CSE 462 Homework #2: SQL

Name: ______________________________ Date: February 14, 2011

***** Due on February 23th, 2011 at the beginning of class. *****

Instructions. This problem set is worth 200 points. For each problem below, formulate the specified queries and/or create the required views. You must only use operators covered in class, but you may create any additional views you deem necessary to help you solve a problem. As a suggestion, create the relations with the given schemas in Oracle (or some other database), insert some test data, and verify your answers.

Problem 1. [20 points] Let `R(A, B)` and `S(A, B)` be two relations and assume that: (i) their attributes have the same type; (ii) both relations have `NOT NULL` check constraints on `B`.

For each pair of queries below, determine whether they are equivalent. Remember that two queries are equivalent if they compute the same set of answers *for every database instance*. You get 4 points for each correct answer, -2 points for each incorrect one, and 0 points for answers left in blank.

	Query #1	Query #2	equivalent?
a)	`SELECT B FROM R WHERE A IN (SELECT A FROM S);`	`SELECT R.B FROM R INNER JOIN S ON (R.A = S.A);`	YES
b)	`SELECT A FROM R WHERE A NOT IN (SELECT A FROM S);`	`SELECT DISTINCT R.A FROM R INNER JOIN S ON (R.A <> S.A);`	NO
c)	`SELECT DISTINCT R.A, R.B FROM R CROSS JOIN R AS R1;`	`SELECT A, B FROM R GROUP BY A, B;`	YES
d)	`SELECT R.A FROM R;`	`SELECT A FROM R GROUP BY A, B;`	YES
e)	`SELECT MIN(B) AS B FROM R;`	`SELECT R.B FROM R WHERE R.B < ALL (SELECT B FROM R);`	NO

Problem 2. [35 points] Consider a car dealership that keeps track of all information their in-stock cars using the relation `Cars(make, model, year, color, price)`.

a) [10 points] A query that returns makes having some model with at least two cars of different colors.

```
SELECT make
FROM Cars
WHERE model IN
      (SELECT C1.model
       FROM Cars C1
       JOIN Cars C2 ON (C1.make=C2.make AND C1.model=C2.model AND C1.color <> C2.color));
```

b) [15 points] Two views: one named `Oldest` returns all attributes of the oldest cars and the other named `ExpensiveOldest` returns all attributes of the most expensive amongst the oldest cars. **Hint:** Use the first view to define the second one.

```
CREATE VIEW Oldest AS
  SELECT make, model, year, color, price
  FROM Cars
  WHERE year = (SELECT MIN(year) FROM Cars);

CREATE VIEW ExpensiveOldest AS
  SELECT make, model, year, color, price
  FROM Oldest
  WHERE price = (SELECT MAX(price) FROM Oldest);
```

c) [10 points] A query that returns the year(s) with the largest number of cars of the same make and model.

```
CREATE VIEW Stock AS
  SELECT make, model, year, COUNT(*) AS available
  FROM Cars GROUP BY make, model, year;

SELECT year FROM Stock WHERE available = (SELECT MAX(available) FROM Stock);
```

Problem 3. [35 points] Consider relations `R(A)` and `S(A, B)` where all attributes have non-NULL values.

a) [10 points] The *median* of a set is the value such that half the numbers in the set are larger and half are smaller. For this problem only, assume that `A` is a key and `R` contains an odd number of tuples. Write a query that returns the *median* of the values in `R`.

```
SELECT A FROM R AS M
WHERE (SELECT COUNT(*) FROM R WHERE A > M.A) = (SELECT COUNT(*) FROM R WHERE A < M.A);
```

b) [10 points] The *mode* of a list is the value that occurs most often. Write a query that returns the *mode* of the values in `R`.

```
SELECT A FROM R GROUP BY A
HAVING COUNT(*) = (SELECT MAX(Dups) AS ModeCount
                   FROM (SELECT A, COUNT(*) AS Dups FROM R GROUP BY A) AS M);
```

c) [15 points] Write a query that associates an integer rank to each tuple in `S` based on the value of `A` alone as follows: assign the rank $k+1$ to all tuples (a,b) such that there exists exactly k tuples in `S` with their `A` value larger than a.

```
-- COUNT(*) would cause the count of the top value to be off by one
-- inner join would cause the top value not to show
SELECT S1.A, S1.B, COUNT(S2.A) + 1 AS Rank
FROM S AS S1 LEFT JOIN S AS S2 ON (S1.A < S2.A)
GROUP BY S1.A, S1.B;
```

Problem 4. [40 points] Consider a small financial application that keeps track of credit and debit operations of their customers' accounts using relations `Credits(opId, accountId, date, value)` and `Debits(opId, accountId, date, value)`. Attribute `opId` is a unique operation identifier, `accountId` is the identifier of the customer account on which the operation is performed, `date` determines the processing date for the operation, and `value` indicates the value of the operation. Assume that all attributes are non-NULL and that `value` is greater than zero. **Attention:** (i) any account may have any number of credits and debits on any date, and (ii) there may exist dates in which only credits or only debits exist for an account.

