نام و نام خانوادگی: مبین خیبری

شماره دانشجویی: ۹۹۴۴۲۱۰۱۷

پاسخ تمرینات سری اول درس پایگاه داده

(1.1) Two disadvantages associated with database systems are listed below.

* setup of the database systems requires more knowledge, money, skills and time.
* The complexity of database may result in poor performance.

(1.7)

1. A file-processing system is more specific to the problem at hand while a DBMS is more general. A file-processing system used by a university is difficult to use in a hospital setting. While a DBMS once written can be used in different places.

2. It is difficult to ensure atomicity in a conversational file-processing system while it is a lot easier in a DBMS. often wrapping a set of SQL statements in a "BEGIN TRANSACTIONS" are often enough in the relational DBMS world.

3. Protecting against concurrent-access anomalies in a file-processing system is difficult. using a DBMS is much easier to protect against concurrent-access anomalies.

4. most DBMS have a concept of a user and what access that user has. Enforcing such authorization in a file-processing system is really difficult.

(1.9)

1. Security - Since DBMS have the concept of a ROLE (user) it easier for setting access management.

2. Needs to offer atomicity when needed - If atomicity is not provided, inconsistency will be inevitable.

3. Needs to offer a simple and efficient way to query data.

4. Needs to offer durability i.e. once an update or an insert has happened it must be persisted.

5. A DBMS needs to offer a way for protecting against concurrent-access anomalies.

(1.12)

* Earlier-generation database applications used a two-tier architecture, where as a three-tier architecture is used by a modern database application.

* In a two-tier architecture the application resides at the client machine, and invokes database system functionality at the server machine through query language statements. In a three-tier architecture the client machine acts as merely a front end and does not contain any direct database calls; the front end communicates with an application server. The application server, in turn, communicates with a database system to access data.

* Three-tier applications provide better security as well as better performance than two-tier applications.

Note: Even though the book classifies database applications in two, the reality is that most famous applications use four-tier architecture. If we take most chatting mobile applications they are a three-tier architecture with a local database such as sqLite for caching the data and accessing it when the mobile is not connected to the internet.

(2.5) The result attributes include all attribute values of student followed by all attributes of advisor. The tuples in the result are as follows: For each student who has an advisor, the result has a row containing the student's attributes, followed by s_id attribute identical to the student's ID attribute, followed by the i_id attribute containing the ID of the student advisor.

students who do not have an advisor will not appear in the result. A student who has more than one advisor will appear a corresponding number of times in the result.

(2.6) a. $\Pi_{personname}(\sigma_{city = "miami"}(employee))$

b. $\Pi_{personname}(\sigma_{salary > 100000}(employee \bowtie works))$

c. $\Pi_{personname}(\sigma_{salary > 100000 \wedge city = "miami"}(employee \bowtie works))$

(2.7)

a. $\Pi_{branchname}(\sigma_{branch\ city = "chicago"}(branch))$

b. $\Pi_{ID}(\sigma_{branch\_name = 'Downtown' \wedge loan.loan\_name = borrower.loan\_number}(borrower))$

(2.12) a.

Relation Name	Primary Key
branch	branch-name
customer	ID
loan	loan-number
borrower	{ID, loan-number}
account	account-number
depositor	{ID, account-number}

b.

Relation	Foreign Key
branch	No foreign key
customer	No foreign key
loan	branch-name
borrower	ID - a foreign key referencing customer relation, loan-number - a foreign key referencing loan relation
account	branch-name
depositor	ID - a foreign key referencing customer relation, account number - a foreign key referencing account relation.

