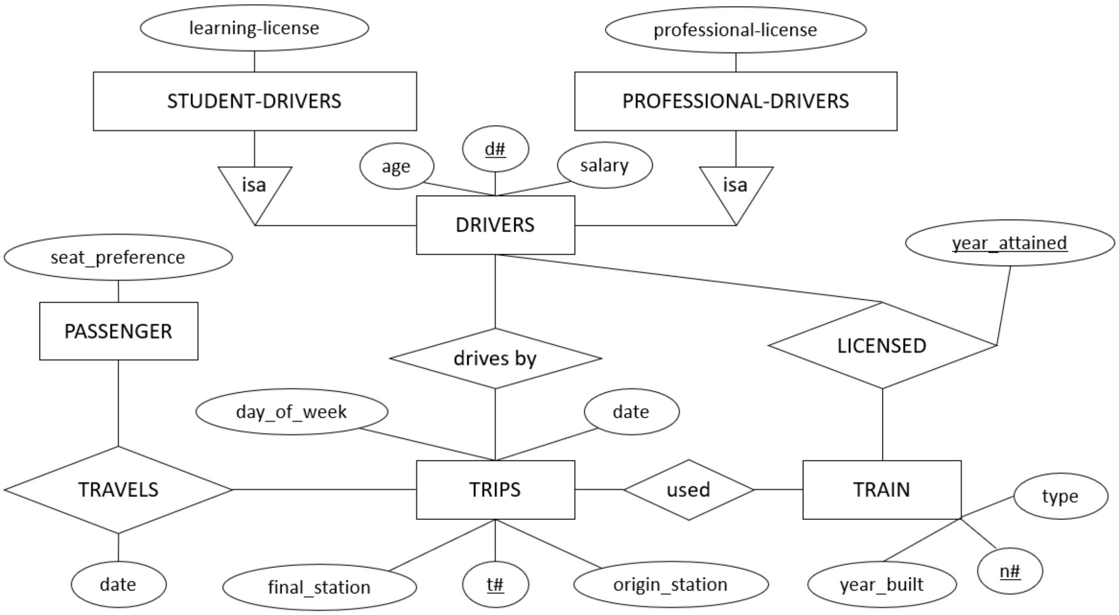
21st CSEC – Past Year Paper Solution (2020 – 2021 Semester 2)
CZ2007 – Introduction to Databases

1 (a)



(b) R1 ≔ STUDENT-DRIVERS $\bowtie_{\text{STUDENT-DRIVERS.d\#=LICENSED.d\#}}$ LICENSED // joining two tables
R2 ≔ $\sigma_{\text{year\_attained>=2018}}$ R1 // filter licensed with year_attained larger than 2018
R3 ≔ R2 $\bowtie_{\text{R2.n\#=TRAIN.n\#}}$ TRAIN // joining two tables
Answer ≔ $\pi_{\text{n\#,type,year\_built}}$ ($\sigma_{\text{year\_built=2010}}$ R3) // filter train built in 2010

(c) R1 ≔ $\sigma_{\text{age=50}}$ DRIVERS // filter drivers who are 50 y.o
R2 ≔ R1 $\bowtie_{\text{R1.d\#=TRIPS.d\#}}$ TRIPS // joining two tables
R3 ≔ $\sigma_{\text{origin\_station='Jurong Station' AND day\_of\_week='Monday'}}$ R2 // filter trips with conditions stated in the question
R4 ≔ $\gamma_{\text{d\#, COUNT(t\#)}\rightarrow\text{trip\_count}}$ R3 // count number of trips
R5 := R4 // duplicate the relation
Answer := R4 $\bowtie_{\text{R4.trip\_count=R5.trip\_count AND R4.d\#<>R5.d\#}}$ R5 // finding pairs of different drivers

Editor's note: There might be other alternatives on solving these kinds of problems. As long you put a logical justification, you will be fine.

2 (a) *Editor's note:* The idea is to find all closures of all possible combinations.

A and F are both nowhere on the RHS of all functional dependencies. Hence, both are included in the candidate keys.
$\{AF\}^+ = \{AFDBC\}$
E and H are not covered. Notice that
$\{AEF\}^+ = \{AEFDBCH\}$
$\{AFH\}^+ = \{AFHDBCE\}$

Therefore, AEF and AFH are both candidate keys.

