birtyday=None, level=None, position=None,
    cursor.execute("SELECT COUNT(*) FROM Emp WHERE eno = ?", (eno,))
    if cursor.fetchone()[0] == 0:
        raise ValueError(f"Employee with eno {eno} not found.")

    update_columns = []
    update_values = []

    if ename is not None:
        update_columns.append("ename = ?")
        update_values.append(ename)
    if birtyday is not None:
        update_columns.append("birtyday = ?")
        update_values.append(birtyday)
    if level is not None:
        update_columns.append("level = ?")
        update_values.append(level)
    if position is not None:
        update_columns.append("position = ?")
        update_values.append(position)
    if salary is not None:
        update_columns.append("salary = ?")
        update_values.append(salary)
    if dno is not None:
        update_columns.append("dno = ?")
        update_values.append(dno)

    if update_columns:
        update_values.append(eno)
        update_query = f'''
            UPDATE Emp
            SET {", ".join(update_columns)}
            WHERE eno = ?
        '''
        cursor.execute(update_query, tuple(update_values))
        print("Record updated successfully.")
    else:
        print("No values provided for update.")

def update_dept(cursor, dno, dname=None, budget=None, manager=None):
    cursor.execute("SELECT COUNT(*) FROM Dept WHERE dno = ?", (dno,))
    if cursor.fetchone()[0] == 0:
        raise ValueError(f"Department with dno {dno} not found.")

    update_columns = []
    update_values = []

    if dname is not None:
        update_columns.append("dname = ?")
        update_values.append(dname)
    if budget is not None:
        update_columns.append("budget = ?")
        update_values.append(budget)
    if manager is not None:
        update_columns.append("manager = ?")
        update_values.append(manager)

    if update_columns:
        update_values.append(dno)
        update_query = f'''
            UPDATE Dept
```

```
            SET {", ".join(update_columns)}
            WHERE dno = ?
        '''
        cursor.execute(update_query, tuple(update_values))
        print("Record updated successfully.")
    else:
        print("No values provided for update.")

def delete_emp(cursor, eno, emp="Emp"):
    cursor.execute(f"SELECT COUNT(*) FROM {emp} WHERE eno = ?", (eno,))
    if cursor.fetchone()[0] == 0:
        raise ValueError(f"Employee with eno {eno} not found.")

    cursor.execute(f'''
        DELETE FROM {emp}
        WHERE eno = ?
    ''', (eno,))
    print("Record deleted successfully.")


def delete_dept(cursor, dno, dept="Dept"):
    cursor.execute(f"SELECT COUNT(*) FROM {dept} WHERE dno = ?", (dno,))
    if cursor.fetchone()[0] == 0:
        raise ValueError(f"Department with dno {dno} not found.")

    cursor.execute(f'''
        DELETE FROM {dept}
        WHERE dno = ?
    ''', (dno,))
    print("Record deleted successfully.")
```

In [6]:

```
def insert_test_data(conn, dept_table='Dept', emp_table='Emp'):
    print("Inserting test data into %s and %s tables..." % (dept_table, emp_table))
    cursor = conn.cursor()
    try:
        cursor.execute('PRAGMA foreign_keys = ON;')
        cursor.execute('BEGIN DEFERRED TRANSACTION;')

        # Insert into Dept table
        cursor.execute(f'''
            INSERT INTO {dept_table} (dno, dname, budget, manager)
            VALUES (1001, '计算机学院', 28000.00, 1004)
        ''')
        cursor.execute(f'''
            INSERT INTO {dept_table} (dno, dname, budget, manager)
            VALUES (1002, '数学学院', 30000.00, 1005)
        ''')
        cursor.execute(f'''
            INSERT INTO {dept_table} (dno, dname, budget, manager)
            VALUES (1003, '智能学院', 4000.00, 1003)
        ''')

        # Insert into Emp table
        cursor.execute(f'''
            INSERT INTO {emp_table} (eno, ename, birtyday, level, position, salary, 
            VALUES (1001, '张伟', '1980-05-15', 3, '教师', 12000.00, 1001)
        ''')
        cursor.execute(f'''
            INSERT INTO {emp_table} (eno, ename, birtyday, level, position, salary, 
            VALUES (1002, '李娜', '1990-03-20', 2, '秘书', 8000.00, 1002)
        ''')
        cursor.execute(f'''
            INSERT INTO {emp_table} (eno, ename, birtyday, level, position, salary, 
            VALUES (1003, '王强', '1985-11-10', 1, '会计', 4000.00, 1003)
```

```
        ''')
        cursor.execute(f'''
            INSERT INTO {emp_table} (eno, ename, birtyday, level, position, salary, 
            VALUES (1004, '赵婷', '1992-08-05', 4, '教务', 16000.00, 1001)
        ''')
        cursor.execute(f'''
            INSERT INTO {emp_table} (eno, ename, birtyday, level, position, salary, 
            VALUES (1005, '孙明', '1988-12-25', 5, '教师', 30000.00, 1002)
        ''')

        conn.commit()
        print("Database created and data inserted successfully.")
    except sqlite3.DatabaseError as e:
        print(f"Error during inserting test data: {e}")
        conn.rollback()
    finally:
        cursor.close()
```

基本约束设计的定义部分

Emp(eno, ename, birtyday, level, position, salary, dno)

Dept(dno, dname, budget, manager)

1. 声明eno和dno是递增序列号形式的主码，长度为4的整型，形式为0001,0002，...
2. 声明Emp中的dno为参照Dept的外码， Dept的manager为参照Emp的外码
3. 测试外码定义的三种形式(最后再测试)
4. 限定dname为枚举型（数学学院、计算机学院、智能学院、电子学院、元培学院）
5. 限定position为枚举型（教师、教务、会计、秘书）
6. 限定level为1到5，缺省为3，salary为2000~200000

In [7]:
```
def create_table(conn):
    # default using no action
    clenup(conn, 'Emp', 'Dept')
    cursor = conn.cursor()
    cursor.execute('PRAGMA foreign_keys = ON;')

    # 创建 Dept 表
    cursor.execute('''
    CREATE TABLE Dept (
        dno INTEGER,
        dname TEXT NOT NULL,
        budget REAL,
        manager INTEGER,
        PRIMARY KEY (dno),
        CONSTRAINT fk_manager FOREIGN KEY (manager) REFERENCES Emp(eno) DEFERRABLE IN
        CONSTRAINT chk_dno_length CHECK (length(dno) = 4),
        CONSTRAINT chk_dname_valid CHECK (dname IN ('数学学院', '计算机学院', '智能学
    );
    ''')

