

# 实习三：非关系数据实习

吕杭州 2200013126      戴思颖 2200094811      戴傅聪 2100013061

2025 年 5 月 29 日

本次实习的目标是请同学们掌握一些典型的非关系数据类型在关系数据库中的处理手段。主要包括如下几方面的练习：

- 1. 递归查询
- 2. JSON 和关系表的导入导出和基本查询
- 3. 向量数据库的使用

## 1. 递归查询

下面是八王之乱的亲属关系示意图，我们先把它存入一个 family 表中，记录图中直接的父子关系。

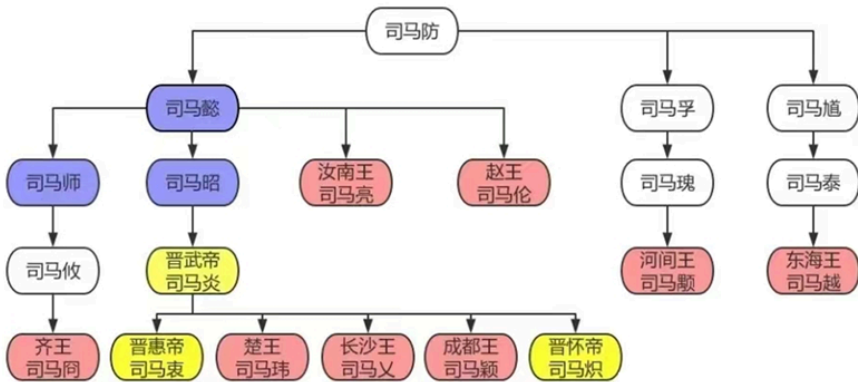


图 1 八王之乱的亲属关系示意图

```
1 import sqlite3
2 import os
3
4 conn = sqlite3.connect('family.db')
5 cursor = conn.cursor()
6
7 cursor.execute('DROP TABLE IF EXISTS family;')
8 # Drop the family table if it already exists
9 cursor.execute('''
10     CREATE TABLE family (
11         father TEXT,
12         son TEXT
13     );
14 ''')
```

<sqlite3.Cursor at 0x1da01bf9940>

```
1 # Insert the values
2 insert_data = [
3     ('司马防', '司马懿'),
4     ('司马防', '司马孚'),
5     ('司马防', '司马馗'),
6     ('司马懿', '司马师'),
7     ('司马懿', '司马昭'),
```

```

8      ('司马懿', '司马亮'),
9      ('司马懿', '司马伦'),
10     ('司马孚', '司马瑰'),
11     ('司马馗', '司马泰'),
12     ('司马师', '司马攸'),
13     ('司马昭', '司马炎'),
14     ('司马瑰', '司马颙'),
15     ('司马攸', '司马囧'),
16     ('司马炎', '司马衷'),
17     ('司马炎', '司马玮'),
18     ('司马炎', '司马义'),
19     ('司马炎', '司马颖'),
20     ('司马炎', '司马炽')
21 ]
22
23 cursor.executemany('INSERT INTO family VALUES (?, ?)', insert_data)
24 conn.commit()

```

我们定义如下规则：

brother(X,Y):-father(Z,X),father(Z,Y). 有共同父亲的是兄弟。

ancestor(X,Y):-father(X,Y). 父亲是祖先。

ancestor(X,Y):-father(X,Z),ancestor(Z,Y). 父亲的祖先是祖先。


请同学们完成如下 SQL 查询：

- 1) 找出所有的兄弟关系。
- 2) 使用递归查询找出所有祖先关系。

```

1  #1. Brother rule: brother(X,Y) :- father(Z,X), father(Z,Y)
2  def find_brothers():
3      cursor=conn.cursor()
4      cursor.execute('''
5          SELECT f1.son AS son1,f2.son AS son2
6          FROM family f1
7          JOIN family f2 on f1.father=f2.father
8          WHERE son1 < son2
9          ORDER BY son1,son2
10
11      ''')
12      brothers = cursor.fetchall()
13      print("Brothers(无重复):")
14      for b in brothers:
15          print(f"({b[0]},{b[1]})")