(b) **(i)** $\{EC\}^+ = \{EC\}$. **No**, AD is not covered.
(ii) $\{ADF\}^+ = \{ADFBC\}$. **No**, E is not covered.
(iii) $\{A\}^+ = \{AD\}$. **No**, DH is not covered.
(iv) $\{AED\}^+ = \{AEDHC\}$. **Yes**, C is covered.
(v) $\{DH\}^+ = \{DHEC\}$. **Yes**, C is covered.

(c) A schema R is in BCNF if and only if the LHS of every non-trivial FD contains a key of R.
A schema R satisfies 3NF, if and only if for every non-trivial FD
either the LHS contains a key of R or each attribute in RHS is contained in a key of R.

Notice that A is the key in R1, E is the key in R2, AFE and AFH are both keys in R3.

A→D satisfies both BCNF and 3NF as the LHS (A) contains the key A.
AE→H violates BCNF as the LHS (AE) does not contain the key AFE. However, it satisfies 3NF as H is contained in the key AFH.
E→C satisfies both BCNF and 3NF as the LHS (E) contains the key E.
H→E violates BCNF as the LHS (H) does not contain the key AFH. However, it satisfies 3NF as E is contained in the key AFE.

The decomposition follows 3NF, but not BCNF.

Editor's note: DF→BC is already broken down, hence is not considered.*

3 **(a)**

```
SELECT DISTINCT   c.customerid, c.name
FROM              CUSTOMER c, PURCHASE r, PURCHASE_ITEM s, ITEM I, PRODUCER
p
WHERE             c.customerid = r.customerid
AND               r.purchaseid = s.purchaseid
AND               s.itemid = i.itemid
AND               i.producerid = p.producerid
AND               p.country = ‘Denmark’
AND               r.date >= ‘2019-12-03’
AND               r.date < ‘2019-12-04’;
```

Editor's note: On the cases where date is stored as a string, we can directly write `r.date=‘2019-12-03’.`

(b)

```
SELECT DISTINCT   c.customerid, c.name
FROM              CUSTOMER c, PURCHASE r
WHERE             c.customerid = r.customerid
AND               r.purchaseid IN (SELECT s.purchaseid
                                   FROM PURCHASE r, PURCHASE_ITEM s,
                             ITEM I, PRODUCER p
```

```
                                    WHERE r.purchaseid = s.purchaseid
                                    AND s.itemid = i.itemid
                                    AND i.producerid = p.producerid
                                    AND i.category = ‘dairy’
                                    AND p.country = ‘AUS’
                                    INTERSECT
                                    SELECT s.purchaseid
                                    FROM PURCHASE r, PURCHASE_ITEM s,
                              ITEM I, PRODUCER p
                                    WHERE r.purchaseid = s.purchaseid
                                    AND s.itemid = i.itemid
                                    AND i.producerid = p.producerid
                                    AND i.category = ‘coffee’
                                    AND p.country = ‘SIN’);
```

Editor’s note: The idea is to create a nested query that returns all purchaseid which contain both items.

(c)

```
WITH PRICE AS
(SELECT           i.itemid, SUM(i.price * s.quantity) AS sales
FROM              PURCHASE r, PURCHASE_ITEM s, ITEM i
WHERE             r.purchaseid = s.purchaseid
AND               s.itemid = i.itemid
AND               YEAR(p.date) = ‘2019’
GROUP BY          i.itemid)

SELECT            i.itemid, i.name, i.price
FROM              ITEM I, PRICE t
WHERE             i.itemid = t.itemid
AND               t.sales = (SELECT MAX(sales) FROM PRICE);
```

Editor’s note: The idea is to create a temporary view that stores the sales of every item sold in 2019.

