```
/******************************************************************************************
********************************/
/*                                          FINAL PROJECT
*/
/*                                          CSCI4125-Spring 2018
*/
/*                                          Queries
*/
/*
*/
/*                                          R. N. Guillory
*/
/*                                          J. T. Marchan
*/
/*                                          G. T. Swanson
*/
/******************************************************************************************
********************************/

/*1. List a specific company?s workers by names. ++++*/
SELECT DISTINCT full_name
FROM works
NATURAL JOIN pers_full_name
NATURAL JOIN position
WHERE comp_id = 1113
AND end_date IS NULL
ORDER BY full_name ASC;

/*2. List a specific company?s staff by salary in descending order. ++++*/
WITH position_yearly_pay AS(
    SELECT pos_code, primary_sector_code, pay_rate AS yearly_pay
    FROM position
    NATURAL JOIN company
    WHERE pay_type = 'S'
    UNION
    SELECT pos_code, primary_sector_code, pay_rate*1920 AS pay_rate
    FROM position
    NATURAL JOIN company
    WHERE pay_type = 'W')
SELECT full_name, yearly_pay
FROM works
NATURAL JOIN pers_full_name
NATURAL JOIN position
NATURAL JOIN position_yearly_pay
WHERE comp_id = 1112
AND end_date IS NULL
ORDER BY yearly_pay DESC;

/*3. List companies' labor cost (total salaries and wage rates by 1920 hours) in descending
order. ++++Both++++*/
WITH position_yearly_pay AS(
    SELECT pos_code, primary_sector_code, pay_rate AS yearly_pay
    FROM position
    NATURAL JOIN company
    WHERE pay_type = 'S'
    UNION
    SELECT pos_code, primary_sector_code, pay_rate*1920 AS pay_rate
    FROM position
    NATURAL JOIN company
    WHERE pay_type = 'W')
SELECT comp_id, comp_name, SUM(yearly_pay) AS labor_cost
FROM company
NATURAL JOIN works
NATURAL JOIN position
```

```sql
NATURAL JOIN position_yearly_pay
WHERE end_date IS NULL
GROUP BY comp_id, comp_name
ORDER BY labor_cost DESC;

/*4. Given a person?s identifier, find all the job positions this person is currently holding
and worked in the past. ++++*/
SELECT p.pos_code, pos_title, start_date, end_date
FROM position p
JOIN works w on p.pos_code = w.pos_code
WHERE pers_id = 7;

/*5. Given a person?s identifier, list this person?s knowledge/skills in a readable format.
++++*/
SELECT ks_title, description, training_level
FROM know_skill ks
JOIN has_skill hs ON ks.ks_code = hs.ks_code
WHERE pers_id = 7;

/*6. Given a person?s identifier, list the skill gap between the requirements of this worker?
s job position(s) and his/her
skills.
    NEED TO REDO THIS ONE. STARTED BUT GOT LOST IN THE WITH STATEMENTS. REVISITING LATER.*/
WITH worker_current_positions AS (
SELECT pers_id, pos_code
FROM works
WHERE pers_id = 2
AND end_date IS NULL)
SELECT pos_code, ks_code
FROM position_skills
NATURAL JOIN worker_current_positions
MINUS
(SELECT pos_code, ks_code
FROM has_skill
NATURAL JOIN worker_current_positions)

/*7. List the required knowledge/skills of a pos_code and a job category code in a readable
format.