```

 Python

```

1  # 2. Ancestor rules:
2  #   ancestor(X,Y) :- father(X,Y)
3  #   ancestor(X,Y) :- father(X,Z), ancestor(Z,Y)
4  def find_ancestors():
5      # This requires a recursive query
6      cursor = conn.cursor()
7      cursor.execute('''
8          WITH RECURSIVE ancestor(ancestor, descendant) AS (

```

 Python

```
9      -- Base case: direct father-son relationships
10     SELECT father, son FROM family
11
12     UNION
13
14     -- Recursive case: father of someone who is already an ancestor
15     SELECT f.father, a.descendant
16     FROM family f
17     JOIN ancestor a ON f.son = a.ancestor
18 )
19 SELECT * FROM ancestor ORDER BY ancestor, descendant
20 '''
21 ancestors = cursor.fetchall()
22 print("\nAncestors:")
23 print("左边是右边的祖先: ")
24 for a in ancestors:
25     print(f"({a[0]},{a[1]})")
```

```
1 # Execute the queries
```

[Python](#)

```
2 find_brothers()
3 find_ancestors()
```

Brothers(无重复):

(司马义,司马炽)  
(司马义,司马玮)  
(司马义,司马衷)  
(司马义,司马颖)  
(司马亮,司马伦)  
(司马亮,司马师)  
(司马亮,司马昭)  
(司马伦,司马师)  
(司马伦,司马昭)  
(司马孚,司马懿)  
(司马孚,司马馗)  
(司马师,司马昭)  
(司马懿,司马馗)  
(司马炽,司马玮)  
(司马炽,司马衷)  
(司马炽,司马颖)  
(司马玮,司马衷)  
(司马玮,司马颖)  
(司马衷,司马颖)


Ancestors:

左边是右边的祖先:

(司马孚,司马瑰)  
(司马孚,司马颙)  
(司马师,司马囧)  
(司马师,司马攸)  
(司马懿,司马义)  
(司马懿,司马亮)  
(司马懿,司马伦)  
(司马懿,司马囧)  
(司马懿,司马师)  
(司马懿,司马攸)

(司马懿,司马昭)  
(司马懿,司马炎)  
(司马懿,司马炽)  
(司马懿,司马玮)  
(司马懿,司马衷)  
(司马懿,司马颖)  
(司马攸,司马囧)  
(司马昭,司马乂)  
(司马昭,司马炎)  
(司马昭,司马炽)  
(司马昭,司马玮)  
(司马昭,司马衷)  
(司马昭,司马颖)  
(司马炎,司马乂)  
(司马炎,司马炽)  
(司马炎,司马玮)  
(司马炎,司马衷)  
(司马炎,司马颖)  
(司马瑰,司马颙)  
(司马防,司马乂)  
(司马防,司马亮)  
(司马防,司马伦)  
(司马防,司马囧)  
(司马防,司马孚)  
(司马防,司马师)  
(司马防,司马懿)  
(司马防,司马攸)  
(司马防,司马昭)  
(司马防,司马泰)  
(司马防,司马炎)  
(司马防,司马炽)  
(司马防,司马玮)  
(司马防,司马瑰)  
(司马防,司马衷)  
(司马防,司马颖)  
(司马防,司马颙)  
(司马防,司马馗)  
(司马馗,司马泰)

```
1 #GROUP_CONCAT
2 def find_ancestors():
3     cursor = conn.cursor()
4     cursor.execute('''
5         WITH RECURSIVE ancestor(ancestor, descendant) AS (
6             -- 基础情况：直接的父子关系
7             SELECT father, son FROM family
8
9             UNION
10
11             -- 递归情况：祖先的祖先也是祖先
12             SELECT f.father, a.descendant
13             FROM family f
14             JOIN ancestor a ON f.son = a.ancestor
15         )
16     SELECT
```