    # 创建 Emp 表
    cursor.execute('''
    CREATE TABLE Emp (
        eno INTEGER,
        ename TEXT NOT NULL,
        birtyday DATE,
        level INTEGER DEFAULT 3,
        position TEXT NOT NULL,
        salary REAL,
```

```
        dno INTEGER,
        PRIMARY KEY (eno),
        CONSTRAINT fk_dno FOREIGN KEY (dno) REFERENCES Dept(dno) DEFERRABLE INITIALL
        CONSTRAINT chk_eno_length CHECK (length(eno) = 4),
        CONSTRAINT chk_level_range CHECK (level BETWEEN 1 AND 5),
        CONSTRAINT chk_position_valid CHECK (position IN ('教师', '教务', '会计', '秘
        CONSTRAINT chk_salary_range CHECK (salary BETWEEN 2000 AND 200000),
        CONSTRAINT chk_salary_level_match CHECK (
            (level = 1 AND salary BETWEEN 2000 AND 4999) OR
            (level = 2 AND salary BETWEEN 5000 AND 9999) OR
            (level = 3 AND salary BETWEEN 10000 AND 14999) OR
            (level = 4 AND salary BETWEEN 15000 AND 19999) OR
            (level = 5 AND salary BETWEEN 20000 AND 200000)
        )
    );
    ''')
    cursor.close()
```

基本约束设计的增删改操作（除去外码删除部分）

In [8]:
```
create_table(conn)
insert_test_data(conn)
displayDB(conn)
```

```
Inserting test data into Dept and Emp tables...
Database created and data inserted successfully.
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Dept_Restrict
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002


Table: Emp_Restrict
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [9]:
```
# a successful insert into Emp table
handle_error(conn, insert_emp, 1006, '王芳', '1995-01-01', 3, '教师', 13000.00, 1001)
```

```
print("After successful insert:")
displayDB(conn)
```

```
Record inserted successfully.
Operation completed successfully.
After successful insert:
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Dept_Restrict
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
5  1006    王芳  1995-01-01      3       教师  13000.0  1001


Table: Emp_Restrict
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [10]:
```
# a failed insert into Emp table (due to chk_eno_length constraint)
handle_error(conn, insert_emp, 100, '王芳', '1995-01-01', 2, '教师', 13000.00, 1001)
```

```
Error: CHECK constraint failed: chk_eno_length
```

In [11]:
```
# a failed insert into Emp table (due to foreign key constraint)
handle_error(conn, insert_emp, 1007, '李四', '1993-02-02', 3, '教师', 13000.00, 9999)
```

```
Record inserted successfully.
Error: FOREIGN KEY constraint failed
```

In [12]:
```
# a failed insert into Emp table (due to chk_level_range constraint)
handle_error(conn, insert_emp, 1007, '王芳', '1995-01-01', 6, '教师', 13000.00, 1001)
```

```
Error: CHECK constraint failed: chk_level_range
```

In [13]:
```
# a failed insert into Emp table (due to chk_position_valid constraint)
handle_error(conn, insert_emp, 1008, '王芳', '1995-01-01', 2, '经理', 13000.00, 1001)
```

```
Error: CHECK constraint failed: chk_position_valid
```

In [14]:
```
# a failed insert into Emp table (due to chk_salary_range constraint)
handle_error(conn, insert_emp, 1009, '王芳', '1995-01-01', 2, '教师', 300000.00, 100
```

```
Error: CHECK constraint failed: chk_salary_range
```

In [15]:
```
# a successful insert into Dept table
handle_error(conn, insert_dept, 1004, '电子学院', 400000.00, 1004)
print("After successful insert:")
displayDB(conn)
```

```
Record inserted successfully.
Operation completed successfully.
After successful insert:
Table: Dept
    dno  dname    budget  manager
0  1001  计算机学院   28000.0     1004
1  1002   数学学院   30000.0     1005
2  1003   智能学院    4000.0     1003
3  1004   电子学院  400000.0     1004


Table: Dept_Restrict
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
5  1006    王芳  1995-01-01      3       教师  13000.0  1001


Table: Emp_Restrict
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [16]:
```
# a failed insert into Dept table (due to chk_dno_length constraint)
handle_error(conn, insert_dept, 100, '电子学院', 400000.00, 1006)
```

```
Error: CHECK constraint failed: chk_dno_length
```

In [17]:
```
# a failed insert into Dept table (due to foreign key constraint)
handle_error(conn, insert_dept, 1005, '电子学院', 400000.00, 9999)
```

```
Record inserted successfully.
Error: FOREIGN KEY constraint failed
```

In [18]:
```
# a failed insert into Dept table (due to chk_dname_valid constraint)
handle_error(conn, insert_dept, 1005, '物理学院', 400000.00, 1006)
```

```
Error: CHECK constraint failed: chk_dname_valid
```

In [19]:
```
# a failed insert into Dept table (due to primary key constraint)
handle_error(conn, insert_dept, 1001, '电子学院', 400000.00, 1006)
```

```
Error: UNIQUE constraint failed: Dept.dno
```

In [20]:
```
# a successful update on Emp table
handle_error(conn, update_emp, eno=1006, ename='张三', salary=14000.00)
print("After successful update:")
displayDB(conn)
```

```
Record updated successfully.
Operation completed successfully.
After successful update:
Table: Dept
    dno  dname    budget  manager
0  1001  计算机学院   28000.0     1004
1  1002   数学学院   30000.0     1005
2  1003   智能学院    4000.0     1003
3  1004   电子学院  400000.0     1004


Table: Dept_Restrict
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
5  1006    张三  1995-01-01      3       教师  14000.0  1001


Table: Emp_Restrict
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [21]:
```
# a failed update on Emp table (due to primary key constraint)
handle_error(conn, update_emp, eno=1010, ename='李四', dno=1001)
```

```
Value Error: Employee with eno 1010 not found.
```

In [22]:
```
# a failed update on Emp table (due to foreign key constraint)
handle_error(conn, update_emp, eno=1006, dno=9999)
```

```
Record updated successfully.
Error: FOREIGN KEY constraint failed
```

In [23]:
```
# a failed update on Emp table (due to chk_level_range constraint)
handle_error(conn, update_emp, eno=1006, level=10)
```

```
Error: CHECK constraint failed: chk_level_range
```

In [24]:
```
# a failed update on Emp table (due to chk_position_valid constraint)
handle_error(conn, update_emp, eno=1006, position='总监')
```

```
Error: CHECK constraint failed: chk_position_valid
```

In [25]:
```
# a failed update on Emp table (due to chk_salary_range constraint)
handle_error(conn, update_emp, eno=1006, salary=300000.00)
```