     (Two queries) +++Both+++*/
/*First query*/
SELECT pos_code, pos_title, ks_title, description
FROM  position
NATURAL JOIN position_skills
NATURAL JOIN know_skill
WHERE  pos_code = 10;

/* Second query.*/
--SELECT job_category_title, nwcet_title, LISTAGG(ks_title, ', ')
SELECT job_category_title, nwcet_title, ks_title
FROM job_category jc
JOIN nwcet ON jc.core_skill = nwcet.nwcet_code
NATURAL JOIN know_skill
WHERE cat_code = '15-1250';

/*8. Given a person?s identifier, list a person?s missing knowledge/skills for a specific
pos_code in a readable format. +++*/
WITH missing_skills AS (
    SELECT ks_code
    FROM position_skills    --No "required_skills" table, so set the position condition for
"REQUIRED SKILLS"
    WHERE prefer = 'R'
    AND pos_code = 15
    MINUS
    SELECT ks_code
    FROM has_skill
    WHERE pers_id = 11)
SELECT ks_title
FROM missing_skills
NATURAL JOIN know_skill;

/*9. Given a person?s identifier and a pos_code, list the courses (course id and title) that
each alone teaches all the
missing knowledge/skills for this person to pursue the specific job position.*/
SELECT DISTINCT c_code
FROM provides_skill ps1
WHERE NOT EXISTS (
  SELECT ks_code
  FROM position_skills
  WHERE pos_code = 7
  MINUS
  SELECT ks_code
  FROM has_skill
  WHERE pers_id = 12
  MINUS
  SELECT ks_code
  FROM provides_skill ps2
  WHERE ps1.c_code = ps2.c_code)

/*10. Suppose the skill gap of a worker and the requirement of a desired job position can be
covered by one course.
Find the ?quickest? solution for this worker. Show the course, section information and the
completion date.*/
WITH quickest_solution AS (
SELECT DISTINCT c_code
FROM            course c
WHERE NOT EXISTS(
                SELECT ks_code
                FROM position_skills    --No "required_skills" table, so set the position
condition for "REQUIRED SKILLS"
                WHERE prefer = 'R'
                AND pos_code = 7
                MINUS
                SELECT ks_code
                FROM provides_skill ps
                WHERE ps.c_code = c.c_code))
SELECT qs.c_code, sec_code, complete_date
FROM quickest_solution qs
JOIN section s ON qs.c_code = s.c_code
NATURAL JOIN person
WHERE complete_date = (SELECT MIN(complete_date) FROM section)
AND pers_id = 15;

/*11. Suppose the skill gap of a worker and the requirement of a desired job position can be
covered by one course.
Find the cheapest course to make up one?s skill gap by showing the course to take and the
cost (of the section
price).*/
--          NEEDS TO BE TINKERED WITH TO RETURN JUST ONE RESULT
WITH cheapest_solution AS (
SELECT DISTINCT c_code, title, retail_price AS course_cost
FROM            course c
WHERE NOT EXISTS(
                SELECT ks_code
                FROM position_skills    --No "required_skills" table, so set the position
condition for "REQUIRED SKILLS"
                WHERE prefer = 'R'
                AND pos_code = 7
                MINUS
                SELECT ks_code
                FROM provides_skill ps
                WHERE ps.c_code = c.c_code)
ORDER BY        retail_price ASC)

SELECT      title, price AS course_cost
FROM        cheapest_solution cs
JOIN        section s on cs.c_code = s.c_code
NATURAL JOIN person
WHERE       pers_id = 9
AND ROWNUM <=1
ORDER BY    course_cost ASC;

/*12.
If query #9 returns nothing, then find the course sets that their combination covers all the
missing knowledge/
skills for a person to pursue a pos_code. The considered course sets will not include more
than three courses. If
multiple course sets are found, list the course sets (with their course IDs) in the order of
the ascending order of the
course sets? total costs.*/
DROP SEQUENCE courseSet_seq;
CREATE SEQUENCE courseSet_seq
START WITH 1
INCREMENT BY 1
MAXVALUE 999999999
NOCYCLE;