 Python

```

17         ancestor,
18         GROUP_CONCAT(descendant, ', ') AS descendants
19     FROM ancestor
20     GROUP BY ancestor
21     ORDER BY ancestor
22     '')
23
24     ancestors = cursor.fetchall()
25     print("\n祖先及其后辈列表：")
26     print("=====")
27     for ancestor, descendants in ancestors:
28         print(f"{ancestor}: {descendants}")
29
30 # 调用函数
31 find_ancestors()

```


祖先及其后辈列表：

```

=====
司马孚：司马瑰、司马颙
司马师：司马攸、司马囧
司马懿：司马师、司马昭、司马亮、司马伦、司马攸、司马炎、司马囧、司马衷、司马玮、司马乂、司马颖、司马炽
司马攸：司马囧
司马昭：司马炎、司马衷、司马玮、司马乂、司马颖、司马炽
司马炎：司马衷、司马玮、司马乂、司马颖、司马炽
司马瑰：司马颙
司马防：司马懿、司马孚、司马馗、司马师、司马昭、司马亮、司马伦、司马瑰、司马泰、司马攸、司马炎、司马颙、司马囧、司马衷、司马玮、司马乂、司马颖、司马炽
司马馗：司马泰

```

```
1 conn.close()
```

 Python

## 2. JSON 操作

同学们选择我们课程的某个章节，结合大模型生成课程某个章节的内容，并以 JSON 格式输出，然后将其存入一个表中，定义一个 JSON 类型的字段来存储大模型的输出内容。


同学们的任务包括如下两项：

1. 自己设计两个查询实例，分别查询特定字段的值以及特定数组的元素数目。
2. 把 JSON 的嵌套结构展开，存入一个 1NF 平面表中。

```

1 import sqlite3
2 import json
3
4 # 创建数据库连接
5 conn = sqlite3.connect('database_course.db')
6 c = conn.cursor()
7
8 # 创建存储JSON的原始表
9 c.execute('''CREATE TABLE IF NOT EXISTS chapter_json (
10             id INTEGER PRIMARY KEY AUTOINCREMENT,
11             content JSON
12         )''')

```

 Python


```
<sqlite3.Cursor at 0x175a50b0540>
```

```
1  # 示例JSON数据
2  data = {
3      "chapter": "关系规范化理论",
4      "sections": [
5          {
6              "section_title": "基本概念与问题引入",
7              "knowledge_points": [
8                  {
9                      "key_point": "关系模式设计问题",
10                     "explanation": "非规范化关系模式会导致数据冗余、插入异常、更新异常和删除异常。例如学生选课表中存储系主任信息时，会导致系主任信息重复存储（冗余）且无法独立维护（异常）",
11                     "qa": {
12                         "question": "非规范化关系模式可能引发哪些问题？请举例说明。",
13                         "answer": "数据冗余（如系主任信息重复存储）、插入异常（无法单独插入未选课学生的系信息）、更新异常（修改系主任需更新多条记录）、删除异常（删除最后一条学生记录会丢失系信息）"
14                     }
15                 },
16                 {
17                     "key_point": "函数依赖",
18                     "explanation": "描述属性间逻辑关系的约束，包括完全依赖、部分依赖和传递依赖。如学号→系名是完全依赖，(学号,课程)→成绩是部分依赖，学号→系主任是传递依赖",
19                     "qa": {
20                         "question": "什么是传递函数依赖？请用学生-系-系主任的例子说明。",
21                         "answer": "若存在学号→系名，系名→系主任，且系名↛学号，则系主任传递依赖于学号"
22                     }
23                 }
24             ]
25         },
26         {
27             "section_title": "范式体系",
28             "knowledge_points": [
29                 {
30                     "key_point": "第一范式（1NF）",
31                     "explanation": "属性值不可再分，消除重复组。如将包含多值的地址字段拆分为省、市、街道",
32                     "qa": {
33                         "question": "判断表结构是否满足1NF：商品表(商品ID, 商品名称, 规格['红色','L'])",
34                         "answer": "不满足1NF，'规格'字段包含多个值，需拆分为独立属性或新建规格表"
35                     }
36                 },
37                 # 其他知识点...
38             ]
39         },