```
Error: CHECK constraint failed: chk_salary_range
```

In [26]:
```
# a successful update on Dept table
handle_error(conn, update_dept, dno=1004, budget=600000.00)
print("After successful update:")
displayDB(conn)
```

```
Record updated successfully.
Operation completed successfully.
After successful update:
Table: Dept
    dno  dname    budget  manager
0  1001  计算机学院   28000.0     1004
1  1002   数学学院   30000.0     1005
2  1003   智能学院    4000.0     1003
3  1004   电子学院  600000.0     1004


Table: Dept_Restrict
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
5  1006    张三  1995-01-01      3       教师  14000.0  1001


Table: Emp_Restrict
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [27]:
```
# a failed update on Dept table (due to primary key constraint)
handle_error(conn, update_dept, dno=10010, budget=700000.00)
```

```
Value Error: Department with dno 10010 not found.
```

In [28]:
```
# a failed update on Dept table (due to foreign key constraint)
handle_error(conn, update_dept, dno=1004, manager=9999)
```

```
Record updated successfully.
Error: FOREIGN KEY constraint failed
```

In [29]:
```
# a failed update on Dept table (due to chk_dname_valid constraint)
handle_error(conn, update_dept, dno=1004, dname='物理学院', budget=700000.00)
```

```
Error: CHECK constraint failed: chk_dname_valid
```

In [30]:
```
# a successful delete on Emp table
handle_error(conn, delete_emp, eno=1006)
print("After successful delete:")
displayDB(conn)
```

```
Record deleted successfully.
Operation completed successfully.
After successful delete:
Table: Dept
    dno  dname    budget  manager
0  1001  计算机学院   28000.0     1004
1  1002   数学学院   30000.0     1005
2  1003   智能学院    4000.0     1003
3  1004   电子学院  600000.0     1004


Table: Dept_Restrict
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002


Table: Emp_Restrict
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [31]:
```
# a successful delete on Dept table
handle_error(conn, delete_dept, dno=1004)
print("After successful delete:")
displayDB(conn)
```

```
Record deleted successfully.
Operation completed successfully.
After successful delete:
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Dept_Restrict
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002


Table: Emp_Restrict
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [32]:
```
# a failed delete on Emp table
handle_error(conn, delete_emp, eno=1006)
```

```
Value Error: Employee with eno 1006 not found.
```

In [33]:
```
clenup(conn, 'Emp', 'Dept')
```

基本约束关系（外码删除相关）

In [34]:
```
clenup(conn, 'Emp_Restrict', 'Dept_Restrict')

cursor = conn.cursor()
cursor.execute('PRAGMA foreign_keys = ON;')

# 重新创建 Dept & Emp 表，使用 RESTRICT 约束
cursor.executescript('''
-- 使用 RESTRICT 的 Dept_Restrict 表
CREATE TABLE Dept_Restrict (
    dno INTEGER,
    dname TEXT NOT NULL,
    budget REAL,
    manager INTEGER,
    PRIMARY KEY (dno),
    FOREIGN KEY (manager) REFERENCES Emp_Restrict(eno) ON DELETE RESTRICT ON UPDATE
    CHECK (length(dno) = 4),
    CHECK (dname IN ('数学学院', '计算机学院', '智能学院', '电子学院', '元培学院'))
```

```
);

-- 使用 RESTRICT 的 Emp_Restrict 表
CREATE TABLE Emp_Restrict (
    eno INTEGER,
    ename TEXT NOT NULL,
    birtyday DATE,
    level INTEGER DEFAULT 3,
    position TEXT NOT NULL,
    salary REAL,
    dno INTEGER,
    PRIMARY KEY (eno),
    FOREIGN KEY (dno) REFERENCES Dept_Restrict(dno) ON DELETE RESTRICT ON UPDATE RES
    CHECK (length(eno) = 4),
    CHECK (level BETWEEN 1 AND 5),
    CHECK (position IN ('教师', '教务', '会计', '秘书')),
    CHECK (salary BETWEEN 2000 AND 200000)
);
''')
cursor.close()

insert_test_data(conn, 'Dept_Restrict', 'Emp_Restrict')
displayDB(conn)
```

```
Inserting test data into Dept_Restrict and Emp_Restrict tables...
Database created and data inserted successfully.
Table: Dept_Restrict
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp_Restrict
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [35]:

```
# an example of RESTRICT constraint
handle_error(conn, delete_emp, eno=1004, emp='Emp_Restrict')
print("After failed delete due to RESTRICT constraint:")
displayDB(conn)
```

```
Error: FOREIGN KEY constraint failed
After failed delete due to RESTRICT constraint:
Table: Dept_Restrict
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp_Restrict
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [36]:
```
clenup(conn, 'Emp_Restrict', 'Dept_Restrict')
```

In [37]:
```
clenup(conn, 'Emp_Cascade', 'Dept_Cascade')

cursor = conn.cursor()
cursor.execute('PRAGMA foreign_keys = ON;')

# 重新创建 Dept & Emp 表，使用 CASCADE 约束
cursor.executescript('''
-- 使用 CASCADE 的 Dept_Cascade 表
CREATE TABLE Dept_Cascade (
    dno INTEGER,
    dname TEXT NOT NULL,
    budget REAL,
    manager INTEGER,
    PRIMARY KEY (dno),
    FOREIGN KEY (manager) REFERENCES Emp_Cascade(eno) ON DELETE CASCADE ON UPDATE CA
    CHECK (length(dno) = 4),
    CHECK (dname IN ('数学学院', '计算机学院', '智能学院', '电子学院', '元培学院'))
);

-- 使用 CASCADE 的 Emp_Cascade 表
CREATE TABLE Emp_Cascade (
    eno INTEGER,
    ename TEXT NOT NULL,
    birtyday DATE,
    level INTEGER DEFAULT 3,
    position TEXT NOT NULL,
    salary REAL,
    dno INTEGER,
    PRIMARY KEY (eno),
    FOREIGN KEY (dno) REFERENCES Dept_Cascade(dno) ON DELETE CASCADE ON UPDATE CASCAI
    CHECK (length(eno) = 4),
    CHECK (level BETWEEN 1 AND 5),
    CHECK (position IN ('教师', '教务', '会计', '秘书')),
    CHECK (salary BETWEEN 2000 AND 200000)
);
''')
cursor.close()

insert_test_data(conn, 'Dept_Cascade', 'Emp_Cascade')
displayDB(conn)
```

```
Inserting test data into Dept_Cascade and Emp_Cascade tables...
Database created and data inserted successfully.
Table: Dept_Cascade
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp_Cascade
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [38]:
```
# an example of CASCADE constraint
handle_error(conn, delete_emp, eno=1004, emp='Emp_Cascade')
print("After successful delete due to CASCADE constraint:")
displayDB(conn)
```