(d)

```
WITH CustomerCategory AS
(SELECT           c1.customerid, COUNT(DISTINCT i1.itemid) AS countitem
FROM              CUSTOMER c1, PURCHASE r1, PURCHASE_ITEM s1, ITEM i1
WHERE             c1.customerid = r1.customerid
AND               r1.purchaseid = s1.purchaseid
AND               s1.itemid = i1.itemid
AND               i1.category = ‘dairy’
GROUP BY          c1.customerid),
UniqueDairy AS
(SELECT           COUNT(DISTINCT i.itemid) AS uniqueitem
FROM              ITEM i2
WHERE             i2.category = ‘dairy’),

CustomerAllDairy AS
(SELECT           c3.customerid
FROM              CustomerCategory c3, UniqueDairy u3
WHERE             c3.countitem = u3.uniqueitem),

SELECT DISTINCT   c.customerid, c.name
FROM              CUSTOMER c, CustomerAllDairy c4
WHERE             c.customerid IN (SELECT * FROM c4)
AND               NOT EXISTS (SELECT *
                              FROM PURCHASE r4, PURCHASE_ITEM s4, ITEM i4
                              WHERE c.customerid = r4.customerid
                              AND   r4.purchaseid = s4.purchaseid
                              AND   s4.itemid = i4.itemid
                              AND   i4.category = ‘coffee’);
```

(e)

```
WITH EXPENSE AS
(SELECT           c.customerid, SUM(i.price * s.quantity) AS total
FROM              CUSTOMER c, PURCHASE r, PURCHASE_ITEM s, ITEM i
WHERE             c.customerid = r.customerid
AND               r.purchaseid = s.purchaseid
AND               s.itemid = i.itemid
GROUP BY          c.customerid)

SELECT            c.customerid, c.name
FROM              CUSTOMER c, EXPENSE e
WHERE             c.customerid = e.customerid
AND               e.total <> (SELECT MAX(sales) FROM EXPENSE
                              WHERE total <> (SELECT MAX(total) FROM
                                          EXPENSE));
```

Editor’s note: We assume that if there are more than one customer with the largest spending, the second biggest money will still belong to the next in line. (not any of the top customers)

4 (a) (i)

```
CREATE VIEW       CategoryCountrySales AS
(SELECT           i.category, p.country, COUNT(i.itemid) AS
numberofitem, SUM(i.price*pi.quantity) AS sales
FROM              ITEM i, PURCHASE r, PRODUCER p, PURCHASE_ITEM s
WHERE           p.purchaseid = s.purchaseid
AND             s.itemid = i.itemid
AND             i.producerid = p.producerid
GROUP BY        i.category, p.country);
```

(ii)

```
WITH            CustomerCatItem AS
(SELECT         c.customerid, c.name AS customername, i.category,
SUM(r.quantity) AS numberofitem
FROM            CUSTOMER c, ITEM i, PURCHASE r, PURCHASE_ITEM s
WHERE           c.customerid = r.customerid
AND             r.purchaseid = s.purchaseid
AND             s.itemid = i.itemid
GROUP BY        c.customerid, i.category);
```

(b) (i)

```
CREATE TRIGGER Q4b(i)
BEFORE INSERT ON WORKS
FOR EACH ROW
BEGIN
      IF NOT EXISTS (SELECT * FROM COMPANY WHERE company_name =
NEW.company_name) THEN
            INSERT INTO COMPANY(company_name)
            VALUES NEW.company_name
      ENDIF
END;
```

(ii)

```
CREATE TRIGGER Q4b(ii)
BEFORE INSERT ON MANAGES
FOR EACH ROW
BEGIN
      IF EXISTS (SELECT manager_name FROM MANAGES
                  GROUP BY manager_name
                  HAVING COUNT(person_name)>5)
      THEN
            RAISE EXCEPTION
      ENDIF

      IF NEW.person_name = NEW.manager_name
      THEN
            RAISE EXCEPTION
      ENDIF
END;
```

(c) (i)

```
CREATE INDEX      Q4c(i)
ON CUSTOMER(age, sex)
```

Editor’s note: *Note that the order is not interchangeable (sex, age is incorrect).*

(ii)

```
CREATE INDEX      Q4c(ii)
ON CUSTOMER(NRIC, sex)
```

Editor's note: Creating the index on only NRIC is sufficient as it speeds up the look up on both queries, whereas NRIC and sex would increase the lookup time on the first query and even more on the second query.

(d)

```
<!DOCTYPE customers[
<!ELEMENT customers(customer*)>
<!ELEMENT customer(name, address, phone, purchase+)>
<!ELEMENT purchase(date, item+)>
<!ELEMENT item(name, price, producerid, category, quantity)>
<!ATTLIST customer customerid ID #required>
<!ATTLIST purchase purchaseid ID #required>
<!ATTLIST item itemid ID #required>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT producerid (#PCDATA)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
]>
```

Solver: Leonardo Irvin Pratama (lpratama001@e.ntu.edu.sg)