```

```
40     # 其他章节 ...
41 ],
42 "question_bank": [
43     {
44         "question": "Armstrong公理包含哪些基本规则? ",
45         "answer": "自反律 (若 $Y \subseteq X$ 则 $X \rightarrow Y$ )、增广律 ( $X \rightarrow Y$ 则 $XZ \rightarrow YZ$ )、传递律 ( $X \rightarrow Y$ 且 $Y \rightarrow Z$ 则 $X \rightarrow Z$ ) "
46     },
47     # 其他问题 ...
48 ]
49 }
```

```
1 # 插入JSON数据
2 c.execute("INSERT INTO chapter_json (content) VALUES (?)",
3           (json.dumps(data, ensure_ascii=False),))
4 conn.commit()
```


 Python

```
1 # 查询1: 获取特定字段值 (所有章节标题)
2 print("查询1: 所有章节标题")
3 c.execute('''SELECT json_extract(content, '$.chapter')
4           FROM chapter_json''')
5 print(c.fetchone()[0])
```

 Python


查询1: 所有章节标题  
关系规范化理论

```
1 # 查询2: 统计特定数组元素数目 (每个section的知识点数量)
2 print("\n查询2: 各章节知识点数量")
3 c.execute('''SELECT json_extract(s.value, '$.section_title'),
4           json_array_length(s.value, '$.knowledge_points')
5           FROM chapter_json,
6           json_each(chapter_json.content, '$.sections') AS s''')
7 for row in c.fetchall():
8     print(f"{row[0]}: {row[1]}个知识点")
```

 Python

查询2: 各章节知识点数量  
基本概念与问题引入: 2个知识点  
范式体系: 1个知识点

```
1 # 创建规范化平面表 (1NF)
2 c.execute('''CREATE TABLE IF NOT EXISTS normalized_chapter (
3           chapter TEXT,
4           section_title TEXT,
5           key_point TEXT,
6           explanation TEXT,
7           question TEXT,
8           answer TEXT
9           )''')
10
11 # 展开JSON结构插入平面表
12 c.execute('''INSERT INTO normalized_chapter
13           SELECT json_extract(content, '$.chapter'),
14           json_extract(s.value, '$.section_title'),
```

 Python

```


15         json_extract(kp.value, '$.key_point'),
16         json_extract(kp.value, '$.explanation'),
17         json_extract(kp.value, '$.qa.question'),
18         json_extract(kp.value, '$.qa.answer')
19     FROM chapter_json,
20         json_each(chapter_json.content, '$.sections') AS s,
21         json_each(s.value, '$.knowledge_points') AS kp'''
22 conn.commit()

```

```

1  # 验证规范化表
2  print("\n规范化表数据示例:")
3  c.execute("SELECT * FROM normalized_chapter LIMIT 2")
4  for row in c.fetchall():
5      print(row)
6
7  # 创建问题库表
8  c.execute('''CREATE TABLE IF NOT EXISTS question_bank (
9              id INTEGER PRIMARY KEY AUTOINCREMENT,
10             chapter TEXT,
11             question TEXT,
12             answer TEXT
13         )''')
14
15  # 插入问题库数据
16  c.execute('''INSERT INTO question_bank (chapter, question, answer)
17             SELECT json_extract(content, '$.chapter'),
18                    json_extract(q.value, '$.question'),
19                    json_extract(q.value, '$.answer')
20             FROM chapter_json,
21                    json_each(chapter_json.content, '$.question_bank') AS q''')
22  conn.commit()

