1.

a) Map function:

```
// Input: transaction ID and a list of items in the transaction for every
transaction in the database (segmented into partitions for each map worker)
// Output: each subset of the itemset in every transaction along with the
transaction id
map( String tid, Item[] items ) {
   // Emit all subsets of items in transaction T
   for ( i = 0; i < items.length; i++ ) {
      EmitIntermediate( items[0 .. i], tid );
   }
}
```

b) Provider function:

```
provider() {
   HashMap<Item[],List<String>> map;

   for ( Entry<Item[] itemSet,String> entry in
getIntermediateItemEntries() ) { // Assumes a method to read intermediate
entries from file system
      if ( map.contains( entry.key ) {
         map.get( entry.key ).add( entry.value );
      } else {
         map.put( entry.key, new List<String>( entry.value );
      }
   }

   for( Entry<Item[], List<String> entry in map.entrySet() ) {
      reduce( entry.key, entry.value );
   }
}
```

c) Reduce function:

```
// Input: an item patern and a list of transactions that contain it
// Output: a count of the number of transactions containing the item set
//     - produces a set of itemsets and their frequency counts in the
database for all <=4 item sets
reduce( Item[] itemPattern, Iterator<String> tids ) {
   int count = 0;
   for ( tid in tids ) {
      count++;
   }

   Emit( itemPattern, count );
}
```

2.

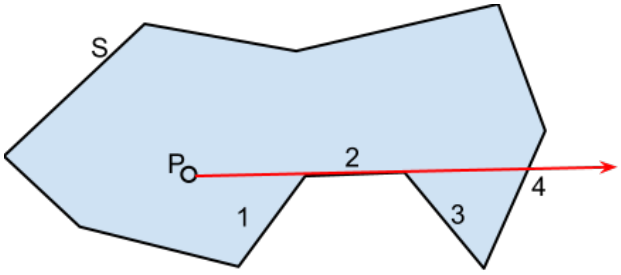
a)

```
SELECT C1.Name, C1.Area(), C2.Name C2.Area()
FROM Country C1, Country C2, River R
WHERE Touch( C1.Shape, C2.Shape ) = 1 AND
      Cross( C1.Shape, R.Shape ) = 1 AND
      Cross( C2.Shape, R.Shape ) = 1;
```

b)

```
SELECT C.Name
FROM Country C, Country US
WHERE US.Name = 'U.S.A.' AND
      C.Name <> US.Name  AND
      Distance( US.Shape, C.Shape ) <= 500;
```

3. Since the algorithm determines if the point is inside the bounds or not based on whether the number of vertexes crossed by a ray coming from the point to the outside of the figure being tested, counting edges whose vertex are tangential to the ray would result in skewed results for this simple test. For example if given the shape S and test point P:



The test will falsely report (due to the even number of intersections 1, 2, 3 and 4) that P does not lie within S. Omitting all line sections without endpoints stricly above P's ray leaves just one intersection (intersection 4) which will correctly report that P is inside S. In the case that the tangential section of S touched the top of P's ray, 3 intersections would be counted still reporting the correct status of P being inside R.

4.

a) If Z1's region completely contains Z2's region Z1's pattern is a prefix of Z2's pattern. Example: Z1 = 1***, Z2 = 10**

b) Since Z values are single valued attributes, the necessity for an R-Tree or an R*Tree goes away. The database can now be stored in a B-Tree similar to databases storing scalar data.

5.

a) N + M / P => Assumes that every row of both R and S must be checked in the query

b) I/O Time is slightly more dominant though with large in memory page sizes, the algorithm is relatively equally tied to I/O and CPU time

c)

i) spatialjoin2 cuts down on only CPU time because it trims out comparisons that may be made once they're already loaded into memory

ii) sorting improves CPU time becuase it leverages an ordering of objects to check for intersection to prevent redundant comparisons of objects already in memory

d) Plane sweep defines a sweep line and keeps track of which rectangles it's currently intersecting. At any given time newly discovered rectangles need only check for intersection with currently visited rectangles by the sweep line.

6.

a)
```
SELECT *
FROM (
   SELECT I2.photo_id, I2.photo, ORDImageSignature.evaluateScore(
I1.photo_sig, I2.photo_sig, 'color=1.0' ) Feature_Distance
```

```
      FROM stockphotos I1, stockphotos I2
      WHERE I1.photo_id='query_image.jpg' AND
            I1.photo_id <> I2.photo_id
      ORDER BY Feature_Distance
   )
   WHERE rownum <= 10;
```

b) Performance of these queries can be measured based on precision by testing how many relevant images are returned out of the number returned (in this case 10). Recall can also be used as a measurement of performance by testing how many relevant images are returned from a query like this out of the total number of relevant images to the query image in the database.

7.

a) The major differences between TSQL2 and the temporal datamodel explained in the Elmasri book are that in the Elmasri book, temporal elements are appended as individual extra columns for each set of records. In TSQL2 the temporal elements are stored as a single object that contains the valid times for the row of data its associated with. In the Elmasri model, each row of data has 4 extra columns VST, VET, TST, TET that indicate the period in which this set of records was valid. This leads to duplicated row data that TSQl2's model would not have if a set of information was invalidated and then revalidated at a later time.

b) The PERIOD keyword in TSQL2 could not be used to create a table from the Elmasri book because it would not reflect the transaction times for each of the records displayed. Only the VST and VET periods could be generated using a query like this.