DROP TABLE courseSet;
CREATE TABLE courseSet(
csetID NUMBER(8,0) PRIMARY KEY,
c_code1 NUMBER(6,0),
c_code2 NUMBER(6,0),
c_code3 NUMBER(6,0),
cSetSize NUMBER(2,0),
cSetCost NUMBER(10,2));

INSERT INTO courseSet(
SELECT courseSet_seq.NEXTVAL, c1.c_code, c2.c_code, null, 2, c1.retail_price +
c2.retail_price
FROM course c1, course c2
WHERE c1.c_code < c2.c_code);

INSERT INTO courseSet(
SELECT courseSet_seq.NEXTVAL, c1.c_code, c2.c_code, c3.c_code, 3, c1.retail_price +
c2.retail_price + c3.retail_price
FROM course c1, course c2, course c3
WHERE c1.c_code < c2.c_code
AND c2.c_code < c3.c_code);

DROP TABLE courseSet_skill;
CREATE TABLE courseSet_skill(
csetID NUMBER(8,0),
ks_code VARCHAR(8)
);

INSERT INTO courseSet_skill(csetID,ks_code)
(SELECT csetID, ks_code
FROM courseSet cSet1
JOIN provides_skill cSkill1
ON cSet1.c_code1 = cSkill1.c_code)
UNION
(SELECT csetID, ks_code
FROM courseSet cSet2
JOIN provides_skill cSkill2
ON cSet2.c_code2 = cSkill2.c_code)
UNION
```

```
(SELECT csetID, ks_code
FROM courseSet cSet3
JOIN provides_skill cSkill3
ON cSet3.c_code3 = cSkill3.c_code);

WITH coverCSET AS (
SELECT csetID, csetSize FROM courseSet
WHERE NOT EXISTS (
SELECT ks_code FROM position_skills WHERE pos_code = 7
MINUS
SELECT ks_code FROM has_skill WHERE pers_id = 7

MINUS
SELECT ks_code FROM courseSet_skill
WHERE
courseSet.csetID = courseSet_skill.csetID))
SELECT c_code1, c_code2, c_code3, csetsize, csetcost
FROM coverCSET
NATURAL JOIN courseSet
WHERE csetsize = (SELECT MIN(csetsize)
                  FROM covercset NATURAL JOIN courseSet);

ORDER BY csetcost ASC;
/*13. Given a person?s identifier, list all the job categories that a person is qualified
for. ++++*/
WITH qualifiedJobCategories AS (
                SELECT nwcet_code
                FROM core_skill
                MINUS
                SELECT nwcet_code
                FROM know_skill
                NATURAL JOIN    (SELECT ks_code
                                 FROM has_skill
                                 WHERE pers_id = 7))
SELECT DISTINCT cat_code
FROM qualifiedJobCategories
NATURAL JOIN core_skill;
/*14. Given a person?s identifier, find the job position with the highest pay rate for this
person according to his/her skill
possession.*/
WITH highest_pay AS(
SELECT DISTINCT full_name, pos_title, MAX(pay_rate) highest_salary
FROM has_skill hs
NATURAL JOIN position
JOIN pers_full_name p
    ON hs.pers_id = p.pers_id
WHERE hs.pers_id = 9
GROUP BY full_name, pos_title
ORDER BY highest_salary DESC)
SELECT *
FROM highest_pay
WHERE ROWNUM <= 1;

/*15. Given a position code, list all the names along with the emails of the persons who are
qualified for this position. */
SELECT (first_name || ' ' ||  last_name) full_name, email
FROM person p
WHERE NOT EXISTS (  SELECT ks_code
                    FROM position_skills ps
                    WHERE pos_code = 1
                    MINUS
                    SELECT ks_code
                    FROM has_skill hs
                    WHERE p.pers_id = hs.pers_id);
```

```
/*16. When a company cannot find any qualified person for a job position, a secondary
solution is to find a person who
is almost qualified to the job position. Make a ?missing-one? list that lists people who miss
only one skill for a
specified pos_code. ++++Double check data, but appears to work. */
WITH pos_skills AS (
SELECT ks_code FROM position_skills WHERE pos_code = 47)
SELECT pers_id, COUNT(*) FROM
(SELECT pers_id, ks_code FROM pos_skills, person
MINUS
SELECT pers_id, ks_code FROM has_skill)
GROUP BY pers_id
HAVING COUNT(*) = 1;

/*17. List each of the skill code and the number of people who misses the skill and are in
the missing-one list for a
given position code in the ascending order of the people counts. ++++*/
WITH pos_skills AS (