```

 Python

规范化表数据示例：

('关系规范化理论', '基本概念与问题引入', '关系模式设计问题', '非规范化关系模式会导致数据冗余、插入异常、更新异常和删除异常。例如学生选课表中存储系主任信息时，会导致系主任信息重复存储（冗余）且无法独立维护（异常）', '非规范化关系模式可能引发哪些问题？请举例说明。', '数据冗余（如系主任信息重复存储）、插入异常（无法单独插入未选课学生的系信息）、更新异常（修改系主任需更新多条记录）、删除异常（删除最后一条学生记录会丢失系信息）')

('关系规范化理论', '基本概念与问题引入', '函数依赖', '描述属性间逻辑关系的约束，包括完全依赖、部分依赖和传递依赖。如学号→系名是完全依赖，(学号, 课程)→成绩是部分依赖，学号→系主任是传递依赖', '什么是传递函数依赖？请用学生-系-系主任的例子说明。', '若存在学号→系名，系名→系主任，且系名↛学号，则系主任传递依赖于学号')

### 3. 向量数据库实习设计

这部分我们的实习设计的比较简单，就是熟悉一下向量嵌入，以及数据库里面向量相似性检索的插件就可以了。我们提供了一个 Jupyter 文件，里面包括 SQLite 里面向量插件的使用，以及生成向量嵌入的 Python 包的使用。

实验用的数据集放在 sts-b-train.txt 中，格式如下，前面是两个句子，后面的数字代表这两个句子的相似性。因为这是一个训练集，所以相似值已经给定了，同学们要做的是生成两个句子的向量嵌入，然后调用 SQLite 的向量插件来计算它们的相似性，并和现有的相似值做个比较。可以截取一部分数据，另外也可以用其他数据库的向量插件。



1	一架飞机要起飞了。 一架飞机正在起飞。	5	<a href="#">Text</a>
2	一个男人在吹一支大笛子。 一个人在吹长笛。	3	
3	三个人在下棋。 两个人在下棋。	2	
4	一个人在拉大提琴。 一个坐着的人正在拉大提琴。	4	
5	有些人在战斗。 两个人在打架。	4	
6	一个男人在抽烟。 一个男人在滑冰。	0	
7	那人在弹钢琴。 那人在弹吉他。	1	
8	一个男人在弹吉他和唱歌。 一位女士正在弹着一把原声吉他，唱着歌。	2	
9	一个人正把一只猫扔到天花板上。 一个人把一只猫扔在天花板上。	5	

数据库 sentence.db schema 如下：

sentence(sid, sentence1, sentence2, similar\_score, sen1\_vector, sen2\_vector, vecSim\_score)

分别是句子 id，句子 1，句子 2，句子相似度得分，句子 1 嵌入向量，句子 2 嵌入向量，向量相似度得分)

任务要求：

调用嵌入模型：可以使用 sqlite-vss 或 sqlite-vec 或调用本地中文嵌入模型（huggingface 等拉取），对每个数据库中每个元组的两个句子分别进行向量嵌入，并计算向量相似度得分。更新数据库 sentence.db 的后三列。

输出要求：

按照这样的方法：

```
1 df = pd.read_sql_query("""
2     SELECT sid, sentence1, sentence2, similar_score, sen1_vector, sen2_vector,
3     vecSim_score FROM sentence
4 """, conn)
5 print(df.head(20))
```