c)

```
   1) SELECT P2.Name
      FROM Prescription( Name, Drug )(PERIOD) AS P1,
           Prescription( Name, Drug )(PERIOD) AS P2
      WHERE VALID( P1 ) = VALID( P2 ) AND
            P1.Name = "Melanie" AND
            P1.Name <> P2.Name AND
            P1.Drug = P2.Drug;
   2) SELECT P1.Name
      FROM Prescription( Name, Drug )(PERIOD) P1,
           Prescription( Name, Drug )(PERIOD) P2
      WHERE P1.Name = P2.Name AND
            P1.Drug <> P2.Drug AND
            VALID(P1) = PERIOD '[2008-01-20 - 2008-01-28]' AND
            TRANSACTION(P2).Start > DATE '2008-01-28';
```

8.

a)

course	Teacher T	Teacher O	Texts X	Texts P
c1	t1	o1	x1	p1
c1	t2	o2	x2	p2
c2	t1	o1	x3	p3
c2	t2	o2	null	null

b) There are redundancy problems because multiple teachers can be assigned to the same course. Also some courses have no text which adds null values to the database that take up space to store no information and all teacher and textbook information is duplicated multiple times depending on how many times each shows up in the set of courses.

c) Schema: course(cid), teacher(tid, oid), text(xid, pid), courseteacher(cid, tid), coursetext(cid, xid)

d) Nested relations by themselves will not solve the redundancy problem because a separate nested table would still be stored for each course. That means that if for example a teacher taught 2 courses, the teacher's information would be duplicated in 2 separate places in the database.

e) Using REF attributes would help resolve this redundnacy issue by issuing essentially pointers into the nested relations for the course object. These pointers would point back to a teacher relation where each teachers information need only be stored one time.

9.

a) Yes. It is necessary to have an inverse relationship defined in both classes becasue a relationship in one direction doesn't imply that there is a relationship in the opposite direction.

b) ODL always requires that both sides of an inverse relationship be defined to not only completely define the relationship between two objects, but also provide an integrity constraint between the two objects in the database.

c) class person
   relationship mother is a person
   relationship father is a person
   relationship children is a set {person}
   inverse of mother is children in person
   inverse of father is children in person
   inverse of children is parents

d) Yes, the mother relationship refers to a set of childen that the mother mothers as its inverse set relationship.

10.

a) I'm assuming that 'game' in this question means 'team' because a player leaving a team and a score being related to a game have nothing to do with eachother. Also, player scores for a game is represented in the EER diagram provied for HW5. Player score for a team is not represented in this diagram for the reason suggested in the question. Since players are relatively independant of teams, score information should be stored between a particular player and the game he or she is playing in at any given time via a relation referencing the Game and Player objects. To get a player score for a team, a new relation could be created called 'player_scores_for_team' with a derived attribute for the score calculated from the games that both the player and the team played in.

b) Employee type cannot be instantiated because it is an abstract superclass for persons employed by a team. It is marked as not final because it is intended to be extended by a concrete object type such as Player.

c) Team1Score and Team2Score are aggregate relationships that refer to sets of player scores. These scores are stored in a table consisting of PlayerScoreType objects. As a table of references to PlayerScoreType objects, SQL statements would have no access to the table, though somewhere in the database there needs to be a physical table that stores the records these tables refer to which plain SQL would have access to.

d) INSERT INTO PlayerScoreRefTable SELECT REF(ps) FROM PlayerScore PS WHERE PS.pid=1 -- inserts all scores from player with pid of 1 into the playerscores ref table

e) Count is the number of rows in the cursor being iterated. Its iterating over indexes instead of over table rows like you would traditionally do with a

cursor. DREF is dereferencing the referenced object in the current row of the iteration. It's getting the physical value from the storage of the REF object.

f)

i)

```
SELECT PST.COLUMN_VALUE.pid
FROM PlayerScoreTable PST
WHERE NOT EXISTS (
   SELECT *
   FROM Games G, TABLE( G.team1Score MULTISET UNION ALL G.team2Score )
PS
   WHERE PS.COLUMN_VALUE.score=G.getMaxScore() AND
         PS.COLUMN_VALUE.pid = PST.pid
);
```

ii)

```
SELECT PS.COLUMN_VALUE.player.pname
FROM Games G, TABLE( G.team1Score MULTISET UNION ALL G.team2Score ) PS,
TeamsTable TT
WHERE PS.COLUMN_VALUE.score=G.getMaxScore() AND
      TT.name=G.getWinner() AND
      TT.league.lid='A';
```

11. 3 Triggers would need to be implemented to handle updates, inserts, and deletes from the fact table.

Assume a table storing old avg grade information

SAVE(sno, oldCount, oldAvg)

```
CREATE TRIGGER UpdateAvgGrade
  AFTER UPDATE ON sc
  FOR EACH ROW
  DECLARE
     oldavg NUMBER;
     oldcount NUMBER;
     newavg NUMBER;
  BEGIN
     SELECT s.oldCount INTO oldcount FROM SAVE WHERE SAVE.sno = :NEW.sno;
     SELECT s.oldAvg INTO oldavg FROM SAVE WHERE SAVE.sno = :NEW.sno;

     newavg = ( oldcount * oldavg - :OLD.grade + :NEW.grade ) / oldcount

     update AvgGrades set avg=newavg where sno=:NEW.sno;

     update SAVE set oldAvg=newavg where sno=:NEW.sno;
  END;
```

12.

a) Minmax distance is the minimum of the maximum distances from the given query point to a particular object in the database. It is intended to provide an upper bound on the distance of the existence of a point from the query point in a given rectangle.

b) It is calculated to find the closest defining vertex of the rectangular object storing points in the database. Since these rectangles are tightly bound on the points they contain, finding the minimum defining vertex guarantees the existence of a point between the query point and the point defining the MinMax distance.

13.

a) Read Uncommitted - W1 writes A, R2 reads A before C1 commmits A

b) Repeatable Read - There is a chance that R1's query for resource A may have applied to resource B when W2 wrote it before C1 committed. This could cause a phantom read which prevents this scenario from being serialized.