```
Record deleted successfully.
Operation completed successfully.
After successful delete due to CASCADE constraint:
Table: Dept_Cascade
    dno dname   budget  manager
0  1002  数学学院  30000.0     1005
1  1003  智能学院   4000.0     1003


Table: Emp_Cascade
    eno ename    birtyday  level position   salary   dno
0  1002    李娜  1990-03-20      2       秘书   8000.0  1002
1  1003    王强  1985-11-10      1       会计   4000.0  1003
2  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [39]:
```
clenup(conn, 'Emp_Cascade', 'Dept_Cascade')
```

In [40]:
```
clenup(conn, 'Emp_SetNull', 'Dept_SetNull')

cursor = conn.cursor()
cursor.execute('PRAGMA foreign_keys = ON;')

# 重新创建 Dept & Emp 表，使用 SET NULL 约束
cursor.executescript('''
-- 使用 SET NULL 的 Dept_SetNull 表
CREATE TABLE Dept_SetNull (
    dno INTEGER,
    dname TEXT NOT NULL,
    budget REAL,
    manager INTEGER,
    PRIMARY KEY (dno),
    FOREIGN KEY (manager) REFERENCES Emp_SetNull(eno) ON DELETE SET NULL ON UPDATE SE
    CHECK (length(dno) = 4),
    CHECK (dname IN ('数学学院', '计算机学院', '智能学院', '电子学院', '元培学院'))
);

-- 使用 SET NULL 的 Emp_SetNull 表
CREATE TABLE Emp_SetNull (
```

```
    eno INTEGER,
    ename TEXT NOT NULL,
    birtyday DATE,
    level INTEGER DEFAULT 3,
    position TEXT NOT NULL,
    salary REAL,
    dno INTEGER,
    PRIMARY KEY (eno),
    FOREIGN KEY (dno) REFERENCES Dept_SetNull(dno) ON DELETE SET NULL ON UPDATE SET N
    CHECK (length(eno) = 4),
    CHECK (level BETWEEN 1 AND 5),
    CHECK (position IN ('教师', '教务', '会计', '秘书')),
    CHECK (salary BETWEEN 2000 AND 200000)
);
''')
cursor.close()

insert_test_data(conn, 'Dept_SetNull', 'Emp_SetNull')
displayDB(conn)
```

```
Inserting test data into Dept_SetNull and Emp_SetNull tables...
Database created and data inserted successfully.
Table: Dept_SetNull
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp_SetNull
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [41]:

```
# an example of SET NULL constraint
handle_error(conn, delete_emp, eno=1004, emp='Emp_SetNull')
print("After successful delete due to SET NULL constraint:")
displayDB(conn)
```

```
Record deleted successfully.
Operation completed successfully.
After successful delete due to SET NULL constraint:
Table: Dept_SetNull
    dno  dname   budget  manager
0  1001  计算机学院  28000.0      NaN
1  1002   数学学院  30000.0   1005.0
2  1003   智能学院   4000.0   1003.0


Table: Emp_SetNull
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [42]:

```
clenup(conn, 'Emp_SetNull', 'Dept_SetNull')
```

中级约束设计 延迟约束

DEFERRABLE INITIALLY DEFERRED

In [43]:
```
create_table(conn)
insert_test_data(conn)
displayDB(conn)
```

```
Inserting test data into Dept and Emp tables...
Database created and data inserted successfully.
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  30000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [44]:
```
# a failed insert into Dept and Emp tables
def imme_insert(cursor):
    cursor.executescript('''
        INSERT INTO Dept (dno, dname, budget, manager)
        VALUES (1004, '电子学院', 500000.00, 1009);

        INSERT INTO Emp (eno, ename, birtyday, level, position, salary, dno)
        VALUES (1009, '李四', '1993-02-02', 2, '教师', 13000.00, 1004);
    ''')
    print("Record inserted successfully.")

handle_error(conn, imme_insert)
```

```
Error: FOREIGN KEY constraint failed
```

In [45]:
```
def delayed_insert(cursor):
    cursor.execute('BEGIN DEFERRED TRANSACTION;')
    cursor.execute('''
        INSERT INTO Dept (dno, dname, budget, manager)
        VALUES (1004, '电子学院', 500000.00, 1009)
    ''')
    cursor.execute('''
        INSERT INTO Emp (eno, ename, birtyday, level, position, salary, dno)
        VALUES (1009, '李四', '1993-02-02', 3, '教师', 13000.00, 1004)
    ''')
    print("Record inserted successfully.")

handle_error(conn, delayed_insert)
displayDB(conn)
```

```
Record inserted successfully.
Operation completed successfully.
Table: Dept
    dno  dname    budget  manager
0  1001  计算机学院   28000.0     1004
1  1002   数学学院   30000.0     1005
2  1003   智能学院    4000.0     1003
3  1004   电子学院  500000.0     1009


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
5  1009    李四  1993-02-02      3       教师  13000.0  1004
```

中级约束设计 工资约束

CONSTRAINT chk_salary_level_match CHECK (
(level = 1 AND salary BETWEEN 2000 AND 4999) OR
(level = 2 AND salary BETWEEN 5000 AND 9999) OR
(level = 3 AND salary BETWEEN 10000 AND 14999) OR
(level = 4 AND salary BETWEEN 15000 AND 19999) OR
(level = 5 AND salary BETWEEN 20000 AND 200000)
)

In [46]:
```
# a successful update on salary and level columns in Emp table
handle_error(conn, update_emp, eno=1009, salary=25000.00, level=5)
print("After successful update:")
displayDB(conn)
```

```
Record updated successfully.
Operation completed successfully.
After successful update:
Table: Dept
    dno  dname    budget  manager
0  1001  计算机学院   28000.0     1004
1  1002   数学学院   30000.0     1005
2  1003   智能学院    4000.0     1003
3  1004   电子学院  500000.0     1009


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
5  1009    李四  1993-02-02      5       教师  25000.0  1004
```

In [47]:
```
# a failed update on Emp table (due to salary level mismatch)
handle_error(conn, update_emp, eno=1009, salary=25000.00, level=2)
```

```
Error: CHECK constraint failed: chk_salary_level_match
```

中级约束设计 员工编码

sqlite 暂不支持sql: create function, 仅仅采用python函数
下提供一种sql实现：

```
DELIMITER $$

CREATE FUNCTION generateSmartCode(p_eno INT) RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
    DECLARE v_eno CHAR(4);
    DECLARE v_dno CHAR(4);
    DECLARE v_birth_year CHAR(4);
    DECLARE v_level CHAR(2);
    DECLARE v_position_code CHAR(2);
    DECLARE v_manager CHAR(4);