输出数据库的前 20 个元组。但请注意，助教评分时可能会运行你的代码，抽查其他元组结果。

```
1 import sqlite3
2 import os
3 import pandas as pd
```

```
1 # 文件路径修改成自己的文件路径
2 DATA_FILE = 'sts-b-train.txt'
3 DB_FILE = 'sentence.db'
4
5 if os.path.exists(DB_FILE):
6     os.remove(DB_FILE)
7
8 conn = sqlite3.connect(DB_FILE)
9 cursor = conn.cursor()
```

```
1 cursor.execute('''
2 CREATE TABLE IF NOT EXISTS sentence (
3     sid INTEGER PRIMARY KEY AUTOINCREMENT,
4     sentence1 TEXT NOT NULL,
5     sentence2 TEXT NOT NULL,
6     similar_score REAL NOT NULL,
7     sen1_vector BLOB,
8     sen2_vector BLOB,
```

```

9     vecSim_score REAL
10 )
11 '''
12 conn.commit()

```

```

1 with open(DATA_FILE, 'r', encoding='utf-8') as f:
2     for line in f:
3         parts = line.strip().split('\t')
4         if len(parts) != 3:
5             continue
6         sentence1, sentence2, score = parts
7         cursor.execute(''
8             INSERT INTO sentence (sentence1, sentence2, similar_score)
9             VALUES (?, ?, ?)
10             ''', (sentence1, sentence2, float(score)))
11 conn.commit()

```

Python

```

1 df = pd.read_sql_query("""
2     SELECT sid, sentence1, sentence2, similar_score, sen1_vector, sen2_vector,
3     vecSim_score FROM sentence
4 """, conn)
5 print(df.head(10))

```

Python

	sid	sentence1	sentence2	similar_score \
0	1	一架飞机要起飞了。	一架飞机正在起飞。	5.0
1	2	一个男人在吹一支大笛子。	一个人在吹长笛。	3.0
2	3	一个人正把切碎的奶酪撒在比萨饼上。	一个男人正在把切碎的奶酪撒在一块未煮好的比萨饼上。	3.0
3	4	三个人在下棋。	两个人在下棋。	2.0
4	5	一个人在拉大提琴。	一个坐着的人正在拉大提琴。	4.0
5	6	有些人在战斗。	两个人在打架。	4.0
6	7	一个男人在抽烟。	一个男人在滑冰。	0.0
7	8	那人在弹钢琴。	那人在弹吉他。	1.0
8	9	一个男人在弹吉他和唱歌。	一位女士正在弹着一把原声吉他，唱着歌。	2.0
9	10	一个人正把一只猫扔到天花板上。	一个人把一只猫扔在天花板上。	5.0

	sen1_vector	sen2_vector	vecSim_score
0	None	None	None
1	None	None	None
2	None	None	None
3	None	None	None
4	None	None	None
5	None	None	None
6	None	None	None
7	None	None	None
8	None	None	None
9	None	None	None

```

1 from sentence_transformers import SentenceTransformer
2
3 # 加载模型，需要先下载到本地文件夹下
4 model = SentenceTransformer('./text2vec-base-chinese')

```

Python

```

1 cursor = conn.cursor()
2
3 cursor.execute("SELECT sid, sentence1, sentence2 FROM sentence")
4 rows = cursor.fetchall()

```

Python

```

5
6 # 提取句子和对应的 sid
7 sid_list = []
8 sentences1 = []
9 sentences2 = []
10
11 for row in rows:
12     sid, s1, s2 = row
13     sid_list.append(sid)
14     sentences1.append(s1)
15     sentences2.append(s2)
16
17 # 生成嵌入向量
18 embeddings1 = model.encode(sentences1, batch_size=64, show_progress_bar=True)
19 embeddings2 = model.encode(sentences2, batch_size=64, show_progress_bar=True)

Batches: 100%|██████████| 82/82 [00:44<00:00, 1.85it/s]
Batches: 100%|██████████| 82/82 [00:44<00:00, 1.85it/s]

```

```

1 import numpy as np
2 def cosine_similarity(vec1, vec2):
3     return float(np.dot(vec1, vec2) / (np.linalg.norm(vec1) * np.linalg.norm(vec2)))

