《 智能系统 》实验报告

实验题目	数据库与知识库设计

一、实验目的

为实现十字路口红绿灯智能控制，本次实验的目的是：

（1） 练习知识库的设计，测试与维护

（2） 大型知识库的实现设计。

（3） 可信度知识库的设计与实现（可选）

二、实验项目内容

1、 基于 pyknow，设计十字路口红绿灯智能控制的知识库

（1） 基于 pyknow 语法，给出十字路口红绿灯智能控制的完整知识库。

（2） 通过模拟运行，测试并修改知识库，保证十字路口红绿灯智能控的正确性。

2、 基于关系数据库管理系统，（如 mysql），设计并实现大型知识库的存储和管理。

（1） 为什么需要用关系数据库来存储大型知识库？

（2） 对 1 中的知识库，设计出表结构（给出 DDL 程序）

（3） 将 1 中的知识库存入关系数据库（给出 DML 程序）

（4） 举例给出知识库的使用和维护过程（给出 DML 程序）

3、可信度知识库设计（可选）

三、实验过程或算法（代码）

1. 基于pyknow，设计十字路口红绿灯智能控制的知识库

（1）十字路口红绿灯智能控制的完整知识库

根据现实生活中的十字路口交通情况，设定以下规则：

① 红绿灯总时间60s；

② 红绿灯每次更改时间单位6s；

因为总车辆数=南北方向车辆数+东西方向车辆数，所以红绿灯的时间分配也应当考虑某方向的车辆占总车辆的比值来决定当前方向分配到的绿灯时间占总时间的比值。另一侧分配的时间则由总时间减此侧时间得出。因此补充知识库规则如下：