    -- 查询
    SELECT
        LPAD(E.eno, 4, '0'),
        LPAD(E.dno, 4, '0'),
        YEAR(E.birtyday),
        LPAD(E.level, 2, '0'),
        IFNULL((SELECT code FROM PositionCode WHERE position =
E.position), '00'),
        LPAD(D.manager, 4, '0')
    INTO v_eno, v_dno, v_birth_year, v_level, v_position_code, v_manager
    FROM Emp E JOIN Dept D ON E.dno = D.dno
    WHERE E.eno = p_eno;

    -- 拼接
    RETURN CONCAT(v_eno, v_dno, v_birth_year, v_level, v_position_code,
v_manager);
END$$

DELIMITER ;
```

In [48]:

```
# add new tables for position and department codes
clenup(conn, 'PositionCode', 'DeptCode')
cursor = conn.cursor()
cursor.executescript('''
    -- 职位编码表
    CREATE TABLE PositionCode (
        position TEXT PRIMARY KEY,
        code TEXT NOT NULL
    );

    -- 院系编码表
    CREATE TABLE DeptCode (
        dname TEXT PRIMARY KEY,
        code TEXT NOT NULL
    );
''')

cursor.executescript('''
    INSERT INTO PositionCode (position, code) VALUES
```

```
    ('教师', '01'),
    ('教务', '02'),
    ('会计', '03'),
    ('秘书', '04');

    INSERT INTO DeptCode (dname, code) VALUES
    ('数学学院', '01'),
    ('计算机学院', '02'),
    ('智能学院', '03'),
    ('电子学院', '04'),
    ('元培学院', '05');
''')

conn.commit()
cursor.close()
```

In [49]:
```
def generate_smart_code(conn, eno):
    cursor = conn.cursor()
    cursor.execute('PRAGMA foreign_keys = ON;')

    cursor.execute('''
        SELECT Emp.eno, Emp.dno, strftime('%Y', Emp.birtyday) AS birth_year,
               Emp.level, Emp.position,
               Dept.manager
        FROM Emp natural join Dept
        WHERE Emp.eno = ?
    ''', (eno,))
    row = cursor.fetchone()

    # 没找到该员工
    if not row:
        cursor.close()
        return None

    eno, dno, birth_year, level, position, manager = row

    # 查职位编码
    cursor.execute('SELECT code FROM PositionCode WHERE position = ?', (position,))
    pos_row = cursor.fetchone()
    position_code = pos_row[0] if pos_row else '00'

    # 最后拼接
    smart_code = f"{eno:04d}{dno:04d}{birth_year}{int(level):02d}{position_code}{man

    cursor.close()
    return smart_code

def print_smart_code(conn, eno):
    smart_code = generate_smart_code(conn, eno)
    if smart_code:
        print(f"Smart code for employee {eno}: {smart_code}")
    else:
        print(f"Employee {eno} not found.")
```

In [50]:
```
print_smart_code(conn, 1009)
print_smart_code(conn, 1010)
```

```
Smart code for employee 1009: 10091004199305011009
Employee 1010 not found.
```

In [51]:
```
clenup(conn, 'PositionCode', 'DeptCode')
```

高级约束设计

1. 使用触发器来保证管理者的工资必须高于他所管理的任何一个员工
2. 使用触发器保证任何一个员工工资的变化额度，都应该体现在他所在部门的预算上面，本质上这相当于实现了一个物化视图的一致性维护机制（触发器应该考虑到员工改变工作部门的情况，从一致性维护效率的角度，完全重算当然最简单，但希望还是实现基于更新行的增量更新）

In [52]:
```
clenup(conn, 'Emp', 'Dept')
cursor = conn.cursor()
cursor.execute('PRAGMA foreign_keys = ON;')

# 创建 Dept 表
cursor.execute('''
    CREATE TABLE Dept (
        dno INTEGER,
        dname TEXT NOT NULL,
        budget REAL,
        manager INTEGER,
        PRIMARY KEY (dno),
        CONSTRAINT fk_manager FOREIGN KEY (manager) REFERENCES Emp(eno) DEFERRABLE IN
        CONSTRAINT chk_dno_length CHECK (length(dno) = 4),
        CONSTRAINT chk_dname_valid CHECK (dname IN ('数学学院', '计算机学院', '智能学
    );
''')

# 创建 Emp 表
cursor.execute('''
    CREATE TABLE Emp (
        eno INTEGER,
        ename TEXT NOT NULL,
        birtyday DATE,
        level INTEGER DEFAULT 3,
        position TEXT NOT NULL,
        salary REAL,
        dno INTEGER,
        PRIMARY KEY (eno),
        CONSTRAINT fk_dno FOREIGN KEY (dno) REFERENCES Dept(dno) DEFERRABLE INITIALLY
        CONSTRAINT chk_eno_length CHECK (length(eno) = 4),
        CONSTRAINT chk_level_range CHECK (level BETWEEN 1 AND 5),
        CONSTRAINT chk_position_valid CHECK (position IN ('教师', '教务', '会计', '秘
        CONSTRAINT chk_salary_range CHECK (salary BETWEEN 2000 AND 200000),
        CONSTRAINT chk_salary_level_match CHECK (
            (level = 1 AND salary BETWEEN 2000 AND 4999) OR
            (level = 2 AND salary BETWEEN 5000 AND 9999) OR
            (level = 3 AND salary BETWEEN 10000 AND 14999) OR
            (level = 4 AND salary BETWEEN 15000 AND 19999) OR
            (level = 5 AND salary BETWEEN 20000 AND 200000)
        )
    );
''')

# 处理工资不高于经理的触发器
# 不考虑删除操作，因为删除时会依据外键约束删除
cursor.executescript('''
DROP TRIGGER IF EXISTS check_manager_salary_after_emp_insert;
DROP TRIGGER IF EXISTS check_manager_salary_after_emp_update;
DROP TRIGGER IF EXISTS check_manager_salary_after_dept_update;
DROP TRIGGER IF EXISTS check_manager_salary_after_dept_insert;
''')
```

```
# 插入新员工时触发
cursor.executescript('''
CREATE TRIGGER check_manager_salary_after_emp_insert
AFTER INSERT ON Emp
FOR EACH ROW
BEGIN
    SELECT CASE
        WHEN EXISTS (
            SELECT 1
            FROM Dept JOIN Emp AS M ON Dept.manager = M.eno
            WHERE NEW.eno != M.eno AND
                      Dept.dno = NEW.dno AND NEW.salary >= M.salary
        )
        THEN
            RAISE(FAIL, '员工工资不能高于或等于其管理工资')
    END;
END;
''')