```

Python

```

1 for sid, vec1, vec2 in zip(sid_list, embeddings1, embeddings2):
2     # 计算余弦相似度
3     sim_score = cosine_similarity(vec1, vec2)
4
5     # 将向量转换为二进制格式
6     vec1_blob = vec1.tobytes()
7     vec2_blob = vec2.tobytes()
8
9     # 更新数据库记录
10    cursor.execute("""
11        UPDATE sentence
12        SET sen1_vector = ?, sen2_vector = ?, vecSim_score = ?
13        WHERE sid = ?
14    """, (vec1_blob, vec2_blob, sim_score, sid))
15
16 # 提交更改
17 conn.commit()

```

Python

```

1 df = pd.read_sql_query("""
2     SELECT sid, sentence1, sentence2, similar_score, sen1_vector, sen2_vector,
3     vecSim_score FROM sentence
4 """, conn)
5 print(df.head(20))

```


Python

	sid	sentence1	sentence2	similar_score \
0	1	一架飞机要起飞了。	一架飞机正在起飞。	5.0
1	2	一个男人在吹一支大笛子。	一个人在吹长笛。	3.0
2	3	一个人正把切碎的奶酪撒在比萨饼上。	一个男人正在把切碎的奶酪撒在一块未煮好的比萨饼上。	3.0
3	4	三个人在下棋。	两个人在下棋。	2.0
4	5	一个人在拉大提琴。	一个坐着的人正在拉大提琴。	4.0

5	6	有些人在战斗。	两个人在打架。	4.0
6	7	一个男人在抽烟。	一个男人在滑冰。	0.0
7	8	那人在弹钢琴。	那人在弹吉他。	1.0
8	9	一个男人在弹吉他和唱歌。	一位女士正在弹着一把原声吉他，唱着歌。	
2.0				
9	10	一个人正把一只猫扔到天花板上。	一个人把一只猫扔在天花板	
		上。		
		5.0		
10	11	那人用棍子打了另一个人。	那人用棍子打了另一个人一巴	
		掌。		
		4.0		
11	12	一个人在吹长笛。	一个男人在吹竹笛。	3.0
12	13	一个人在叠一张纸。	有人在叠一张纸。	4.0
13	14	一条狗正试图把培根从他的背上弄下来。	一只狗正试图吃它背上的培根。	
3.0				
14	15	北极熊在雪地上滑行。	一只北极熊在雪地上滑行。	5.0
15	16	一个女人在写作。	一个女人在游泳。	0.0
16	17	一只猫在婴儿的脸上摩擦。	一只猫在蹭婴儿。	3.0
17	18	那人在骑马。	一个人骑着马。	5.0
18	19	一个人往锅里倒油。	一个人把酒倒进锅里。	3.0
19	20	一个男人在弹吉他。	一个女孩在弹吉他。	2.0

