In-class exercises (SQL):

1. Write a SQL query that returns the average price and variance per product

```
SELECT PRODNR, AVG(PURCHASE_PRICE) AS AVG_PRICE, VARIANCE(PURCHASE_PRICE) AS
VAR_PRICE
FROM SUPPLIES
GROUP BY PRODNR
```

2. Write a SQL query that returns the product name, average price, and variance for each product

```
SELECT P.PRODNAME, AVG(S.PURCHASE_PRICE) AS AVG_PRICE, VARIANCE(S.PURCHASE_PRICE)
AS VAR_PRICE
FROM PRODUCT P, SUPPLIES S
WHERE P.PRODNR = S.PRODNR
GROUP BY P.PRODNR
```

3. Write a nested SQL query to retrieve the supplier name of each supplier who supplies more than five products

```
SELECT SUPNAME
FROM SUPPLIER
WHERE SUPNR IN
(SELECT SUPNR FROM SUPPLIES GROUP BY SUPNR HAVING COUNT(DISTINCT PRODNR) > 5)
```

4. Write a correlated SQL query to retrieve the number and status of all suppliers, except for the three suppliers with the lowest supplier status

```
SELECT R1.SUPNR, R1.SUPSTATUS
FROM SUPPLIER R1
WHERE R1.SUPSTATUS IS NOT NULL
AND 2 < (SELECT COUNT(*) FROM SUPPLIER R2 WHERE R2.SUPSTATUS < R1.SUPSTATUS)
```

5. Write a SQL query to retrieve the name, number, and outstanding orders of all suppliers that have outstanding orders, except for the supplier(s) with the least outstanding orders

```
SELECT R1.SUPNAME, R1.SUPNR, COUNT(*)
FROM PURCHASE_ORDER PO1, SUPPLIER R1
WHERE PO1.SUPNR = R1.SUPNR
GROUP BY R1.SUPNR
HAVING COUNT(*) > ANY
(SELECT COUNT(*) FROM PURCHASE_ORDER GROUP BY SUPNR)
```

6. Write a query to select all supplier numbers, together with their supplier name and total number of outstanding orders for each supplier. Include all suppliers in the result, even if there are no outstanding orders for that supplier at the moment.

```
SELECT S.SUPNR, S.SUPNAME, COUNT(PO.PONR) AS OUTSTANDING_ORDERS
FROM SUPPLIER AS S LEFT OUTER JOIN PURCHASE_ORDER AS PO ON (S.SUPNR = PO.SUPNR)
GROUP BY S.SUPNR
```

In-class exercises (Stored Procedure):

1. Create a stored procedure that retrieves the total outstanding orders

```
CREATE PROCEDURE GETTOTALORDER()
SELECT SUM(QUANTITY)
FROM PO_LINE
```

To test:

```
CALL GETTOTALORDER()
```

2. Create a stored procedure that retrieves the corresponding product name of a given product number

```
CREATE PROCEDURE GETPRODNAME(IN PDNR CHAR(4))
SELECT PRODNAME
FROM PRODUCT
WHERE PRODNR = PDNR
```

To test:

```
CALL GETPRODNAME('0119')
```

3. Create a stored procedure that retrieves the product name and total order with the largest total order

```
CREATE PROCEDURE GETLARGESTORDER()
SELECT P.PRODNAME, SUM(PO.QUANTITY)
FROM PRODUCT P, PO_LINE PO
WHERE P.PRODNR = PO.PRODNR
GROUP BY PO.PRODNR
HAVING SUM(PO.QUANTITY) >= ALL
(SELECT SUM(QUANTITY) FROM PO_LINE GROUP BY PRODNR)
```

To test:

```
CALL GETLARGESTORDER()
```

Alternative approach (the way we did in class, not the most efficient way):

```
CREATE PROCEDURE GETLARGESTORDER()
```

```
SELECT T.NAME, T.TOTORDER FROM
(SELECT P.PRODNAME NAME, SUM(PO.QUANTITY) TOTORDER
        FROM PRODUCT P, PO_LINE PO
        WHERE P.PRODNR = PO.PRODNR
        GROUP BY P.PRODNR) T
WHERE T.TOTORDER >= ALL
        (SELECT SUM(QUANTITY) FROM PO_LINE GROUP BY PRODNR)
```

To test:

```
CALL GETLARGESTORDER()
```

In-Class Exercises (stored procedure, recursive, etc.)

Stored Procedure:

```
use world;
create procedure getcityname(in ide int, out cityname char(35))
select name into cityname from city where ID = ide;


call getcityname(100, @cityname1);
select @cityname1;
```

Recursive:

```
WITH RECURSIVE cte (n) AS
(
  SELECT 1
  UNION ALL
  SELECT (n + 1) FROM cte WHERE n < 100
)
SELECT * FROM cte;
```

Stored procedure to calculator factorial:

```
Delimiter //
CREATE PROCEDURE fact(IN x INT)
            BEGIN
    DECLARE result INT;
    DECLARE i INT;
    SET result = 1;
    SET i = 1;
    WHILE i <= x DO
```

```
    SET result = result * i;
    SET i = i + 1;
    END WHILE;
    SELECT x AS Number, result as Factorial;
    end //

call fact(5);
```