# 更新员工时触发
cursor.executescript('''
CREATE TRIGGER check_manager_salary_after_emp_update
AFTER UPDATE ON Emp
FOR EACH ROW
BEGIN
    SELECT CASE
        WHEN EXISTS (
            SELECT 1
            FROM Dept JOIN Emp AS M ON Dept.manager = M.eno
            WHERE NEW.eno != M.eno AND
                      Dept.dno = NEW.dno AND NEW.salary >= M.salary
        )
        THEN
            RAISE(FAIL, '更新失败：员工工资不能高于或等于其管理工资')
    END;
END;
''')

# 更新管理时触发
cursor.executescript('''
CREATE TRIGGER check_manager_salary_after_dept_update
AFTER UPDATE OF manager ON Dept
FOR EACH ROW
BEGIN
    SELECT CASE
        WHEN EXISTS (
            SELECT 1
            FROM Emp
            WHERE dno = NEW.dno AND eno != NEW.manager AND
                      salary >= (SELECT salary FROM Emp WHERE eno = NEW.manager)
        )
        THEN
            RAISE(FAIL, '更新失败：经理工资必须高于所有员工工资')
    END;
END;
''')

# 插入新部门时触发(其实理应不需要，新部门第一个就是管理员)
cursor.executescript('''
CREATE TRIGGER check_manager_salary_after_dept_insert
AFTER INSERT ON Dept
FOR EACH ROW
BEGIN
```

```
    SELECT CASE
        WHEN EXISTS (
            SELECT 1
            FROM Emp
            WHERE dno = NEW.dno AND eno != NEW.manager AND
                      salary >= (SELECT salary FROM Emp WHERE eno = NEW.manager)
        )
        THEN
            RAISE(FAIL, '插入失败：经理工资必须高于所有员工工资')
    END;
END;
''')

# 处理预算统计触发器
cursor.executescript('''
DROP TRIGGER IF EXISTS budget_statistics_trigger_after_insert;
DROP TRIGGER IF EXISTS budget_statistics_trigger_after_update;
DROP TRIGGER IF EXISTS budget_statistics_trigger_after_delete;
DROP TRIGGER IF EXISTS budget_statistics_trigger_after_create;
''')

# 创建预算统计触发器
cursor.executescript('''
CREATE TRIGGER budget_statistics_trigger_after_create
AFTER INSERT ON Dept
FOR EACH ROW
BEGIN
    UPDATE Dept SET budget = 0 WHERE dno = NEW.dno;
END;
''')

# 插入触发器
cursor.executescript('''
CREATE TRIGGER budget_statistics_trigger_after_insert
AFTER INSERT ON Emp
FOR EACH ROW
BEGIN
    UPDATE Dept SET budget = budget + NEW.salary WHERE dno = NEW.dno;
END;
''')

# 删除触发器
cursor.executescript('''
CREATE TRIGGER budget_statistics_trigger_after_delete
AFTER DELETE ON Emp
FOR EACH ROW
BEGIN
    UPDATE Dept SET budget = budget - OLD.salary WHERE dno = OLD.dno;
END;
''')

# 更新触发器
cursor.executescript('''
CREATE TRIGGER budget_statistics_trigger_after_update
AFTER UPDATE OF salary, dno ON Emp
FOR EACH ROW
BEGIN
    -- 部门没变，只更新差值
    UPDATE Dept
    SET budget = budget + (NEW.salary - OLD.salary)
    WHERE dno = NEW.dno AND OLD.dno = NEW.dno;

    -- 部门变了，分别扣旧部门、加新部门
    UPDATE Dept
```

```
    SET budget = budget - OLD.salary
    WHERE dno = OLD.dno AND OLD.dno != NEW.dno;

    UPDATE Dept
    SET budget = budget + NEW.salary
    WHERE dno = NEW.dno AND OLD.dno != NEW.dno;
END;
''')

conn.commit()
cursor.close()
insert_test_data(conn)
displayDB(conn)
```

```
Inserting test data into Dept and Emp tables...
Database created and data inserted successfully.
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1004
1  1002   数学学院  38000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
```

In [53]:

```
# a successful insert into Emp table
handle_error(conn, insert_emp, 1006, '王芳', '1995-01-01', 3, '教师', 13000.00, 1001)
print("After successful insert:")
displayDB(conn)
```

```
Record inserted successfully.
Operation completed successfully.
After successful insert:
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  41000.0     1004
1  1002   数学学院  38000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
5  1006    王芳  1995-01-01      3       教师  13000.0  1001
```

In [54]:

```
# a failed insert into Emp table
handle_error(conn, insert_emp, 1007, '王芳', '1995-01-01', 5, '教师', 30000.00, 1001)
```

```
Error: 员工工资不能高于或等于其管理工资
```

In [55]:
```
# a successful update on Emp table (change salary to 15500.00)
handle_error(conn, update_emp, eno=1006, salary=15500.00, level=4)
print("After successful update:")
displayDB(conn)
```

```
Record updated successfully.
Operation completed successfully.
After successful update:
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  43500.0     1004
1  1002   数学学院  38000.0     1005
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      4       教务  16000.0  1001
4  1005    孙明  1988-12-25      5       教师  30000.0  1002
5  1006    王芳  1995-01-01      4       教师  15500.0  1001
```

In [56]:
```
# a failed update on Emp table
handle_error(conn, update_emp, eno=1006, salary=20000.00, level=5)
```

```
Error: 更新失败：员工工资不能高于或等于其管理工资
```

In [57]:
```
# a failed update on Dept table (due to manager salary constraint)
handle_error(conn, update_dept, dno=1001, manager=1006)
```

```
Error: 更新失败：经理工资必须高于所有员工工资
```

In [58]:
```
def swap_dept_manager(cursor, dno1, dno2):
    # 查出两个部门的当前经理
    cursor.execute('SELECT manager FROM Dept WHERE dno = ?', (dno1,))
    row1 = cursor.fetchone()
    if row1 is None:
        raise ValueError(f"Department {dno1} not found.")
    manager1 = row1[0]

    cursor.execute('SELECT manager FROM Dept WHERE dno = ?', (dno2,))
    row2 = cursor.fetchone()
    if row2 is None:
        raise ValueError(f"Department {dno2} not found.")
    manager2 = row2[0]

    # 查出两个经理的工资和级别
    cursor.execute('SELECT salary, level FROM Emp WHERE eno = ?', (manager1,))
    m1 = cursor.fetchone()
    if m1 is None:
        raise ValueError(f"Manager {manager1} not found in Emp table.")
    salary1, level1 = m1

    cursor.execute('SELECT salary, level FROM Emp WHERE eno = ?', (manager2,))
    m2 = cursor.fetchone()
    if m2 is None:
        raise ValueError(f"Manager {manager2} not found in Emp table.")
    salary2, level2 = m2