	sen1_vector \	
0	b'tw\xbbb\xbe\xbc=#>\xfc\x86\xac=lG→6\x8aw>wzU...	
1	b'\t\x03\xa5\xbc\xfc\xdd\x92\xbe\xc1&\x86\xbf\...	
2	b'\xac\xc1\n?\x0b\xbb\x0b?\x8dz\x1b\xbe\x83\xd...	
3	b'\xd5C\xfd>T\xfa#\xbf\xac\xcd\x04?\x18\x1e\x9...	
4	b'Q\x92\xde;;q\x04?\xb0\x1d\x83\xbe\xd3\x08\x9...	
5	b'\x8c\xcf\xdc5\xbe\x0M'\xbe\xed\x9\xbeu\x19...	
6	b'W I\xbe'\xfaz?Z\x1dq?zP\xab?\xb0a\x12\xbdH\x...	
7	b's\x8V?\xcbj\xfc=\x97\xeeh\xbe\x84 5?h\x7=...	
8	b'\xce\x80\xed>\xaf\xfc>\x83\xbe7?\xc8H"?h\x...	
9	b'U1\xc5\xbd}p\x98=\x17\xdc\xef=jWR\xbe\x7\x...	
10	b'&\x88\x4={\x19\t\xbe\x19\x99\xbeI\xcd ?...	
11	b'\x92c\x90\xbe\xdc9\x8b\x1\xbe\xdc/\x00\xbf>...	
12	b'\x17B\x9\xbe\xae:\x06>\x9c\x9c\x03?\xd6\xd5...	
13	b'\xc80\x6>\x0e\xedS\xbf\x7=D>2\xba\xaa\xbe...	
14	b'K\x2P\xbd+\xbb\x08\xbf};\x88\xbe\xfc\xab\xc...	
15	b'Q\x1&?\xee\x0b\xef\xbdQ(?x85wb?\xc2U\xab\...	
16	b'\x03M\x4\xbd7\x9c\x9f\xbdn\x0\x14>>\x81k\x...	
17	b'\x13\xa3\x0<\xb8\x0bX\xbfUG\x01?\xff\x15\x9...	
18	b'\xb2\x8a\x19?jQ\xae?\xf9\x16-\xbd\x7\x82^>\$...	
19	b'\x97\x85-?\xe2w&?\xa6\xfc>\x1c\xfc62>\xbd\x...	
	sen2_vector	vecSim_score
0	b'_\x96\x1e\xbf\x11_y>\xec\x88\x03?\xdc(\xf1=...	0.956823
1	b'\x92c\x90\xbe\xdc9\x8b\x1\xbe\xdc/\x00\xbf>...	0.884109
2	b'\x16\x0c\x0b?\xc5\xbd\x0e?\x9aU:\xbe\xfc4\x8...	0.974107
3	b'\xf2&\xe4>'{'\xbf\xdc]J?\xb7)S?\x079X?h\x84...	0.747162
4	b'P\xfe\x8f=\x11\x06?)\xa0\x1\xbe\xfc9\x85\x8...	0.961783
5	b'\x12h\x09\xbc\x93o\x9c\xbe)&\xbe"\xd8\x17?\...	0.846043
6	b'\xba\xdb\x1?\xa6P\xa9=\x9a\xcd\x4\xbd\xfc0m...	0.231173
7	b'IP\x07?B\xbb\x5>\xc0;\xef>\xf5\x93\x87>i\xfc...	0.484320
8	b'\xdbi>;\xdb\xa3<\xbb\xcb6\x85\xbd{\xdf\x8b?\...	0.734604
9	b'\xb8\x8\x14\xbd\xca\x81\x1\xbd2\x86\x9=\x...	0.983240
10	b'\xcd\x88>\xa2\x2\x8c>\xd0\x18,\xbe\x9e\xcc...	0.909791
11	b'j\x0\x86>\x89\x89\x1a=\xc3lq\xbf\x81\xfc6\x...	0.836796
12	b'\x1f\xdc2\xaa\xbe\xfc8\xa3j>\xb9\xfa>\x88\x9b...	0.979772
13	b'r\x93\xa8>\xfa\x1e\x95\xbeu\xdc9\xfa\xbe\x89\...	0.798483
14	b'\xae\xa9\x7\xbc\x10\x2\xda\xbe\x0e\x9d\x83...	0.985583
15	b'H\xfc8\x95?Nm\x8>\xa2L\x0e\xbf\x3\x95\x8e>a...	0.425817
16	b'A\xac\xa7\xbea\xba=\xbf*A\x80\xbf\x9b=\xfdc\x...	0.798582
17	b'=t\t=DIS\xbf\x7E*>\x9b\xabw?\x8d\x1e)\xbe\x...	0.944069
18	b'8=E?D\x82\xdc?\xd5\$\r\xbc\xad\x1f<\xc9q\x85...	0.687595
19	b'd\xaeA?\xf6X\xeb<PL\x94>\xcd\xa3\x9e>\xe9\x...	0.706035

```
1 conn.close()
```

 Python