③ 如果南北方向车辆数与东西方向车辆数均＜15辆，则红灯与

报告创建时间：

绿灯时间比例1:1；
④ 如果南北方向车辆数/总车辆 > 当前南北方向时间/总时间，则给南北向增加一份绿灯比例；
⑤ 如果南北方向车辆数/总车辆 < 当前南北方向时间/总时间，则给南北向减少一份绿灯比例；
⑥ 其余情况维持不变；
基于以上规则，参考https://gitee.com/xingyongkang/trafficLights/，实现pyknow十字路口红绿灯智能控制知识库如下：

```
import time
import random
from experta import *
exeTimes = 30  # time in second

class TrafficLights(KnowledgeEngine):
    @DefFacts()
    def _initial_action(self):
        yield Fact(Ticks = 0)
        yield Fact(Second = 0)
        yield Fact(ASecond = False)
        yield Fact(SwitchTime = 5)
        yield Fact(Period = 10)
        yield Fact(NSLight = 'GREEN')
        yield Fact(WELight = 'RED')
        yield Fact(NSCars = 0)
        yield Fact(WECars = 0)

    # 设置计数规则
    @Rule(AS.oldFact << Fact(Ticks = MATCH.times))

    def ticks(self,times,oldFact):
        self.retract(oldFact) # 收回事实
        self.declare(Fact(Ticks = times + 1))
        time.sleep(0.1)
        if times == exeTimes * 10:
            print('bye!')
            self.halt()
        else:
            if times%10 == 0:
```

```
30.                     self.declare(Fact(ASecond = True))
31.                 self.comm()
32.
33.         # 时间递增规则
34.         @Rule(AS.fact1 << Fact(Second=MATCH.times),
35.               AS.fact2 << Fact(ASecond = True), # 一秒钟过去
36.               salience= 1
37.         )
38.         def step(self,times,fact1,fact2):
39.             self.retract(fact1)
40.             self.retract(fact2)
41.             self.declare(Fact(Second= times + 1)) # 声明秒数加 1 的事实
42.             print("{}-*-".format(times) ) # 实时打印出来
43.
44.         # 1 是时间递增规则
45.         # 2 是南北车辆规则
46.         # 3 是东西车辆规则
47.         # 4 是转换时间规则
48.         @Rule(AS.fact1 << Fact(Second=MATCH.times),
49.               Fact(Period = MATCH.period),
50.               TEST(lambda times,period: times == period),
51.               AS.fact2 << Fact(NSCars = MATCH.nsCars),
52.               AS.fact3 << Fact(WECars = MATCH.weCars),
53.               AS.fact4 << Fact(SwitchTime= MATCH.switchTime),
54.               salience = 2
55.             )
56.         # 转换函数
57.         def startSwitch1(self,fact1,fact2,fact3,fact4,nsCars,weCars,switchTime,period):
58.             self.retract(fact1)
59.             self.retract(fact2)
```

```
            self.retract(fact3)
            self.retract(fact4)
            self.decision(nsCars,weCars,switchTime,period)
            # 转换完毕后重置参数
            self.declare(Fact(Second = 0))
            self.declare(Fact(Switch = True))
            self.declare(Fact(NSCars = 0))
            self.declare(Fact(WECars = 0))

        # 设置转换规则 2
        @Rule(
              Fact(Second = MATCH.times),
              Fact(SwitchTime = MATCH.switchTime),
              TEST(lambda switchTime,times:times == switchTime),
              salience= 2
           )
        def startSwitch2(self):
            self.declare(Fact(Switch = True))
        # 设置前一次红绿灯情况，换灯规则 1
        @Rule(
            AS.oldSwtich << Fact(Switch = True),
            AS.oldNS << Fact(NSLight = 'RED'),
            AS.oldWE << Fact(WELight = 'GREEN'),
            salience = 2
          )
        def switch1(self,oldSwtich,oldNS,oldWE):
            self.declare(Fact(NSLight = 'GREEN'))
            self.declare(Fact(WELight = 'RED'))
            self.retract(oldSwtich)
            self.retract(oldWE)
            self.retract(oldNS)
        # 换灯规则 2
        @Rule(
            AS.oldSwtich << Fact(Switch = True),
            AS.oldNS << Fact(NSLight = 'GREEN'),
            AS.oldWE << Fact(WELight = 'RED'),
            salience = 2
```

```
97.           )
98.       def switch2(self,oldSwtich,oldNS,oldWE):
99.           self.declare(Fact(NSLight = 'RED'))
100.          self.declare(Fact(WELight = 'GREEN'))
101.          self.retract(oldSwtich)
102.          self.retract(oldWE)
103.          self.retract(oldNS)
104.      # 展示灯当前情况的规则
105.      @Rule(
106.          Fact(NSLight = MATCH.NScolor),
107.          Fact(WELight = MATCH.WEcolor),
108.          salience = 2
109.        )
110.      def show(self,NScolor,WEcolor):
111.          #print('\n NS:WE={}:{} \n'.format(NScolor,W
    Ecolor))
112.          if NScolor == 'RED':
113.              print('-X-  -V-')
114.          else:
115.              print('-V-  -X-')
116.
117.      # 南北车辆计数规则
118.      @Rule(
119.          AS.fact1 << Fact(NSSign = True),
120.          AS.fact2 << Fact(NSCars = MATCH.cars)
121.      )
122.      def countNS(self,fact1,fact2,cars):
123.          self.retract(fact1)
124.          self.retract(fact2)
125.          self.declare(Fact(NSCars = cars + 1 ))
126.
127.      # 东西车辆计数规则
128.      @Rule(
129.          AS.fact1 << Fact(WESign = True),
130.          AS.fact2 << Fact(WECars = MATCH.cars)
131.      )
132.      def countWE(self,fact1,fact2,cars):
133.          self.retract(fact1)
134.          self.retract(fact2)
```

```
135.             self.declare(Fact(WECars = cars + 1 ))
136.
137.         # 转换红绿灯的持续时间
138.         def decision(self,nsCars,weCars,switchTime,peri
    od):
139.             if nsCars == 0:
140.                 nsCars = 1
141.             if weCars == 0:
142.                 weCars = 1
143.             newSwitchTime = int(nsCars/(nsCars+weCars)*
     period)
144.             if newSwitchTime == 0:
145.                 newSwitchTime = 1
146.             if newSwitchTime == period:
147.                 newSwitchTime = period -1
148.
149.             print('nsCars ={} and weCars={},so we chang
    ed switch time from {} to {}'.format(nsCars,weCars,sw
    itchTime,newSwitchTime))
150.             self.declare(Fact(SwitchTime = newSwitchTim
    e))
151.
152.         def comm(self):
153.             if random.randint(0,5) == 0 :
154.                 self.declare(Fact(NSSign = True))
155.             if random.randint(0,5) == 0 :
156.                 self.declare(Fact(WESign = True))
157.
158.     def main(args = None):
159.         engine =TrafficLights()
160.         engine.reset()
161.         engine.run()
162.
163.     if __name__ == "__main__":
164.         main()
```