    # 保证工资较低的经理在后
```

```
    if salary1 <= salary2:
        tmp_salary = salary1
        tmp_level = level1
        tmp_dno = dno1
        tmp_manager = manager1
        dno1 = dno2
        salary1 = salary2
        level1 = level2
        manager1 = manager2
        dno2 = tmp_dno
        salary2 = tmp_salary
        level2 = tmp_level
        manager2 = tmp_manager

    print(f"Swapping managers: {manager1} (salary: {salary1}, level: {level1}) from
          f"with {manager2} (salary: {salary2}, level: {level2}) from {dno2}")

    # 开启事务
    cursor.execute('BEGIN DEFERRED TRANSACTION;')

    # 暂时提升后面部门的经理工资
    cursor.execute('UPDATE Emp SET salary = ?, level = ? WHERE eno = ?',
                   (salary1 + 0.01, level1, manager2))
    # 交换部门的manager
    cursor.execute('UPDATE Dept SET manager = ? WHERE dno = ?', (manager2, dno1))
    cursor.execute('UPDATE Emp SET dno = ? WHERE eno = ?', (dno1, manager2))

    cursor.execute('UPDATE Dept SET manager = ? WHERE dno = ?', (manager1, dno2))
    cursor.execute('UPDATE Emp SET salary = ?, level = ?, dno = ? WHERE eno = ?'
                   , (salary2, level2, dno2, manager1))

    cursor.execute('UPDATE Emp SET salary = ? WHERE eno = ?', (salary1, manager2))

    print(f"Managers of department {dno1} and {dno2} swapped successfully.")

handle_error(conn, swap_dept_manager, 1001, 1002)
print("After swapping managers:")
displayDB(conn)
```

```
Swapping managers: 1005 (salary: 30000.0, level: 5) from 1002 with 1004 (salary: 160
00.0, level: 4) from 1001
Managers of department 1002 and 1001 swapped successfully.
Operation completed successfully.
After swapping managers:
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  43500.0     1005
1  1002   数学学院  38000.0     1004
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      5       教务  30000.0  1002
4  1005    孙明  1988-12-25      4       教师  16000.0  1001
5  1006    王芳  1995-01-01      4       教师  15500.0  1001
```

In [59]:

```
# a successful update on Emp's department and salary
handle_error(conn, update_emp, eno=1006, dno=1002, salary=25000.00, level=5)
```

```
print("After successful update:")
displayDB(conn)
```

```
Record updated successfully.
Operation completed successfully.
After successful update:
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1005
1  1002   数学学院  63000.0     1004
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      5       教务  30000.0  1002
4  1005    孙明  1988-12-25      4       教师  16000.0  1001
5  1006    王芳  1995-01-01      5       教师  25000.0  1002
```

In [60]:
```
# a successful delete on Emp table
handle_error(conn, delete_emp, eno=1006)
print("After successful delete:")
displayDB(conn)
```

```
Record deleted successfully.
Operation completed successfully.
After successful delete:
Table: Dept
    dno  dname   budget  manager
0  1001  计算机学院  28000.0     1005
1  1002   数学学院  38000.0     1004
2  1003   智能学院   4000.0     1003


Table: Emp
    eno ename    birtyday  level position   salary   dno
0  1001    张伟  1980-05-15      3       教师  12000.0  1001
1  1002    李娜  1990-03-20      2       秘书   8000.0  1002
2  1003    王强  1985-11-10      1       会计   4000.0  1003
3  1004    赵婷  1992-08-05      5       教务  30000.0  1002
4  1005    孙明  1988-12-25      4       教师  16000.0  1001
```

In [61]:
```
conn.close()
```

终极约束设计

my_stock(stock_id, volume, avg_price, profit)： 表示所持有的股票编号、数量、持仓平均价格、利润

trans(trans_id,stock_id, date, price, amount, sell_or_buy)： 表示一次交易的编号、股票编号、交易日期、成交价格、成交数量、买入还是卖出

使用触发器完成下面的工作：

1. 往trans里面插入一条记录时，根据其是买入还是卖出，调整my_stock中的volume以及avg_price。如果是初次插入的股票交易，就在my_stock中为该股票新建一条记录，profit置为0。注意，如果一笔卖出交易的amount大于my_stock中该股票的volume，说明是无效的下单交易，应该加以拒绝，直接抛弃。
2. profit的计算方式如下：每当有卖出交易发生时，将其与尽可能远的买入交易进行匹配，比如如果trans中现有的记录为{(t01,s01,d01,10,1000,buy), (t02,s01,d02,12,500,buy)},如果现在插入{(t03,s01,d03,11,700,sold)}，本次交易产生的profit=(11-10)*700,如果再插入{(t04,s01,d04,9,700,sold)}，本次交易产生的profit=(9-10)*300 + (9-12)*400 = -1500.将每次卖出交易的profit都累加到my_stock的profit上。

由于sqlite不支持for & if，我也没有想到替代方案，第二题将符合MySQL的部分放在下方，替换现在触发器中的sell成功部分即可

```
-- Update the stock volume for a sell transaction
    -- Accumulate the profit from previous buy transactions
    DECLARE profit_accumulated REAL DEFAULT 0;
    DECLARE remaining_amount INT DEFAULT NEW.amount;

    -- Calculate the total amount already sold from previous sell
transactions
    DECLARE total_selled_amount INT DEFAULT (
        SELECT SUM(amount)
        FROM trans
        WHERE stock_id = NEW.stock_id AND sell_or_buy = 'S'
    );

    -- Loop through all previous buy transactions for this stock and
match with the sell
    FOR trans_row IN (
        SELECT trans_id, price, amount
        FROM trans
        WHERE stock_id = NEW.stock_id AND sell_or_buy = 'B'
        ORDER BY date ASC  -- Ensure buy transactions are processed in
the order they occurred
    ) DO
        -- Skip the buy transaction if the amount has already been fully
matched
        DECLARE available_buy_amount INT DEFAULT trans_row.amount -
total_selled_amount;
        IF available_buy_amount <= 0 THEN
            SET total_selled_amount = total_selled_amount -
trans_row.amount;
            CONTINUE;
        ELSE
            SET total_selled_amount = 0;
        END IF;

        -- Match as much as possible from the current buy transaction
        DECLARE amount_to_match INT DEFAULT LEAST(remaining_amount,
available_buy_amount);

        IF amount_to_match > 0 THEN
            -- Calculate profit for the matched amount
            DECLARE profit REAL DEFAULT (NEW.price - trans_row.price) *
amount_to_match;
```