SELECT ks_code FROM position_skills WHERE pos_code = 7),
people_missing_one AS (
SELECT pers_id FROM
(SELECT pers_id, ks_code FROM pos_skills, person
MINUS
SELECT pers_id, ks_code FROM has_skill)
GROUP BY pers_id
HAVING COUNT(*) = 1)
SELECT ks_code, COUNT(*) FROM
(SELECT ks_code FROM
    (SELECT pers_id, ks_code FROM pos_skills, person
    MINUS
    SELECT pers_id, ks_code FROM has_skill))
GROUP BY ks_code;

/*18. Suppose there is a new position that has nobody qualified. List the persons who miss
the least number of skills
that are required by this pos_code and report the ?least number?. ++++ */
WITH pos_skills AS (
SELECT ks_code FROM position_skills WHERE pos_code = 7),
missing_skills AS (
SELECT pers_id, COUNT(*) AS missing_skills_count FROM
(SELECT pers_id, ks_code FROM pos_skills, person
MINUS
SELECT pers_id, ks_code FROM has_skill)
GROUP BY pers_id
)
SELECT pers_id, missing_skills_count
FROM missing_skills
WHERE missing_skills_count =
      (SELECT MIN(missing_skills_count) FROM missing_skills);

/*19. For a specified position code and a given small number k, make a ?missing-k? list that
lists the people?s IDs and
the number of missing skills for the people who miss only up to k skills in the ascending
order of missing skills. ++++*/
SELECT pers_id, num_missing
FROM missing_count
WHERE num_missing <= 10
AND pos_code = 7
ORDER BY num_missing;

/*20. Given a position code and its corresponding missing-k list specified in Question 19.
Find every skill that is
needed by at least one person in the given missing-k list. List each skill code and the
number of people who need
it in the descending order of the people counts. ++++*/
```

```
SELECT ks_code, COUNT(*)
FROM missing_skills
NATURAL JOIN (SELECT pers_id, pos_code, COUNT(*) AS num_missing
              FROM missing_skills
              GROUP BY pers_id, pos_code)
WHERE pos_code = 7
AND num_missing <= 10
GROUP BY ks_code
ORDER BY COUNT(*) DESC;

/*21. In a local or national crisis, we need to find all the people who once held a job
position of the special job category
identifier. List per_id, name, job position title and the years the person worked (starting
year and ending year). +++++*/
SELECT pers_id, full_name, pos_title, start_date, end_date
FROM person
NATURAL JOIN pers_full_name
NATURAL JOIN works
NATURAL JOIN position
WHERE cat_code = '15-1250'
AND end_date IS NOT NULL;

/*22. Find all the unemployed people who once held a job position of the given pos_code.
+++++*/
SELECT pers_id
FROM unemployed_people
NATURAL JOIN works
WHERE end_date IS NOT NULL
AND pos_code = 12;

/*23. Find out the biggest employer in terms of number of employees and the total amount of
salaries and wages paid to
employees. (Two queries) +++++*/
/*First query*/
WITH company_employee_count AS (
    SELECT comp_id, COUNT(*) AS empl_count
    FROM works
    NATURAL JOIN position
    WHERE end_date IS NULL
    GROUP BY comp_id)
SELECT comp_name, empl_count
FROM company
NATURAL JOIN company_employee_count
WHERE empl_count =
                (SELECT MAX(empl_count)
                 FROM company_employee_count);

/*Second query*/
WITH company_labor_cost AS (
    SELECT comp_id, SUM(yearly_pay) AS labor_cost
    FROM company
    NATURAL JOIN position
    NATURAL JOIN position_yearly_pay
    GROUP BY comp_id)
SELECT comp_id, comp_name, labor_cost
FROM company_labor_cost
NATURAL JOIN company
WHERE labor_cost = (SELECT MAX(labor_cost)
                    FROM company_labor_cost);

/*24. Find out the job distribution among business sectors; find out the biggest sector in
terms of number of employees
and the total amount of salaries and wages paid to employees. (Two queries) +++++*/
/*Query 1*/
```

```
SELECT primary_sector_code
FROM sector_employee_count
WHERE sec_empl_count = (
                    SELECT MAX(sec_empl_count)