需要注意的是，实验框架的 pyknow 包已经不在 pypi 中，有两种解决办法：

- 下载 pyknow-develop，然后使用 python+pyknow 下 setup.py 的绝对路径+install 命令安装；

```
PS C:\Users\Jackson1125\Desktop\智能系统\trafficLights-master> python D:\KSoftware\P
ython\Lib\site-packages\pyknow-develop\setup.py install
running install
D:\KSoftware\Python\Lib\site-packages\setuptools\command\install.py:34: SetuptoolsDe
precationWarning: setup.py install is deprecated. Use build and pip and other standa
rds-based tools.
  warnings.warn(
D:\KSoftware\Python\Lib\site-packages\setuptools\command\easy_install.py:144: EasyIn
stallDeprecationWarning: easy_install command is deprecated. Use build and pip and o
ther standards-based tools.
  warnings.warn(
running bdist_egg
running egg_info
creating pyknow.egg-info
writing pyknow.egg-info\PKG-INFO
writing dependency_links to pyknow.egg-info\dependency_links.txt
writing requirements to pyknow.egg-info\requires.txt
writing top-level names to pyknow.egg-info\top_level.txt
writing manifest file 'pyknow.egg-info\SOURCES.txt'
reading manifest file 'pyknow.egg-info\SOURCES.txt'
writing manifest file 'pyknow.egg-info\SOURCES.txt'
installing library code to build\bdist.win-amd64\egg
running install_lib
running build_py
creating build
```

- 换为 experta 包即可：pip install experta；

建议使用后一种，因为前一种还有可能遇到版本冲突问题。

（2）模拟运行，测试并修改知识库

```
-V-  -X-
0-*-
1-*-
2-*-
3-*-
```

```
7-*-
8-*-
9-*-
nsCars =18 and weCars=9,so we changed switch time from 5 to 6
-V-  -X-
0-*-
```

```
 -X-  -V-
 6-*-
 7-*-
 8-*-
 9-*-
 nsCars =26 and weCars=14,so we changed switch time from 6 to 6
 -V-  -X-
 bye!
```

2. 基于关系数据库管理系统，设计并实现大型知识库的存储和管理

（1）需要用关系数据库来存储大型知识库的原因

关系数据库在存储大型知识库时能够提供结构化数据存储、强大的查询功能、数据完整性和一致性、扩展性和性能优化以及数据安全和权限控制等优势，具体如下：

• 结构化数据存储：关系数据库使用表格的形式来组织和存储数据，这使得数据能够被高效地组织、查询和管理。对于大型知识库而言，拥有结构化的数据存储方式可以提供更好的数据组织和管理能力；

• 灵活的查询语言：关系数据库使用结构化查询语言（SQL）进行数据查询和操作，SQL 提供了强大的查询功能，能够方便地执行各种复杂的查询操作，例如连接多个表、过滤和排序数据等。这对于大型知识库的数据查询和分析非常重要；

• 数据完整性和一致性：关系数据库通常支持事务处理，可以确保在数据插入、更新或删除时保持数据的完整性和一致性。这是非常关键的，特别是在大型知识库中，数据的准确性和一致性对于知识的正确性和可靠性至关重要；

• 扩展性和性能优化：关系数据库可以通过水平和垂直扩展来应对大型知识库的存储需求。水平扩展是通过在多台服务器上分布数据来增加存储容量和处理能力，而垂直扩展则是通过增加单台服务器的处理能力来提高性能。此外，关系数据库还提供了各种性能优化技术，如索引、查询优化等，以提高数据查询的速度和效率；