a) [25 points] A view `DailyNetResult` that returns, for each account and date, the *daily net result* of that account on that date. The *daily net result* is defined as the sum of all credits minus the sum of all debits on the particular account and date.

```
CREATE VIEW CreditsByDate(accountId, date, credit) AS
  SELECT accountId, date, SUM(value) FROM Credits GROUP BY accountId, date;

CREATE VIEW DebitsByDate(accountId, date, debit) AS
  SELECT accountId, date, SUM(value) FROM Debits GROUP BY accountId, date;

CREATE VIEW DailyNetResult(accountId, date, value) AS
  SELECT
    COALESCE(C.accountId, D.accountId),
    COALESCE(C.date, D.date),
    COALESCE(credit, 0.0) - COALESCE(debit, 0.0)
  FROM CreditsByDate C NATURAL FULL OUTER JOIN DebitsByDate D;
```

b) [15 points] A view `DailyBalance` that returns, for each account and date, the sum of all daily net results of that account on all dates up to and including that date.

```
CREATE VIEW DailyBalance(accountId, date, balance) AS
  SELECT DNR1.accountId, DNR1.date, SUM(DNR2.value)
  FROM DailyNetResult DNR1
  INNER JOIN DailyNetResult DNR2 ON (DNR1.accountId = DNR2.accountId AND DNR1.date >= DNR2.date)
  GROUP BY DNR1.accountId, DNR1.date;
```

Problem 5. [70 points] A scientific conference maintains a web site that provides an interface for users to query information about their publications and authors, which is stored in relations `Publications(pId, title, year)`, `Authors(aId, lastName, firstName)`, and `Authorship(pId, aId)`. Attribute `pId` is the publication identifier and `aId` the author identifier. Assume that `{lastName, firstName}` is also a key in `Authors`.

a) [10 points] A view `Coauthors` that returns distinct pairs of author identifiers representing authors who co-authored some publicaton. With the pair of authors, also return the publication which they co-authored and the year of the publication. If your query returns tuple `(aId1, aId2, pId, y)`, it must not return `(aId2, aId1, pId, y)`.

```
CREATE VIEW Coauthors(aId1, aId2, pId, year) AS
  SELECT A.aId, B.aId, A.pId, P.year
  FROM Authorship A
  INNER JOIN Authorship B ON (A.pId = B.pId AND A.aId < B.aId)
  INNER JOIN Publication P ON (A.pId = P.pId);
```

b) [20 points] A view `StableCoauthors` that returns distinct pairs of author identifiers representing authors who co-authored some publicaton in every year for the past 10 years (that is, 2001-2010). If your query returns the pair `(aId1, aId2)`, it must not return the pair `(aId2, aId1)`. Use the `Coauthors` view from (a) to simplify your view.

```
CREATE VIEW CoauthorsMissingYear(aId1, aId2, year) AS
  (SELECT aId1, aId2, P.year FROM Coauthors, Publications P WHERE 2001 <= P.year AND P.year <= 2010)
  EXCEPT
  (SELECT aId1, aId2, year FROM Coauthors);

CREATE VIEW StableCoauthors(aId1, aId2) AS
  (SELECT aId1, aId2 FROM Coauthors WHERE 2001 <= year AND year <= 2010)
  EXCEPT
  (SELECT aId1, aId2 FROM CoauthorsMissingYear);
```

c) [25 points] A view `NoExperiencedCoauthors` that returns the identifier, last and first names of authors who never co-authored with a more experienced author, that is, one who at the time of their co-authorship had published more than they had. **Hint:** Create a helper view that associates, for each author and year, the total number of publications; then, create another view that extends the co-authors view by associating this total to each author.

```
CREATE VIEW PublicationsByYear(aId, year, total) AS
  SELECT aId, year, COUNT(pId) AS total
  FROM Authorship A INNER JOIN Publication P ON (A.pId = P.pId)
  GROUP BY aId, year;

CREATE VIEW CPublicationsByYear(aId, year, ctotal) AS
  SELECT PY1.aId, PY1.year, SUM(PY2.total)
  FROM PublicationsByYear AS PY1
  INNER JOIN PublicationsByYear AS PY2 ON (PY1.aId = PY2.aId AND PY1.year > PY2.year) AS total
  GROUP BY PY1.aId, PY1.year;

CREATE VIEW CumulativeCoauthors(aId1, aId2, year, ctotal1, ctotal2) AS
  SELECT CA.aId1, CA.aId2, CA.year, PY1.ctotal, PY2.ctotal
  FROM Coauthors CA
  INNER JOIN CPublicationsByYear PY1 ON (PY1.aId = CA.aId1 AND PY1.year = CA.year)
  INNER JOIN CPublicationsByYear PY2 ON (PY2.aId = CA.aId2 AND PY2.year = CA.year)


CREATE VIEW NoExperiencedCoauthors(aId, lastName, firstName) AS
  SELECT aId, lastName, firstName
  FROM Authors A WHERE NOT EXISTS
    (SELECT * FROM CumulativeCoauthors
     WHERE (aId1 = A.aId AND ctotal1 < ctotal2) OR (aId2 = A.aId AND ctotal2 < ctotal1));
```

d) [15 points] A view `TopAuthors` that returns the identifier, last and first names of authors who have the three largest publication counts. **Hint:** Define helper views to return the largest, second largest, and third largest numbers of publications.

```
CREATE VIEW PublicationsT1(total) AS
```

```
  SELECT MAX(pubs)
  FROM (SELECT COUNT(pId) AS pubs FROM Authorship GROUP BY aId) AS C;

CREATE VIEW PublicationsT2(total) AS
  SELECT MAX(pubs)
  FROM (SELECT COUNT(pId) AS pubs FROM Authorship
        GROUP BY aId HAVING COUNT(pId) < (SELECT total FROM PublicationsT1)) AS C;

CREATE VIEW PublicationsT3(total) AS
  SELECT MAX(pubs)
  FROM (SELECT COUNT(pId) AS pubs FROM Authorship
        GROUP BY aId HAVING COUNT(pId) < (SELECT total FROM PublicationsT2)) AS C;

CREATE VIEW TopAuthors(aId, lastName, firstName) AS
  SELECT aId, lastName, firstName
  FROM Authors A WHERE aId IN
    (SELECT aId FROM Authorship IA GROUP BY aId
     HAVING COUNT(pId) >= (SELECT total FROM PulicationsT3));
```