Homework Assignment

(Deadline: Friday, 15-Nov-2024, 11:59pm. Submit via Blackboard)

Question A. Integrity Constraints [35 marks]

The Department of Computing wants to create a database to manage the information about its staff, students, and courses. The tables, attributes in each table, and data type of each attribute are summarized below, along with some detailed requirements for the information to be stored in the database. Write SQL statements to create these tables and specify the integrity constraints, e.g., domain constraint (including custom domains), not null, primary key, foreign key, etc.

Attribute	Data Type	Requirement
Staff		
ID	INTEGER	Each staff must have a unique ID.
Name	VARCHAR(50)	Each staff must have a name.
Email	VARCHAR(50)	Email must be in the form of '…@staff.comp.polyu.hk'.
Position	VARCHAR(20)	The department offers three positions: 'Assistant Professor', 'Associate Professor', 'Professor'.
Office	CHAR(5)	Office must be in the form of 'PQ…'.
Salary	INTEGER	Monthly salary must be between 80000 and 160000.
Student		
ID	INTEGER	Each student must have a unique ID
Name	VARCHAR(50)	Each student must have a name.
Email	VARCHAR(50)	Email must be in the form of '…@student.comp.polyu.hk'.
Type	VARCHAR(20)	Student type is either 'Domestic' or 'International'.
Major	CHAR(2)	The department offers four majors: 'CS', 'IT', 'DS', 'AI'.
AvgGrade	DECIMAL(2,1)	Grade must be between 0 and 4.3.
Advisor	INTEGER	Each student must be assigned a staff as his/her advisor; A staff cannot leave the department if he/she is the advisor of some students.
Course		
ID	INTEGER	Each course must have a unique ID.
Name	VARCHAR(50)	Each course must have a name.
Credits	INTEGER	Credits must be between 1 and 6.
Teach		
StaffID	INTEGER	In each semester ('Spring', 'Summer', or 'Fall'), a staff can teach more than one course; a course can be co-taught by multiple staff. A staff can teach the same course multiple times in different semesters; all their evaluations (between 0 and 4) must be recorded in the database. Staff can teach only the courses offered by the department; Courses can be taught only by staff from the department. When a staff leaves the department or a course is closed, all the teaching and evaluation information must be deleted from the database.
CourseID	INTEGER	
Semester	CHAR(6)	
Evaluation	DECIMAL(2,1)	

Enrolment		
StudentID	INTEGER	Each student can take more than one course; each course can be enrolled by many students. A student can take the same course multiple times, but only the last grade is recorded in the database; The grade must be between 0 and 4.3. Students can take only the courses offered by the department; Courses are only opened for students from the department. When a student leaves the department, all his/her enrolment information must be deleted from the database; A course cannot be closed if there are students enrolled in it.
CourseID	INTEGER	
Grade	DECIMAL(2,1)	

Question B. Functional Dependency and Normalization [25 marks]

(1) Consider relation R(A, B, C, D, E, F) and the set of functional dependencies $\mathcal{F} = \{AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, BC \rightarrow A, E \rightarrow BCF\}$. Find all the candidate keys for R by calculating the attribute closures.

(2) Consider relation R(A, B, C, D, E, F, G, H) and the set of functional dependencies $\mathcal{F} = \{A \rightarrow CD, C \rightarrow EF, G \rightarrow A, CE \rightarrow F, BG \rightarrow H\}$. Determine the highest normal form of R, and decompose R into 2NF, 3NF and then BCNF relations. Show your steps of decomposition.

Question C. File Organization and Indexing [40 marks]

Consider the relation: *Vehicle (<u>VID</u>, Maker, Model, Color, Price)*, which contains 6400 records with each record occupying 50 bytes.

- *VID* (Integer, 4 bytes) is the primary key of the relation; the values are between 1 and 6400.
- *Maker* contains 80 distinct values; the records are evenly distributed among these values.
- *Color* contains 10 distinct values; the records are evenly distributed among these values.

