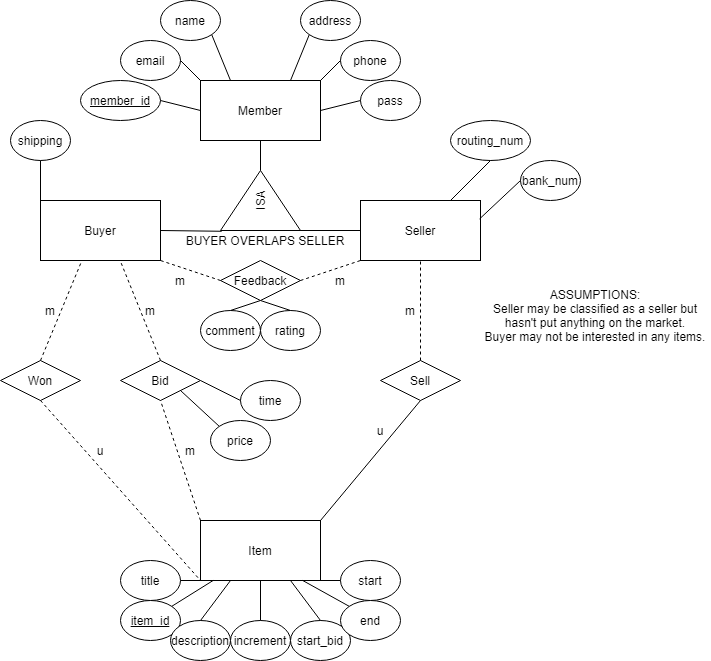
1. 
2. SQL

CREATE TABLE IF NOT EXISTS faculty(

FID int,

address varchar(80),

phone int,

name varchar(42),

PRIMARY KEY(FID)

);

CREATE TABLE IF NOT EXISTS students(

SID int,

name varchar(42),

degree varchar(20),

advisor\_ID int NOT NULL, # total-uni particpation "workaround"

FOREIGN KEY(advisor\_ID) REFERENCES faculty(FID),

PRIMARY KEY(SID)

);

CREATE TABLE IF NOT EXISTS labOffice(

ID int,

seats int,

address varchar(50),

PRIMARY KEY(ID)

);

CREATE TABLE IF NOT EXISTS advises(

FID int,

SID int UNIQUE, # gurantees only one advisor relationship per student

PRIMARY KEY(SID),

FOREIGN KEY(FID) REFERENCES faculty(FID),

FOREIGN KEY(SID) REFERENCES students(SID)

);

CREATE TABLE IF NOT EXISTS works(

office\_id int,

sid int,

since datetime,

FOREIGN KEY(office\_id) REFERENCES labOffice(ID),

FOREIGN KEY(sid) REFERENCES students(SID)

)

1. Relational Algebra

|  |
| --- |
| **Level** |
| Undergraduate |



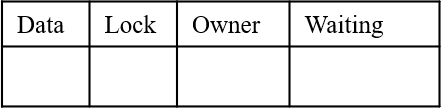
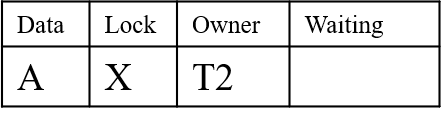
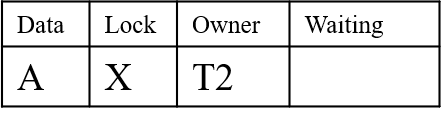
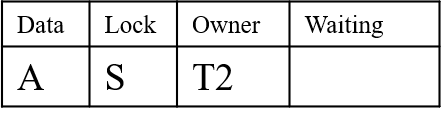
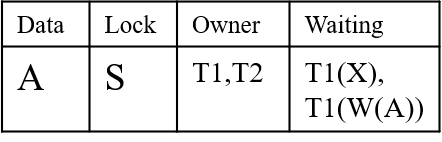
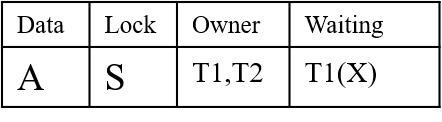
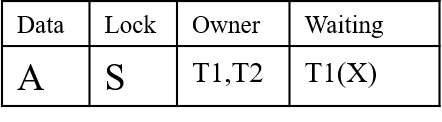
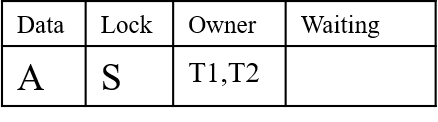
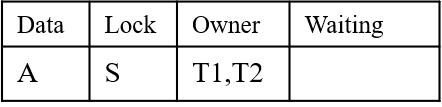
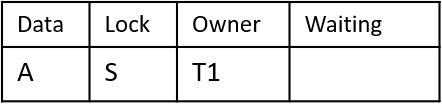
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Students.snum** | **Students.name** | **gender** | **Majors.snum** | **Majors.name** | **level** |
| 1001 | Randy | M | 1001 | Computer Science | BS |
| 1001 | Randy | M | 1005 | Applied Mathematics | MS |
| 1005 | Nicole | F | 1001 | Computer Science | BS |
| 1005 | Nicole | F | 1005 | Applied Mathematics | MS |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **snum** | **Minors.name** | **Minors.level** | **Degrees.name** | **Degrees.level** | **department\_code** |
| 1005 | Computer Science | BS | Computer Science | BS | 401 |
| 1005 | Computer Science | BS | Computer Science | MS | 401 |
| 1005 | Computer Science | BS | Computer Science | PHD | 401 |
| 1001 | Software Engineering | BS | Software Engineering | BS | 401 |

* 2. SELECT m.name FROM Students s JOIN Majors m ON s.snum=m.snum AND s.name = “Randy”;

1. Left-Deep
   1. Allows us to generate fully pipelined plans while reducing the search space
   2. 
2. Block Nested Loop Join
   1. M\*Cr seconds to load R
   2. N\*Cr seconds to load S
   3. (M+ceil(M/(B-2))\*N)\*Cw seconds to write results
3. Sorting
   1. Pass 1: Load 3 pages at a time. Sort and merge them. 30/3 = 10 sorted lists
   2. Pass 2: Load 2 lists in 2 pages of memory, use third page as output (two-way merge). 10/2=5 lists with 6 pages each
   3. Pass 3: Perform two-way merge again. 5/2=3 lists. First two lists have 12 pages, third list has 6.
   4. Pass 4: Repeat. 3/2=2 lists. One with 24, other with 6
   5. Pass 5: Repeat. 2/2 = 1 sorted list with 30 pages.
   6. 5\*2\*30=300 pages I/O cost
4. Schedules
   1. S1: No, No, No
   2. S2: No, Yes, Yes
   3. S3: No, Yes, Yes

1. Lock Tables



1. MGL
   1. T1
      1. IS on db GRANTED
      2. IS on f1 GRANTED
      3. IS on p1 GRANTED
      4. S on r3 GRANTED
   2. T2
      1. IX on db GRANTED
      2. IX on f2 GRANTED
      3. IX on p3 REJECTED (already has S)
      4. X on r6 REJECTED (parent has no IX or SIX)
2. R-tree
   1. Overlap Search
      1. (R1, R2)
      2. (R3, R4, R5), (R6, R7)
      3. (R8, R9, R10)
   2. R1, R5
3. Data Mining
   1. 2/6 = .33333333333333333333333333333333333…
   2. Confidence: 2/3; Support: 2/6 = 1/3
   3. 2 item sets
      1. {HotDogs}, {Coke}
      2. {HotDogs}, {Chips}
      3. {Coke}, {Chips}