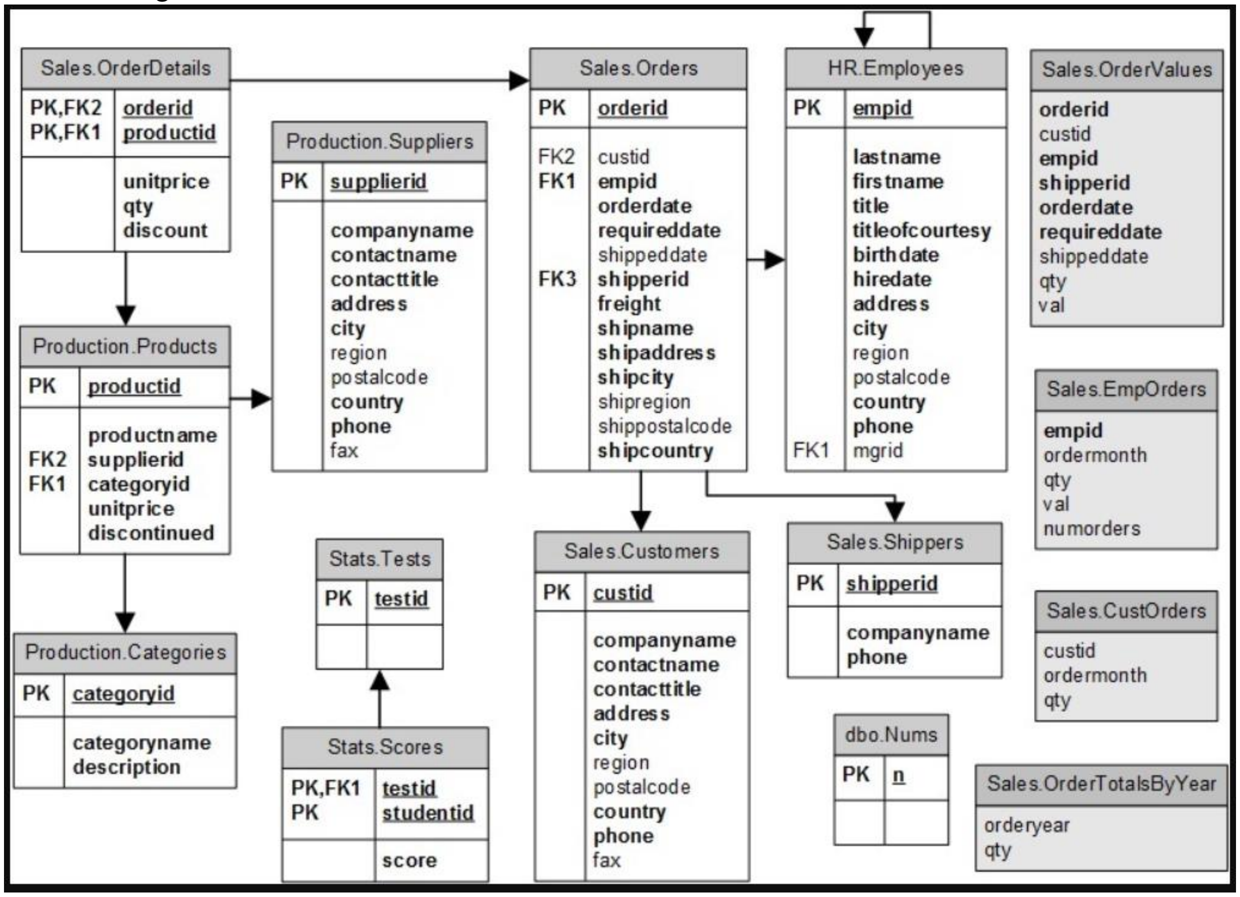
```
/*
Neba Nfonsang
Project 5: Queries that use certain keywords
*/

USE TSQLV4
```



```
/*
Query # 1: find cumulative sum by empid where
results are ordered by ordermonth or date
*/

SELECT empid, ordermonth, val,
    SUM(val) OVER (PARTITION BY empid
                ORDER BY ordermonth
                ROWS BETWEEN UNBOUNDED PRECEDING
                    AND CURRENT ROW) AS runval
FROM Sales.EmpOrders;

/*
Query # 2: Use RANK and DENSE_RANK based on how you
want ranks returned after a tie. DENSE_RANK counts
distinct values lower than the current row while
RANK counts rows that have lower values. Rank of 8
indicates 7 rows have lower values and a DENSE_RANK
of 8 means 8 distinct rows have lower values.