The file is sorted by *VID*, and stored on a disk with the following configuration:

- Block size = 1000 bytes
- Block pointer size = 6 bytes

(1) What is the total cost (i.e., number of block accesses) to retrieve all records with *VID* $>$ 6000 using linear scan? What is the total cost to retrieve these records using binary search?

(2) A primary index is built on the *VID* field. What is the total cost to retrieve all records with *VID* $>$ 6000 using the primary index?

(3) A B^+ tree index is built on the *VID* field, and each tree node is 60% full on average. What is the total cost to retrieve all records with VID $>$ 6000 using the B^+ tree index? How would the total cost change if the file is not sorted by *VID*?

(4) A bitmap index is built on both Maker and Color fields. How many blocks are required to store the bitmap index (*Note:* 1 byte = 8 bits; each bitmap is stored as a fixed-length record)? Assume there are 40 records with Maker = 'Toyota' and Color = 'Black'. What is the total cost to retrieve these records using the bitmap index?

Student Name: ______________________ Student ID: ______________________

Provide your answers for Question A on this page.

// SQL statements for creating the Staff table:

```
CREATE DOMAIN Staff_Email AS VARCHAR(50)
CHECK (VALUE LIKE '%@staff.comp.polyu.hk');    [1 mark]

CREATE DOMAIN Staff_Position AS VARCHAR(20)
CHECK (VALUE IN ('Assistant Professor', 'Associate Professor', 'Professor'));    [1 mark]

CREATE DOMAIN Staff_Office AS CHAR(5)
CHECK (VALUE LIKE 'PQ%');    [1 mark]

CREATE DOMAIN Staff_Salary AS INTEGER
CHECK (VALUE ≥ 80000 AND VALUE ≤ 160000);    [1 mark]

CREATE TABLE Staff (
        ID INTEGER,
        Name VARCHAR(50) NOT NULL,    [1 mark]
        Email Staff_Email,
        Position Staff_Position,
        Office Staff_Office,
        Salary Staff_Salary,
        PRIMARY KEY (ID)    [1 mark]
);
```

// SQL statements for creating the Student table:

```
CREATE DOMAIN Student_Email AS VARCHAR(50)
CHECK (VALUE LIKE '%@student.comp.polyu.hk');    [1 mark]

CREATE DOMAIN Student_Type AS VARCHAR(20)
CHECK (VALUE IN ('Domestic', 'International'));    [1 mark]

CREATE DOMAIN Student_Major AS CHAR(2)
CHECK (VALUE IN ('CS', 'IT', 'DS', 'AI'));    [1 mark]

CREATE DOMAIN Student_Grade AS DECIMAL(2, 1)
CHECK (VALUE ≥ 0.0 AND VALUE ≤ 4.3);    [1 mark]

CREATE TABLE Student (
        ID INTEGER,
        Name VARCHAR(50) NOT NULL,    [1 mark]
        Email Student_Email,
        Type Student_Type,
        Major Student_Major,
        AvgGrade Student_Grade,
        Advisor INTEGER NOT NULL,    [1 mark]
```

```
        PRIMARY KEY (ID),    [1 mark]
        FOREIGN KEY (Advisor) REFERENCES Staff(ID) ON DELETE RESTRICT    [2 marks]
);
```

// SQL statements for creating the Course table:

```
CREATE DOMAIN Course_Credit AS INTEGER
CHECK (VALUE ≥ 1 AND VALUE ≤ 6);    [1 mark]

CREATE TABLE Course (
        ID INTEGER,
        Name VARCHAR(50) NOT NULL,    [1 mark]
        Credits Course_Credit,
        PRIMARY KEY (ID)    [1 mark]
);
```