• 数据安全和权限控制：关系数据库通常提供安全性和权限控制机制，可以限制用户对数据的访问和操作权限。这对于大型知识库来说至关重要，可以确保只有授权的用户能够访问和修改知识库中的数据，并保护敏感信息的安全；

因此，我们使用关系数据库来存储大型知识库。

（2）知识库的表结构

变量说明如下：

变量名	说明
NScar	南北方向车辆数
EWcar	东西方向车辆数
NSmorethanEW	单位时间内通过南北侧车辆是否大于通过东西侧车辆的数量（即南北车辆/总车辆是否大于南北绿灯时间/总时间）；单位时间内通过南北侧车辆大于通过东西侧车辆的数量时 NSmorethanEW 为 1，反之为 0.
AddNS	是否要增加南北侧绿灯分配时间，为 1 增加，为 0 不增加

AddEW	是否要增加东西侧绿灯分配时间，为 1 增加，为 0 不增加
Reset	是否将红绿灯分配时间重置为初始状态（各 30s），为 1 重置,为 0 不重置

表结构如下：

DDL 程序如下：

```
CREATE TABLE `trafficLights` (
  `id` INT auto_increment,
  `NScar` INT  NULL,
  `EWcar` INT  NULL,
  `NSmorethanEW` TINYINT(1) NULL,
  `AddNS` TINYINT(1) NULL,
  `AddEW` TINYINT(1) NULL,
  `reset` TINYINT(1) NULL);
```

（3）将知识库存入关系数据库

需要装入知识库的规则如下：

```
NSmorethanEW_p = 1 if nsCars / (nsCars + weCars) > switch
Time / period else 0;
AddEW_p = 1 if NSmorethanEW_p == 1 else 0;
AddEW_p = 1 if NSmorethanEW_p == 0 else 0;
reset_p = 1 if nsCars / (nsCars + weCars) == switchTime /
 period else 0;
```

因此插入数据的 DML 指令如下：

```
INSERT INTO trafficLights (NScar,EWcar,NSmorethanEW
,AddNS,AddEW,reset) VALUES (NSCars,EWcars,NSmorethanEW
_p,AddNS_p,AddEW_p,reset_p);
```

（4）知识库的使用和维护过程

SQL语句获取最新数据如下：

```
SELECT * from trafficLights where id = (SELECT max(
```

```
    id) FROM trafficLights); #获取最新一条的数据
```

程序如下：

```
1.    # 检索人员信息
2.    def searchUser():
3.        db, cursor = connectDataBase()
4.        sql = "SELECT * from a where id = (SELECT max(i
    d) FROM a);"
5.        try:
6.            # 执行 sql 语句
7.            cursor.execute(sql)
8.            # 提交到数据库执行
9.            results = cursor.fetchone()
10.           # 获取数据
11.           db.commit()
12.           if results:
13.               print("已存在")
14.               # 关闭数据库连接
15.               db.close()
16.               return results
17.           else:
18.               print("不存在")
19.               # 关闭数据库连接
20.               db.close()
21.               return None
22.       except:
23.           print("Error: unable to fetch data")
24.           # 关闭数据库连接
25.           db.close()
26.           # 返回 None
27.           return None
```

四、实验结果及分析

刚开始阶段如下：

```
-V-  -X-
0-*-
1-*-
2-*-
3-*-
4-*-
-X-  -V-
5-*-
6-*-
7-*-
```

初始时 swtich time 为 30 秒，即南北-东西两个方向所分配到的绿灯时间相同。经过一个周期 60s 后，统计到南北车辆为 92，东西方向车辆为 114，根据设定的规则，此时南北车辆数/总车辆 < 当前南北时间/总时间，所以给东西向减少一份绿灯比例，为 26s：

```
59-*-
nsCars =92 and weCars=114,so we changed switch time from 30 to 26
-V-  -X-
0-*-
1-*-
2-*-
```

可以看到修改时间后，南北方向经过 26s 后变红灯：

```
22-*-
23-*-
24-*-
25-*-
-X-  -V-
26-*-
27-*-
28-*-
29-*-
```

最终结束：

```
35-*-
36-*-
37-*-
38-*-
39-*-
bye!
```