ASSIGNMENT-8

QUESTION:1-
SOLUTION:

```
WITH KNOWS_PERSON_WITH_SKILLS AS
(SELECT DISTINCT P.PID, P.PNAME FROM PERSON P, KNOWS K
WHERE P.PID=K.PID1 AND K.PID2 IN (SELECT PS.PID FROM PERSONSKILL PS
GROUP BY PS.PID HAVING COUNT(PS.SKILL)>3 ))

SELECT KPWS.PID, KPWS.PNAME FROM KNOWS_PERSON_WITH_SKILLS KPWS,
PERSON P WHERE KPWS.PID=P.PID AND P.CITY='Chicago';
```

QUESTION:2-
SOLUTION:

A)

```
SELECT DISTINCT P.PID, P.PNAME FROM PERSON P, KNOWS K WHERE
P.PID=K.PID1 AND K.PID2 IN
(SELECT W.PID FROM WORKSFOR W WHERE W.CNAME='Apple'
INTERSECT
SELECT W.PID FROM WORKSFOR W WHERE W.SALARY<=60000
INTERSECT
SELECT P.PID FROM PERSON P WHERE P.BIRTHYEAR <2000);
```

B)

```
WITH KNOWS_NUMBER_OF_PEOPLE AS
(SELECT K.PID1 AS PID, COUNT(K.PID2) AS KNOWS_PEOPLE FROM KNOWS K
GROUP BY K.PID1)

SELECT DISTINCT KNOP1.PID AS P1, KNOP2.PID AS P2 FROM
KNOWS_NUMBER_OF_PEOPLE KNOP1, KNOWS_NUMBER_OF_PEOPLE KNOP2
WHERE KNOP1.PID!=KNOP2.PID AND KNOP1.KNOWS_PEOPLE =
KNOP2.KNOWS_PEOPLE ORDER BY KNOP1.PID;
```

QUESTION:3-
SOLUTION:

```
CREATE TABLE A (X INTEGER);
INSERT INTO A VALUES (1),(2),(3),(4),(5),(6),(7),(8);
SELECT * FROM A;
SELECT (X, CBRT(X), POWER(X,2),10^X, FACTORIAL(X), LOG(2,X)) AS
RESULT_TUPLE FROM A;
```

QUESTION:4-
SOLUTION:

```
CREATE OR REPLACE FUNCTION INS_INTO_R()
RETURNS TRIGGER AS
$$
BEGIN
INSERT INTO V (SELECT A FROM R WHERE R.A!=A
                        AND B IN (SELECT B FROM S)
                        AND A NOT IN (SELECT A FROM V));
RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER INSERT_R
AFTER INSERT ON R
FOR EACH ROW
EXECUTE PROCEDURE INS_INTO_R();


CREATE OR REPLACE FUNCTION INS_INTO_S()
RETURNS TRIGGER AS
$$
BEGIN
INSERT INTO V (SELECT B FROM R WHERE B IN (SELECT B FROM S)
                        AND (SELECT C FROM S WHERE S.C!=c)
                        AND B NOT IN (SELECT B FROM V));
RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER INSERT_S
AFTER INSERT ON S
FOR EACH ROW
EXECUTE PROCEDURE INS_INTO_S();
```

QUESTION:5-
SOLUTION:
A)
USING BLOCK NESTED-LOOP JOIN
O(R + (S+ (S*T)/B)*R/B)

B)
USING SORT-MERGE JOIN
O(|R|log_B(|R|) + |S|log_B(|S|) + |T|log_B(|T|))

QUESTION:6-
SOLUTION:

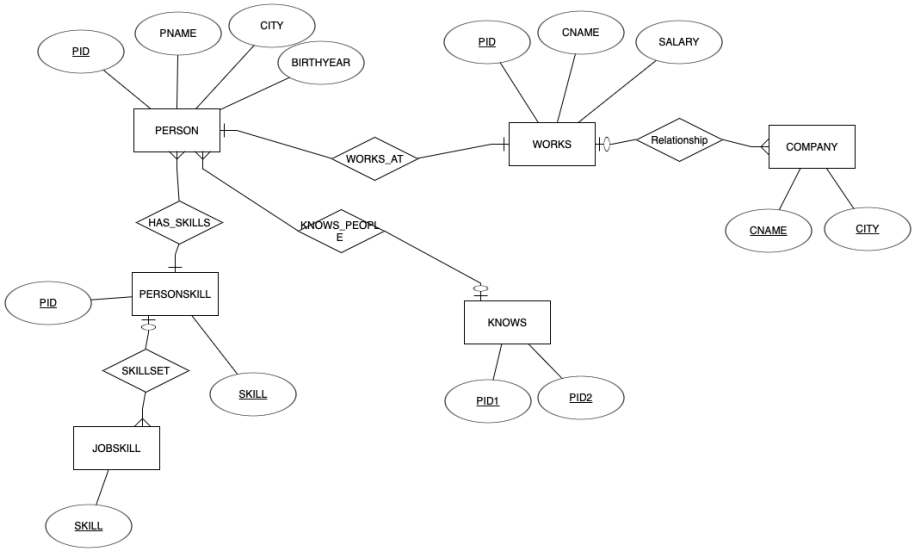
```
CREATE OR REPLACE FUNCTION PRIME_NUM(X INTEGER)
RETURNS BOOLEAN AS
$$
DECLARE I INT:= 2;
BEGIN
IF X=1 OR X=0 THEN
RETURN FALSE;
ELSE
FOR I IN 2.. X/2
LOOP
IF X%I=0 THEN
RETURN FALSE;
END IF;
END LOOP;
END IF;
RETURN TRUE;
END;
$$
LANGUAGE PLPGSQL;
```

QUESTION:7-
SOLUTION:

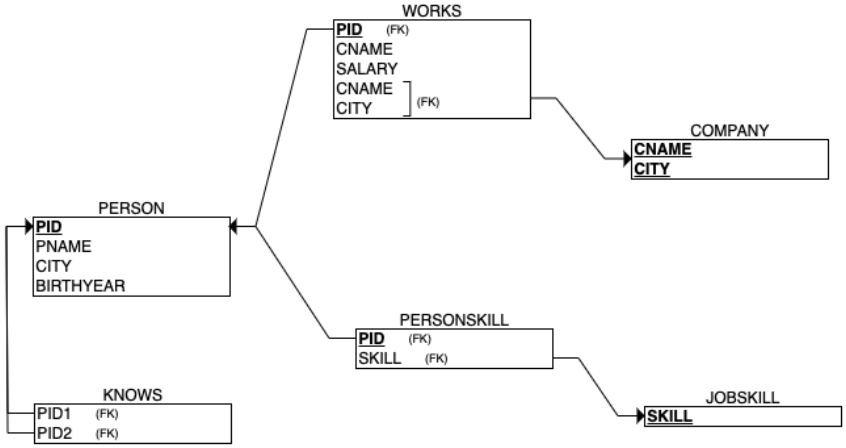
QUESTION:8-
SOLUTION:
A)

```
WITH RECURSIVE PARENT_CHILD(ORDER1) AS
((SELECT PARENT, PARENT FROM PC) UNION (SELECT CHILD, CHILD FROM PC)
UNION
 SELECT PC1.CHILD, PC2.CHILD FROM RPC PARENT_CHILD, PC PC1, PC PC2
 WHERE RPC.ORDER1= PC2.PARENT)
 SELECT DISTINCT ORDER1 FROM PARENT_CHILD;
```

QUESTION:9-
SOLUTION:

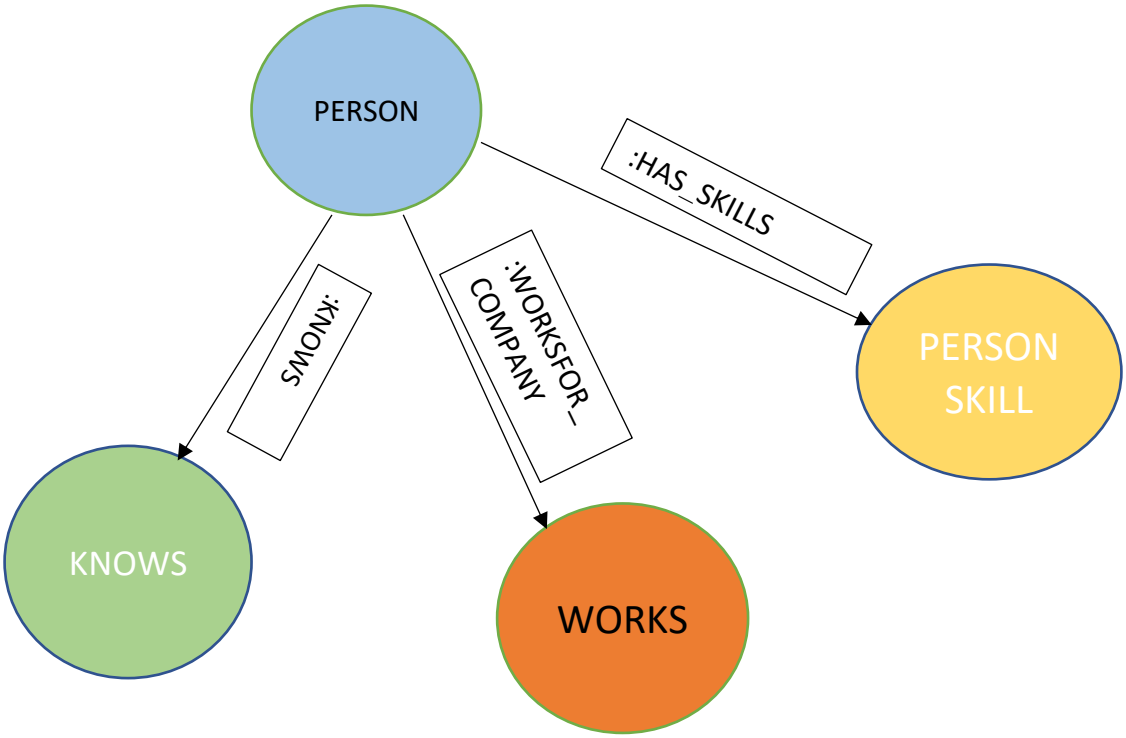


QUESTION:10-
SOLUTION:



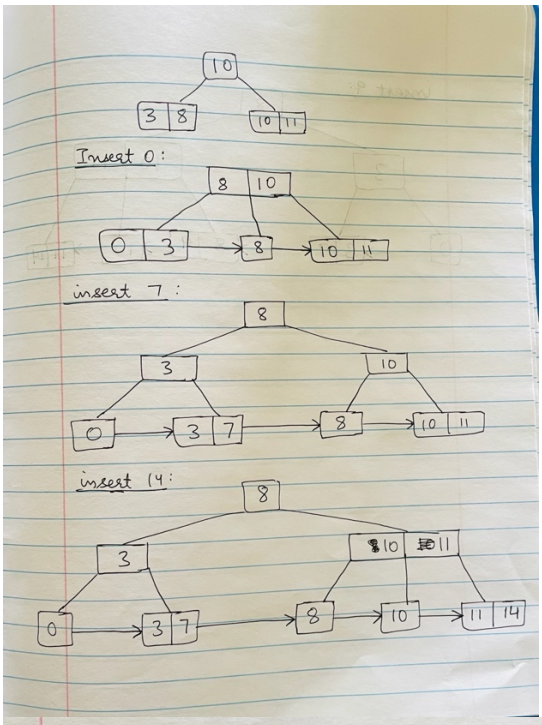
QUESTION:11-
SOLUTION:

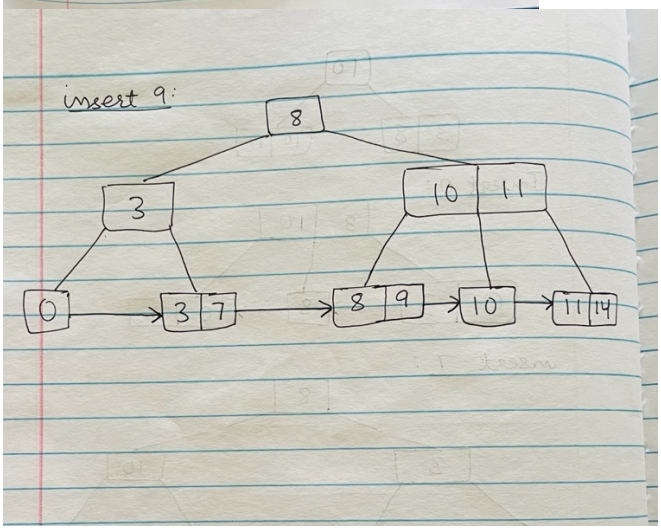
A)



B)
MATCH(p:PERSON {PNAME:'John'}) –[w:WORKS_FOR] -> (:WORKS)
WHERE w.SALARY >=50000
RETURN p

QUESTION:12-
SOLUTION:





QUESTION:13-
SOLUTION:
A) R1(x); R2(y); R1(z); R2(x); R1(y)

Since there are no cycles, hence the given the given schedule is conflict serializable.
The conflict-equivalent schedule can be given by:
R1(x); R1(z); R1(y); R2(x); R2(y)

B) R1(x); W2(y); R1(z); R3(z); W2(x); R1(y)

As we can see that the given schedule has cycle. Thus, it is not conflict serializable.

C) R1(z); W2(x); R2(z); R2(y); W1(x); W3(z); W1(y); R3(x)

Since there are no cycles, hence the given the given schedule is conflict serializable.
The conflict-equivalent schedule can be given by:
R1(z); W2(x); R2(y); R2(z); W1(x); W3(z); W1(y); R3(x)