```
            SET profit_accumulated = profit_accumulated + profit;
            SET remaining_amount = remaining_amount - amount_to_match;

            IF remaining_amount = 0 THEN
                BREAK;
            END IF;
        END IF;
    END FOR;


    UPDATE my_stock
    SET volume = volume - NEW.amount,
        profit = profit + profit_accumulated
    WHERE stock_id = NEW.stock_id AND NEW.sell_or_buy = 'S';
```

In [67]:

```
conn = sqlite3.connect('test2.db')

clenup(conn, 'my_stock', 'trans')
cursor = conn.cursor()
cursor.execute('PRAGMA foreign_keys = ON;')

cursor.executescript('''
CREATE TABLE my_stock (
    stock_id INTEGER PRIMARY KEY,  -- 股票编号
    volume INTEGER,                -- 股票数量
    avg_price REAL,                -- 持仓平均价格
    profit REAL                    -- 利润
);
''')

cursor.executescript('''
CREATE TABLE trans (
    trans_id INTEGER PRIMARY KEY,  -- 交易编号
    stock_id INTEGER,              -- 股票编号
    date INTEGER,                  -- 成交日期
    price REAL,                    -- 成交价格
    amount INTEGER,                -- 成交数量
    sell_or_buy TEXT,              -- 'B' 或 'S'
    FOREIGN KEY (stock_id) REFERENCES my_stock(stock_id),
    CONSTRAINT chk_sell_or_buy CHECK (sell_or_buy IN ('B', 'S'))
);
''')

cursor.execute('''
DROP TRIGGER IF EXISTS trans_insert_trigger;
''')

cursor.executescript('''
CREATE TRIGGER trans_insert_trigger
BEFORE INSERT ON trans
FOR EACH ROW
BEGIN
    -- For a buy transaction
    INSERT OR IGNORE INTO my_stock (stock_id, volume, avg_price, profit)
    VALUES (NEW.stock_id, 0, 0, 0);

    -- Update the stock volume and average price for a buy transaction
    UPDATE my_stock
    SET volume = volume + NEW.amount,
        avg_price = (avg_price * volume + NEW.amount * NEW.price) / (volume + NEW.am
    WHERE stock_id = NEW.stock_id AND NEW.sell_or_buy = 'B';
```

```
    -- For a sell transaction, check if enough volume exists
    -- If sell is attempted with more than available volume, reject the transaction
    SELECT CASE
               WHEN (SELECT volume FROM my_stock WHERE stock_id = NEW.stock_id) < NEW
               THEN RAISE(FAIL, 'Insufficient stock volume for sell transaction')
               ELSE NULL
           END;

    UPDATE my_stock
    SET volume = volume - NEW.amount
    WHERE stock_id = NEW.stock_id AND NEW.sell_or_buy = 'S';
END;
''')

conn.commit()
cursor.close()
```

In [74]:
```
def insert_trans(cursor, trans_id, stock_id, date, price, amount, sell_or_buy):
    cursor.execute('''
        INSERT INTO trans (trans_id, stock_id, date, price, amount, sell_or_buy)
        VALUES (?, ?, ?, ?, ?, ?)
    ''', (trans_id, stock_id, date, price, amount, sell_or_buy))
```

In [75]:
```
handle_error(conn, insert_trans, 1, 1, 1, 10, 1000, 'B')
displayDB(conn)
```

```
Operation completed successfully.
Table: my_stock
   stock_id  volume  avg_price  profit
0         1    1000       10.0     0.0


Table: trans
   trans_id  stock_id  date  price  amount sell_or_buy
0         1         1     1   10.0    1000           B
```

In [76]:
```
handle_error(conn, insert_trans, 2, 1, 2, 11, 500, 'B')
displayDB(conn)
```

```
Operation completed successfully.
Table: my_stock
   stock_id  volume  avg_price  profit
0         1    1500  10.333333     0.0


Table: trans
   trans_id  stock_id  date  price  amount sell_or_buy
0         1         1     1   10.0    1000           B
1         2         1     2   11.0     500           B
```

In [77]:
```
handle_error(conn, insert_trans, 3, 1, 3, 12, 800, 'S')
displayDB(conn)
```

```
Operation completed successfully.
Table: my_stock
   stock_id  volume  avg_price  profit
0         1     700  10.333333     0.0


Table: trans
   trans_id  stock_id  date  price  amount sell_or_buy
0         1         1     1   10.0    1000          B
1         2         1     2   11.0     500          B
2         3         1     3   12.0     800          S
```

In [78]:
```
handle_error(conn, insert_trans, 4, 1, 4, 12, 1000, 'S')
displayDB(conn)
```

```
Error: Insufficient stock volume for sell transaction
Table: my_stock
   stock_id  volume  avg_price  profit
0         1     700  10.333333     0.0


Table: trans
   trans_id  stock_id  date  price  amount sell_or_buy
0         1         1     1   10.0    1000          B
1         2         1     2   11.0     500          B
2         3         1     3   12.0     800          S
```

In [79]:
```
handle_error(conn, insert_trans, 5, 1, 5, 9, 1000, 'B')
displayDB(conn)
```

```
Operation completed successfully.
Table: my_stock
   stock_id  volume  avg_price  profit
0         1    1700    9.54902     0.0


Table: trans
   trans_id  stock_id  date  price  amount sell_or_buy
0         1         1     1   10.0    1000          B
1         2         1     2   11.0     500          B
2         3         1     3   12.0     800          S
3         5         1     5    9.0    1000          B
```

In [80]:
```
handle_error(conn, insert_trans, 6, 1, 6, 12, 800, 'S')
displayDB(conn)
```

```
Operation completed successfully.
Table: my_stock
   stock_id  volume  avg_price  profit
0         1     900    9.54902     0.0


Table: trans
   trans_id  stock_id  date  price  amount sell_or_buy
0         1         1     1   10.0    1000           B
1         2         1     2   11.0     500           B
2         3         1     3   12.0     800           S
3         5         1     5    9.0    1000           B
4         6         1     6   12.0     800           S
```

In [81]:
```
handle_error(conn, insert_trans, 7, 1, 7, 7, 800, 'S')
displayDB(conn)
```

```
Operation completed successfully.
Table: my_stock
   stock_id  volume  avg_price  profit
0         1     100    9.54902     0.0


Table: trans
   trans_id  stock_id  date  price  amount sell_or_buy
0         1         1     1   10.0    1000           B
1         2         1     2   11.0     500           B
2         3         1     3   12.0     800           S
3         5         1     5    9.0    1000           B
4         6         1     6   12.0     800           S
5         7         1     7    7.0     800           S
```