                    FROM sector_employee_count);
/*Query 2*/
SELECT primary_sector_code
FROM sector_labor_cost
WHERE sec_labor_cost = (
                    SELECT MAX(sec_labor_cost)
                    FROM sector_labor_cost);

/*25. Find out (1) the number of the people whose earnings increased, (2) the number of those
whose earnings
decreased, (3) the ratio of (# of earning increased : # of earning decreased), (4) the
average earning changing rate
of for the workers in a specific business sector (use attribute ?primary sector? in table
Company. [Hint: earning
change = the sum of a person?s current income ? the sum of the person?s earning when he/she
was holding his/her
the latest previous job position. For (4), only count the earning from the specified sector
(companies? ?primary
sector?)] +++++*/
-- (SELECT SYSDATE FROM DUAL) == NOW
/* 25.1 */
SELECT COUNT(*) AS increase_count
FROM pay_change
WHERE diff > 0;
/* 25.2 */
SELECT COUNT(*) AS decrease_count
FROM pay_change
WHERE diff < 0;
/* 25.3 */
WITH inc_count AS (
        SELECT COUNT(*) AS increase_count
        FROM pay_change
        WHERE diff > 0),
     dec_count AS (
        SELECT COUNT(*) AS decrease_count
        FROM pay_change
        WHERE diff < 0)
SELECT increase_count / decrease_count AS ratio
FROM inc_count, dec_count;
/* 25.4 */
SELECT AVG(pay_diff)
FROM pay_change_by_sector
WHERE primary_sector_code = '45102010';

/*26. Find the leaf-node job categories that have the most openings due to lack of qualified
workers. If there are many
opening positions of a job category but at the same time there are many qualified jobless
people, then training
cannot help fill up this type of job position. What we want to find is such a job category
that has the largest
difference between vacancies (the unfilled job positions of this category) and the number of
jobless people who
are qualified for the job positions of this category.*/
WITH leafNodes AS(
    SELECT cat_code
    FROM job_category child
    WHERE NOT EXISTS (
        SELECT *
        FROM job_category
        WHERE parent_cat_code = child.cat_code)),
vacancies AS(
    SELECT w.pers_id,p.pos_code,cat_code
    FROM position p
    JOIN works w
        ON p.pos_code = w.pos_code
    WHERE end_date IS NOT NULL),
qualifiedJobCategories AS (
                SELECT nwcet_code
                FROM core_skill
                MINUS
                SELECT nwcet_code
                FROM know_skill
                NATURAL JOIN    (SELECT ks_code
                                 FROM has_skill
                                 WHERE pers_id = 2))
SELECT DISTINCT cat_code
FROM position p
JOIN works w ON p.pos_code = w.pos_code
NATURAL JOIN core_skill cs
JOIN qualifiedJobCategories q ON cs.nwcet_code = q.nwcet_code
NATURAL JOIN vacancies  --ON p.cat_code = v.cat_code
NATURAL JOIN leafNodes  --ON p.cat_code = l.cat_code
WHERE end_date IS NOT NULL;

/*27. Find the courses that can help most jobless people find a job position by training them
toward the jobs of this
category that have the most openings due to lack of qualified workers.*/
DROP SEQUENCE courseSet_seq;
CREATE SEQUENCE courseSet_seq
    START WITH 1
    INCREMENT BY 1
    MAXVALUE 999999999
    NOCYCLE;

DROP TABLE courseSet;
CREATE TABLE courseSet(
    csetID NUMBER(8,0) PRIMARY KEY,
    c_code1 NUMBER(6,0),
    c_code2 NUMBER(6,0),
    c_code3 NUMBER(6,0),
    cSetSize NUMBER(2,0),
    cSetCost NUMBER(10,2));

INSERT INTO courseSet(
    SELECT courseSet_seq.NEXTVAL, c1.c_code, null, null, 1, c1.retail_price
    FROM course c1);

INSERT INTO courseSet(
        SELECT courseSet_seq.NEXTVAL, c1.c_code, c2.c_code, null, 2, c1.retail_price +
c2.retail_price
        FROM course c1, course c2
        WHERE c1.c_code < c2.c_code);

INSERT INTO courseSet(