(2.14)

a. $\Pi_{ID, personname}(employee \bowtie employee.ID = works.ID(\sigma_{company\ name = "Big\ Bank"}(works))$

b. $\Pi_{ID, personname, city}(employee \bowtie employee.ID = works.ID(\sigma_{company\ name = "BigBank"}(works))$

c. $\Pi_{ID, personname, street, city}(employee \bowtie employee.ID = works.ID(\sigma_{company\ name = "BigBank"}(works))$

d. $\Pi_{ID, personname}(employee \bowtie employee.ID = works.ID\ works \bowtie works.Company\ name = Company.Company\ name \wedge Company\ city = Company\ city\ Company)$

(2.15)

1. When a value of an attribute is unknown.
2. When a value of an attribute does not exist.

(2.18)

a. $\Pi_{ID, name}(\sigma_{dept\_name = "Physical"}(instructor))$

b. $\Pi_{ID, name}(instructor \bowtie instructor.dept\_name = department.dept\_name(\sigma_{building = "Watson"}(department)))$

c. $\Pi_{student.ID, student.name}(\sigma_{dept\_name = "Comp. Sci"}(student \bowtie student.ID = takes.ID\ takes \bowtie takes.course\_id = course.course\_id\ course))$

d. $\Pi_{student.ID, student.name}(\sigma_{year = 2018}(student \bowtie student.ID = takes.ID\ takes))$

e. $\Pi_{ID, name}(student) - \Pi_{student.ID, student.name}(\sigma_{year = 2018}(student \bowtie student.ID = takes.ID\ takes))$

(3.9)

a.
```
SELECT e.ID, e.person_name, city
FROM employee AS e, works AS w
WHERE w.company_name = 'First Bank Corporation' AND w.ID = e.ID
```

b.
```
SELECT ID, name, city
FROM employee
WHERE ID IN (
    SELECT ID
    FROM works
    WHERE company_name = 'First Bank Corporation' AND salary > 10000
)
```

another solution:
```
SELECT e.ID, e.person_name, city
FROM employee AS e, works AS w
WHERE w.company_name = 'First Bank Corporation' AND w.ID, e.ID
  AND w.salary > 10000
```

c.
```
SELECT ID
FROM works
WHERE company_name <> 'First Bank Corporation'
```

another solution:
```
SELECT ID
FROM employee
WHERE ID NOT IN (
    SELECT ID
    FROM works
    WHERE company_name = 'First Bank Corporation'
)
```

d.
```
SELECT ID
FROM works
WHERE salary > ALL (
    SELECT salary
    FROM works
    WHERE company_name = 'Small Bank Corporation'
)
```

(3.9) e.
```
SELECT S.company_name
FROM company AS S
WHERE NOT EXISTS (
    ( SELECT city
      FROM company
      WHERE company_name = 'Small Bank Corporation'
    )
  EXCEPT
    ( SELECT city
      FROM company AS T
      WHERE T.company_name = S.company_name
    )
)
```

f.
```
SELECT company_name
FROM works
GROUP BY company_name
HAVING COUNT(DISTINCT ID) >= ALL (
    SELECT COUNT (DISTINCT ID)
    FROM Works
    GROUP BY company_name
)
```

g.
```
SELECT company_name
FROM works
GROUP BY company_name
HAVING AVG (salary) > (
    SELECT AVG(salary)
    FROM works
    WHERE company_name = 'First Bank Corporation'
)
```

(3.12)

a.
```
INSERT INTO course (course_id, title, dept_name, credits)
VALUES ('CS-001', 'Weekly Seminar', 'Comp. Sci.', 0);
```

b.
```
INSERT INTO section (course_id, sec_id, semester, year) VALUES
('CS-001', '1', 'Fall', 2017)
```

c.
```
INSERT INTO takes (ID, course_id, sec_id, semester, year)
SELECT student.ID, 'CS-001', '1', 'Fall', 2017
FROM student
WHERE student.dept_name = 'Comp. Sci.'
```

