C10. SQL Set Operations: Union, Intersect, Difference

The SQL set operations are derived from algebra set operations.

- Union (∪) – all tuples from R1 and all tuples from R2 (discards duplicates) – implemented by SQL UNION
- Intersect (∩) – tuples in both R1 and in R2, implemented by SQL INTERSECT
- Set difference (/) – tuples in R1, but not in R2, implemented by SQL MINUS

The two relations (R1, R2) must be union-compatible:

- have same number of fields
- corresponding field have the same type.

1.1. SQL UNION

Combines the results of two queries, and eliminates duplicate selected rows.

Relations have to be union compatible.

UNION ALL is used when is necessary to preserves duplicates.

The general syntax for union is:

```
SELECT r1.* FROM Relation1 r1 …

UNION [ALL]

SELECT r2.*  FROM Relation2 r2 …;
```

Union can be used to implement SQL FULL OUTER JOIN. This is special useful in system that do not support this operation.

```
SELECT * FROM T1
        LEFT JOIN T2 ON T1.id = T2.id
UNION
SELECT * FROM T1
        RIGHT JOIN T2 ON T1.id = T2.id;
```

Parameter ALL is not used in this context. This is due the necessity to avoid duplicated “inner join pairs”.

ACTIVITY 1: Using SQL Workshop -> SQL Commands run the following SQL command that selects all names of sailors and boats:

```
SELECT 'sailor', name FROM Sailors
UNION
SELECT 'boat', name FROM Boats;
```

Answer to the following question: how about using UNION ALL in this case?

ACTIVITY 2: Using SQL Workshop -> SQL Commands design an union query that extract a list of names of persons that work in department dep_id=1 and names of sailors that have rank = 1.

1.2. SQL Intersect

The INTERSECT operator returns only rows returned by both queries:

```
SELECT ProjectionList_X FROM Relation1 …
INTERSECT
SELECT ProjectionList_Y FROM Relation2 …;
```

- Relation1 and Relation2 must be “union compatible” (same number of fields and corresponding filed types on ProjectionList_X and ProjectionList_Y).

ACTIVITY 3: Using SQL Workshop -> SQL Commands run the following SQL command that find sid of sailors who have reserved both a red and a green boat:

```
SELECT s.sid
        FROM  Sailors s, Boats b, Reserves r
        WHERE s.sid=r.sid AND r.bid=b.bid
        AND b.color=‘Red’
INTERSECT
SELECT s.sid
        FROM  Sailors s, Boats b, Reserves r
        WHERE s.sid=r.sid AND r.bid=b.bid
        AND b.color=‘Green’;
```

ACTIVITY 4: Using SQL Workshop -> SQL Commands design a query that extract a list of sailors of the rank 7 that have namesakes of the rank 9 (namesake: someone that has the same name as another person).

1.3. Set Difference.

Implemented by the SQL MINUS (or SQL EXCEPT) operator.

Relations have to be union compatible.

It is not implemented by some systems (can be implemented using subquery with NOT IN operator). For example MySQL does not implement either of MINUS or EXCEPT.

The syntax for SQL MINUS is:

```
    SELECT ProjectionList_X
            FROM Relation1 …
    MINUS

            SELECT ProjectionList_X
                    FROM Relation2 …;
```

ACTIVITY 5: Using SQL Workshop -> SQL Commands run the following SQL command that extracts sailors who have reserved all boats:

```
SELECT s.name

        FROM Sailors s
        WHERE  NOT EXISTS
                ((SELECT b.bid
                                FROM  Boats b)
                  MINUS
                  SELECT r.bid
                                FROM Reserves r
                                WHERE r.sid=s.sid);
```

Note: to validate result please add first a reserve for boat bid=104 made by the sailor sid=58.

ACTIVITY 6: Create an interactive report based on a set operation query to display sailors who have reserved a Color1 boat but not a Color2 one. For reading the colors add two input fields (ifColor1 and ifColor2) and a validation button to the report.