        SELECT courseSet_seq.NEXTVAL, c1.c_code, c2.c_code, c3.c_code, 3,  c1.retail_price +
c2.retail_price + c3.retail_price
        FROM course c1, course c2, course c3
        WHERE c1.c_code < c2.c_code
        AND c2.c_code < c3.c_code);

DROP TABLE courseSkill;
CREATE TABLE courseSkill(
    c_code NUMBER(6,0),
    ks_code VARCHAR(8)
);
```

```
INSERT INTO courseSkill(
        SELECT c_code, ks_code
        FROM course, know_skill
);

DROP TABLE courseSet_skill;
CREATE TABLE courseSet_skill(
    csetID NUMBER(8,0),
    ks_code VARCHAR(8)
);

INSERT INTO courseSet_skill(csetID,ks_code)
    (SELECT csetID, ks_code
    FROM courseSet cSet1
    JOIN courseSkill cSkill1
        ON cSet1.c_code1 = cSkill1.c_code)
    UNION
    (SELECT csetID, ks_code
    FROM courseSet cSet2
    JOIN courseSkill cSkill2
        ON cSet2.c_code2 = cSkill2.c_code)
    UNION
    (SELECT csetID, ks_code
    FROM courseSet cSet3
    JOIN courseSkill cSkill3
        ON cSet3.c_code3 = cSkill3.c_code);

WITH leafNodes AS(
    SELECT cat_code
    FROM job_category child
    WHERE NOT EXISTS (
        SELECT *
        FROM job_category
        WHERE parent_cat_code = child.cat_code)),
vacancies AS(
    SELECT w.pers_id,p.pos_code,cat_code
    FROM position p
    JOIN works w
        ON p.pos_code = w.pos_code
    WHERE end_date IS NOT NULL),
qualifiedJobCategories AS (
                SELECT nwcet_code
                FROM core_skill
                MINUS
                SELECT nwcet_code
                FROM know_skill
                NATURAL JOIN    (SELECT ks_code
                                 FROM has_skill
                                 WHERE pers_id = 2)),
qualifiedCatCodes AS(
    SELECT DISTINCT cat_code
    FROM position p
    JOIN works w ON p.pos_code = w.pos_code
    NATURAL JOIN core_skill cs
    JOIN qualifiedJobCategories q ON cs.nwcet_code = q.nwcet_code
    NATURAL JOIN vacancies  --ON p.cat_code = v.cat_code
    NATURAL JOIN leafNodes  --ON p.cat_code = l.cat_code
    WHERE end_date IS NOT NULL),
coverCSet(csetID, cSetSize) AS (
    SELECT csetID, cSetSize
    FROM courseSet cSet
    WHERE NOT EXISTS(
        SELECT ks_code
        FROM (  SELECT DISTINCT ks_code
                FROM            courseSet_Skill c
                WHERE NOT EXISTS(
                                SELECT ks_code
                                FROM position_skills    --No "required_skills" table, so set
the position condition for "REQUIRED SKILLS"
                                WHERE prefer = 'R'
                                AND pos_code = 10
                                MINUS
                                SELECT ks_code
                                FROM has_skill
                                WHERE pers_id = 9
                                MINUS
                                SELECT ks_code
                                FROM courseSet_Skill cs
                                WHERE c.cSetID = cs.cSetID))
        MINUS
        SELECT ks_code
        FROM courseSet_skill cSkill
        WHERE cSkill.csetID = cSet.cSetID
    )
),
coursesNeeded AS(
    SELECT c_code1, c_code2, c_code3, cSetCost
    FROM coverCSet
    NATURAL JOIN courseSet
    WHERE cSetSize = (SELECT MIN(cSetSize)
                     FROM coverCSet)
    ORDER BY cSetCost ASC)
SELECT DISTINCT pos_code, pos_title, c_code1 AS c_code, title
FROM qualifiedCatCodes q
JOIN position p ON q.cat_code = p.cat_code
NATURAL JOIN coursesNeeded cn
JOIN course c ON cn.c_code1 = c.c_code;



/*Graduate requirement*/
/*28. NOT REQUIRED FOR 05APR18 TURN IN
List all the courses, directly or indirectly required, that a person has to take in order to
be qualified for a job
position of the given category, according to his/her skills possessed and courses taken.
(required for graduate
students only)*/
```