// SQL statements for creating the Teach table:

```
CREATE DOMAIN University_Semester AS CHAR(6)
CHECK (VALUE IN (‘Spring’, ‘Summer’, ‘Fall’));    [1 mark]

CREATE DOMAIN Teaching_Evaluation AS DECIMAL(2, 1)
CHECK (VALUE ≥ 0.0 AND VALUE ≤ 4.0);    [1 mark]

CREATE TABLE Teach (
        StaffID INTEGER,
        CourseID INTEGER,
        Semester University_Semester,
        Evaluation Teaching_Evaluation,
        PRIMARY KEY (StaffID, CourseID, Semester),    [3 marks]
        FOREIGN KEY StaffID REFERENCES Staff(ID) ON DELETE CASCADE,    [2 marks]
        FOREIGN KEY CourseID REFERENCES Course(ID) ON DELETE CASCADE    [2 marks]
);
```

// SQL statements for creating the Enrolment table:

```
CREATE TABLE Enrolment (
        StudentID INTEGER,
        CourseID INTEGER,
        Grade Student_Grade,
        PRIMARY KEY (StudentID, CourseID),    [2 marks]
        FOREIGN KEY StudentID REFERENCES Student(ID) ON DELETE CASCADE, [2 marks]
        FOREIGN KEY CourseID REFERENCES Course(ID) ON DELETE RESTRICT    [2 marks]
);
```

Correct data types (or domains) for all attributes in the five tables   [2 marks]

Provide your answers for Question B on this page.

// Answers for Question B(1)

Consider size-1 attribute sets: [1 mark]

$A^+ = A$ $B^+ = BD$ $C^+ = C$ $D^+ = D$ $E^+ = ABCDEF$ $F^+ = F$

E is the size-1 candidate key for R.

Consider size-2 attribute sets (E is excluded from the expansion): [4 marks]

$(AB)^+ = ABCDEF$ $(AC)^+ = ABCDEF$ $(AD)^+ = ABCDEF$ $(AF)^+ = AF$

$(BC)^+ = ABCDEF$ $(BD)^+ = BD$ $(BF)^+ = BDF$

$(CD)^+ = CD$ $(CF)^+ = CF$

$(DF)^+ = DF$

AB, AC, AD, BC are the size-2 candidate keys for R.

Consider size-3 attribute sets (E, AB, AC, AD, BC are excluded from the expansion):

$(BDF)^+ = BDF$ $(CDF)^+ = CDF$

There is no size-3 candidate key for R. Besides, no more checking is needed for attribute sets with size > 3.

Therefore, the candidate keys for R are E, AB, AC, AD, BC [5 marks]

// Answers for Question B(2)

R(A, B, C, D, E, F, G, H) $\mathcal{F} = \{A \to CD, C \to EF, G \to A, CE \to F, BG \to H\}$ CK: BG

R violates 2NF because of the partial dependency $G \to A$

Hence, the highest normal form of R is 1NF [2 marks]

Decompose R into: [3 marks]

R_1(B, G, H) $\mathcal{F} = \{BG \to H\}$ CK: BG R_1 is in BCNF

R_2(A, C, D, E, F, G) $\mathcal{F} = \{A \to CD, C \to EF, G \to A, CE \to F\}$ CK: G

R_2 violates 3NF because of the transitive dependency $G \to A$ and $A \to CD$

Decompose R_2 into: [3 marks]

R_{21}(A, G) $\mathcal{F} = \{G \to A\}$ CK: G R_{21} is in BCNF

R_{22}(A, C, D, E, F) $\mathcal{F} = \{A \to CD, C \to EF, CE \to F\}$ CK: A

R_{22} violates 3NF because of the transitive dependency $A \to CD$ and $C \to EF$

Decompose R_{22} into: [3 marks]

R_{221}(A, C, D) $\mathcal{F} = \{A \to CD\}$ CK: A R_{221} is in BCNF

R_{222}(C, E, F) $\mathcal{F} = \{C \to EF, CE \to F\}$ CK: C R_{222} is in BCNF

The final set of BCNF relations are (B, G, H), (A, G), (A, C, D), (C, E, F) [4 marks]

Provide your answers for Question C on this page.