*/

SELECT empid, ordermonth, val,
    RANK() OVER (ORDER BY val) AS [Rank],
    DENSE_RANK() OVER (ORDER BY val) AS DenseRank
FROM Sales.EmpOrders;
```

```
/*
Query # 3: USE ROW_NUMBER to produce unique values
even when there is a tie
*/

SELECT empid, ordermonth, val,
    ROW_NUMBER()  OVER (ORDER BY val) AS rownum,
    RANK() OVER (ORDER BY val) AS [Rank],
    DENSE_RANK() OVER (ORDER BY val) AS DenseRank
FROM Sales.EmpOrders;


/*
Query # 4: Use the rank functions with PARTITION to
specify groups for which ranking should be done
*/

SELECT orderid, custid, val,
    ROW_NUMBER()  OVER (PARTITION BY custid
                        ORDER BY val DESC) AS
rownum
FROM Sales.OrderValues
ORDER BY custid, val;
```

```
/*
Query # 5: The order of computation is different
from the order of presentation. The ORDER BY after
the FROM clause determines the order of
presentation.
*/

SELECT orderid, custid, val,
    ROW_NUMBER()  OVER (PARTITION BY custid
                        ORDER BY val DESC) AS
rownum
FROM Sales.OrderValues
ORDER BY custid, val;

/*
Query # 6: Use NTILE to create equal size groups
with sequential group numbers based on the rows
*/

SELECT orderid, custid, val,
    ROW_NUMBER()  OVER (ORDER BY val DESC) AS
rownum,
    NTILE(100) OVER (ORDER BY val DESC) AS [ntile]
FROM Sales.OrderValues;
```

```
/*
Query # 7: to return only unique values in the val
column use the GROUP BY because the DISTINCT clause
is for distinct rows in the data distinct values of
a single column even if only a single column is
being retrieved.
*/

SELECT val,
    ROW_NUMBER()  OVER (ORDER BY val DESC) AS rownum
FROM Sales.OrderValues
GROUP BY val;

/*
Query # 8: LAST_NAME and FIRST_NAME retrieves last
name and first name in the window frame
then use it for the new column specified
*/

SELECT custid, orderid, val,
  FIRST_VALUE(val) OVER(PARTITION BY custid
                     ORDER BY orderdate, orderid
                        ROWS BETWEEN UNBOUNDED
PRECEDING
                                 AND CURRENT ROW) AS
firstval,
  LAST_VALUE(val)  OVER(PARTITION BY custid
                        ORDER BY orderdate, orderid
                        ROWS BETWEEN CURRENT ROW
                                 AND UNBOUNDED
FOLLOWING) AS lastval
FROM Sales.OrderValues
ORDER BY custid, orderdate, orderid;
```

```
/*
Query # 9: LAG AND LEAD uses the previous and next
value of the specified column
*/

SELECT custid, orderid, val,
LAG(val) OVER(PARTITION BY custid
                        ORDER BY orderdate, orderid)
AS preval,

 LEAD(val)  OVER(PARTITION BY custid
                        ORDER BY orderdate, orderid)
AS nextval

FROM Sales.OrderValues
ORDER BY custid, orderdate, orderid;

/*
Query # 10:
*/

SELECT orderid, custid, val,
  100. * val / SUM(val) OVER() AS pctall,
  100. * val / SUM(val) OVER(PARTITION BY custid) AS
pctcust
FROM Sales.OrderValues;
```

```
/*
Query # 11:
*/

SELECT empid, ordermonth, val,
    MIN(val) OVER (PARTITION BY empid
                ORDER BY ordermonth
                ROWS BETWEEN UNBOUNDED PRECEDING
                    AND CURRENT ROW) AS runmin,

    AVG(val) OVER (PARTITION BY empid
                ORDER BY ordermonth
                ROWS BETWEEN UNBOUNDED PRECEDING
                    AND CURRENT ROW) AS runaverage,
    SUM(val) OVER (PARTITION BY empid
                ORDER BY ordermonth
                ROWS BETWEEN UNBOUNDED PRECEDING
                    AND CURRENT ROW) AS runtotal

FROM Sales.EmpOrders;
```