(3.12)

e. DELETE FROM course
WHERE course_id = 'CS-001';

```
create table section
    ( course_id          varchar(8),
      sec_id             varchar(8),
      semester           varchar(6),
                 check (semester in ('Fall', 'Winter', 'Spring', 'Summer')),
      year               numeric(4,0) check (year > 1701 and year < 2100),
      building           varchar(15),
      room_number        varchar(7),
      time_slot_id       varchar(4),
      primary key  (course_id, sec_id, semester, year),
      foreign key  (course_id) references course (course_id)
                   on delete cascades
      foreign key  (building, room_number) references classroom (building, room_number)
                   on delete set null
    );
```

f. DELETE FROM takes
WHERE course_id IN (
SELECT course_id
FROM course
WHERE LOWER(title) LIKE '%advanced%'
)

(3.15)

a. WITH all_branches_in_brooklyn (branch_name) AS (
SELECT branch_name
FROM branch
WHERE branch_city = 'Brooklyn'
)
SELECT ID, customer_name
FROM customer AS C1
WHERE NOT EXISTS (
(SELECT branch_name FROM all_branches_in_brooklyn)
EXCEPT
(
SELECT branch_name
FROM account INNER JOIN depositor
ON account_account_number = depositor.account_number
WHERE depositor.ID = C1.ID
)
)

b. SELECT SUM(amount)
FROM loan

(3.14)

```
SELECT branch-name
FROM branch
WHERE assets > some (
 SELECT assets
 FROM branch
 WHERE branch-city = 'Brooklyn'
);
```

(3.16)

a.
```
SELECT employee.id, employee.person-name
FROM employee INNER JOIN works on employee.id = works.id
              INNER JOIN company ON works.company-name = company.company-name
WHERE employee.city = company.city
```

b.
```
SELECT E.id, E.person_name
FROM employee AS E INNER JOIN messages ON E.id = manages.id
                   INNER JOIN employee AS manager_of_E ON manages.manager_id = manager_of_E.id
WHERE E.street = manager_of_E.street AND
      E.city = manager_of_E.city;
```

c.
```
WITH average-salary-per-company (company-name, avg-salary) AS (
    SELECT company-name, AVG(salary)
    FROM works
    GROUP BY company-name
)
SELECT E.id , E.person-name
FROM employee AS E INNER JOIN works ON E.id = works.id
WHERE works.salary > (
    SELECT avg-salary
    FROM average-salary-per-company
    WHERE company-name = works.company-name
)
```

d.
```
SELECT company-name, SUM(salary) AS total-payroll
FROM works
GROUP BY company-name
ORDER BY total-payroll AS (
LIMIT 1
```

(3.22)

```
WHERE UNIQUE (SELECT title FROM course)
```

```
WHERE 1 >= ALL (
   SELECT COUNT(*)
   FROM course
   GROUP BY title
)
```

```
WHERE NOT EXISTS (
    SELECT *
    FROM course AS c1, course AS c2
    WHERE c1.course-id != c2.course-id AND
    c1.title = c2.title
)
```

(3.23)
```
SELECT dept_name
FROM (SELECT dept_name, sum(salary) As value FROM instructor GROUP BY dept_name) AS dept_total,
    (SELECT AVG(value) AS value FROM(SELECT dept_name, sum(salary) AS value FROM instructor
    GROUP BY dept_name) As x) As dept_total_avg
WHERE dept_total.value >= dept_total_avg.value
```

(3.25)
```
SELECT dept_name
FROM department
WHERE budget > (SELECT budget FROM department WHERE dept_name = 'philosophy')
ORDER BY dept_name ASC;
```

(3.28)
```
SELECT id, name
FROM instructor As i
WHERE NOT EXISTS (
  (SELECT course_id FROM course WHERE dept_name = i.dept_name)
  EXCEPT
  (SELECT course_id FROM teaches WHERE teaches.id = i.id)
)
ORDER BY name ASC
```

(3.37)
```
SELECT id, name
FROM instructor As i
WHERE 'A' NOT IN (
  SELECT takes.grade
  FROM takes INNER JOIN teaches
    ON (takes.course_id, takes.sec_id, takes.semester, takes.year) =
      (teaches.course_id, teaches.sec_id, teaches.semester, teaches.year)
  WHERE teaches.id = i.id
)
```