// Answers for Question C(1)

Record blocking factor (bfr) = block size / record size = 1000 / 50 = 20 records/block [1 mark]
file blocks (B) = # records / bfr = 6400 / 20 = 320 blocks [1 mark]
Total cost of linear scan = B = 320 block access [1 mark]

Binary search: Find the record with VID = 6000, and then retrieve all the matching records
Total cost of binary search = $\log_2$ B + # matching blocks
= ceiling ($\log_2$ 320) + 400 / 20 = 9 + 20 = 29 block access [3 marks]

// Answers for Question C(2)

Record blocking factor (bfr) = block size / record size = 1000 / 50 = 20 records/block [1 mark]
file blocks (B) = # records / bfr = 6400 / 20 = 320 blocks [1 mark]
index entries = B = 320 entries [1 mark]
Index entry size = 4 + 6 = 10 bytes [1 mark]
Index blocking factor (ibfr) = block size / index entry size = 1000 / 10 = 100 entries/block [1 mark]
index blocks (IB) = # index entries / ibfr = ceiling (320 / 100) = 4 blocks [1 mark]

Search using primary index: Find the record with VID = 6000 using the primary index, and then retrieve all the matching records
Index access = $\log_2$ IB = $\log_2$ 4 = 2 block access [1 mark]
Data access = # matching blocks = 400 / 20 = 20 block access [1 mark]
Total cost = index access + data access = 2 + 20 = 22 block access [1 mark]

// Answers for Question C(3)

Used capacity of each tree node = block size * 60% = 1000 * 60% = 600 bytes [1 mark]
Index entry size = 4 + 6 = 10 bytes [1 mark]
Index blocking factor (ibfr) = 600 / 10 = 60 entries/block [1 mark]
index entries = # records = 6400 entries [1 mark]
Tree height = $\log_{ibfr}$ # index entries = ceiling ($\log_{60}$ 6400) = 3 [2 marks]

Search using B^+ tree index when the records are sorted by VID: Find the record with VID = 6000 using the B^+ tree index, and then retrieve all the matching records
Index access = tree height = 3 block access [1 mark]
Data access = # matching blocks = 400 / 20 = 20 block access [1 mark]
Total cost = index access + data access = 3 + 20 = 23 block access [1 mark]

Search using B^+ tree index when the records are not sorted by VID: Find the leaf node with index entry VID = 6000 using the B^+ tree index, and then retrieve all the matching leaf nodes. For each index entry in these leaf nodes, retrieve the corresponding record from the file
matching leaf nodes = ceiling (400 / 60) = 7 blocks [1 mark]
Index access = tree height + # matching leaf nodes = 3 + 7 = 10 block access [1 mark]
Data access = # matching records = 400 block access [1 mark]
Total cost = index access + data access = 10 + 400 = 410 block access [1 mark]

// Answers for Question C(4)

Bitmap size = # records = 6400 bits = 800 bytes [2 marks]
Index blocking factor (ibfr) = block size / bitmap size = floor (1000 / 800) = 1 bitmap/block [1 mark]
blocks for *Maker* bitmaps = # bitmaps / ibfr = # distinct values / ibfr = 80 blocks [2 marks]
blocks for *Color* bitmaps = # bitmaps / ibfr = # distinct values / ibfr = 10 blocks [2 marks]
Total storage cost = 80 + 10 = 90 blocks [1 mark]

Search using bitmap index: Obtain the bitmap for 'Toyota' and the bitmap for 'Black'; Intersect the bitmaps to get the row-ids; Retrieve the corresponding records based on row-ids
Obtain the bitmap for 'Toyota' = 1 block access [1 mark]
Obtain the bitmap for 'Black' = 1 block access [1 mark]
Intersect the bitmaps to get the 40 row-ids = 0 block access
Retrieve the 40 records = 40 block access [1 mark]
Total cost = index access + data access = 1 + 1 + 40 = 42 block access [1 mark]