SQL Practicum – 100 points total **Name____________________________**

1. (10 pts) Write a query that shows all attributes of P_DISEASE, sort by ascending TRANSMISSION_MODE as the primary sort key and by descending DEATHS_WORLD as the secondary sort key.

```
SELECT *
FROM P_DISEASE
ORDER BY TRANSMISSION_MODE ASC, DEATHS_WORLD DESC;
```

2. (5 pts) Write a statement that changes the value for GENDER from 'U' to 'M' for the person with first and last names of 'Sam' and 'Spade' respectively. It is known that there is only one person with the name Sam Spade in the database.

```
UPDATE P_PERSON
SET GENDER = 'M'
WHERE FIRSTNAME = 'Sam'
AND LASTNAME = 'Spade';
```

3. (5 pts) Write a statement that would insert the first record shown of the P_PREVIOUS_NAMES table if it was not in the database already. The first record is shown below.

```
PERSON_ID NAME_NUM FIRSTNAME MIDDLE_NAME_STRING LASTNAME
--------- -------- --------- ------------------ -----------------
10001            1 Ludmilla  (null)             Smith
```

```
Insert into P_ PREVIOUS_NAMES
(PERSON_ID,NAME_NUM,FIRSTNAME,MIDDLE_NAME_STRING,LASTNAME) values
('10022',1,'Ludmilla',null,'Smith');
-or-
Insert into P_ PREVIOUS_NAMES (PERSON_ID,NAME_NUM,FIRSTNAME, LASTNAME)
values ('10022',1,'Ludmilla', 'Smith');
```

4. (10 pts) Write a query that returns all the attributes of the P_PROVINCE table when *any* of the following conditions are true: 1. the population is between 500,000 and 1,500,000 inclusive; 2. the POST_ABBREV begins with an 'N' as the first character; 3. the AREA_WATER is less than 25,000.

```
SELECT *
FROM P_PROVINCE
WHERE (POPULATION BETWEEN 500000 AND 1500000)
OR (POST_ABBREV LIKE ('N%'))
OR (AREA_WATER < 25000);
```

5. (10 pts) Show each PROVINCE_NAME in P_PROVINCE with a count of the number of UNIQUE ADDRESSES associated with each province. Alias the count appropriately.

```
SELECT PROVINCE_NAME, COUNT(DISTINCT ADDRESS_ID) AS ADDRESS_COUNT
FROM P_PROVINCE P, P_ADDRESS A
WHERE P.PROVINCE_ID = A.PROVINCE_ID
GROUP BY P.PROVINCE_ID, PROVINCE_NAME;
```

6. (10 pts) Use a correlated subquery to show the PERSON_ID, LASTNAME, and FIRSTNAME from the P_PERSON table that also has a record in the P_PREVIOUS_NAMES table.

```
SELECT PERSON_ID, LASTNAME, FIRSTNAME
FROM P_PERSON P
WHERE EXISTS
     (SELECT *
     FROM P_PREVIOUS_NAMES PN
     WHERE P.PERSON_ID = PN.PERSON_ID);
```

7. (10 pts) For each illness event in the database, show its associated values for: LASTNAME, FIRSTNAME, DATE_ACTIVATED, and PROVINCE_NAME.

```
SELECT P.LASTNAME, P.FIRSTNAME, I.DATE_ACTIVATED, PROVINCE_NAME
FROM P_ILLNESS_EVENT I, P_PERSON P, P_PERSON_ADDRESS_INT PA,
P_ADDRESS A, P_PROVINCE PR
WHERE I.PERSON_ID = P.PERSON_ID
AND P.PERSON_ID = PA.PERSON_ID
AND PA.ADDRESS_ID = A.ADDRESS_ID
AND A.PROVINCE_ID = PR.PROVINCE_ID;
```

8. (10 pts) Write a query that shows any *disease* that is associated with *every* person in the P_PERSON table. This would mean that every person has at least one illness event that is associated with that disease. Again, find any *diseases* that are associated with every *person* – do ***not*** write the query to find any *person* that has had every *disease* (that is a different query). Show DISEASE_ID, DISEASE_NAME.

```
SELECT DISEASE_ID, DISEASE_NAME
FROM P_DISEASE D
WHERE NOT EXISTS
     (SELECT *
     FROM P_PERSON P
     WHERE NOT EXISTS
          (SELECT *
          FROM P_ILLNESS_EVENT I
          WHERE D.DISEASE_ID = I.DISEASE_ID
          AND I.PERSON_ID = P.PERSON_ID));
```

9. (10 pts) In the database, values for P_PERSON.PERSON_ID always begin with a '1' and are always five characters long. Write a statement to add a constraint to the existing P_PERSON table (assume it has been appropriately created previously – except for this constraint). The constraint should verify that all values entered into the PERSON_ID field begin with a '1' and have *exactly* 5 characters. You *do not* have to make sure each character is a number.

```
ALTER TABLE P_PERSON ADD
CONSTRAINT P_PERSON_FORMAT_CHK CHECK
(PERSON_ID LIKE '1____');
```

10. (10 pts) Write a query that lists out each person in the P_PERSON table (whether or not they have any previous names in the database) along with each previous name they are associated with. Write the query so that the person's information is shown even if they have no previous names entered into the database (the PREVIOUS_LASTNAME and PREVIOUS_FIRSTNAME attributes would be NULL for these records). Show PERSON_ID, FIRSTNAME, LASTNAME, PREVIOUS_FIRSTNAME, AND PREVIOUS_LASTNAME. PREVIOUS_FIRSTNAME is aliased from the FIRSTNAME attribute from the P_PREVIOUS_NAMES table. Likewise, PREVIOUS_LASTNAME is aliased from the LASTNAME attribute from the P_PREVIOUS_NAMES table.

```
SELECT P.PERSON_ID, P.FIRSTNAME, P.LASTNAME, PN.FIRSTNAME AS
PREVIOUS_FIRSTNAME, PN.LASTNAME AS PREVIOUS_LASTNAME
FROM P_PERSON P LEFT JOIN P_PREVIOUS_NAMES PN
ON P.PERSON_ID = PN.PERSON_ID;
```

11. (10 pts) Write the SQL statement needed to create the table P_PREVIOUS_NAMES. Include the primary key constraint and any alternate key and foreign key constraints necessary.

```
CREATE TABLE P_PREVIOUS_NAMES
   ( PERSON_ID CHAR(5) NOT NULL,
     NAME_NUM NUMBER(3,0) DEFAULT 1 NOT NULL,
     FIRSTNAME VARCHAR2(20),
     MIDDLE_NAME_STRING VARCHAR2(50),
     LASTNAME VARCHAR2(50),
CONSTRAINT P_PREVIOUS_NAMES_PK PRIMARY KEY (PERSON_ID, NAME_NUM),
CONSTRAINT P_PREVIOUS_NAMES_UK1 UNIQUE (PERSON_ID, FIRSTNAME,
MIDDLE_NAME_STRING, LASTNAME),
CONSTRAINT P_PREVIOUS_NAMES_FK1 FOREIGN KEY (PERSON_ID)
       REFERENCES P_PERSON (PERSON_ID));
```