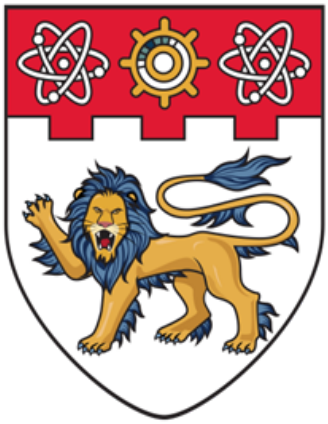

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2207 INTRODUCTION TO DATABASES

LAB 5: FINAL DEMONSTRATION

Names	Nalin Sharma Lim Jia Earn Jayden Yeo Derrick Ng Choon Seng Dexter Voon Kai Xian
Lab Group	A50
Group Number	4

Table of Contents

APPENDIX C: INDIVIDUAL CONTRIBUTION

Full Name	Individual Contribution to Lab 5	% of contribution	Signature
Nalin Sharma	SQL DDL Commands, Queries	20%	
Lim Jia Earn	SQL DDL Commands, Queries	20%	
Jayden Yeo	SQL DDL Commands, Queries	20%	
Derrick Ng Choon Seng	SQL DDL Commands, Queries	20%	
Dexter Voon Kai Xian	SQL DDL Commands, Queries	20%	

I - SQL DDL Commands:

```sql
CREATE TABLE PUBLICATIONS (
    Publication_id INT IDENTITY(1,1),
    Avg_rating DECIMAL(2,1) DEFAULT null,
    Publisher VARCHAR(50),
    Year_of_publication INT,
    PRIMARY KEY (Publication_id)
)

CREATE TABLE CUSTOMERS (
    Customer_id INT,
    Customer_name VARCHAR(50),
    PRIMARY KEY (Customer_id)
)

CREATE TABLE EMPLOYEES (
    Employee_id INT,
    Employee_name VARCHAR(50),
    Monthly_salary DECIMAL(10,2),
    PRIMARY KEY (Employee_id)
)

CREATE TABLE BOOKSTORES (
    Company_id INT,
    PRIMARY KEY(Company_id)
)


CREATE TABLE BOOK_PUBLICATIONS (
    Publication_id INT,
    Book_title VARCHAR(50)
    PRIMARY KEY (Publication_id, Book_title),
    FOREIGN KEY (Publication_id) REFERENCES PUBLICATIONS(Publication_id)
)

CREATE TABLE MAGAZINE_PUBLICATIONS (
    Publication_id INT,
    Magazine_title VARCHAR(50),
    Issue_number INT,
    PRIMARY KEY (Publication_id, Magazine_title),
    FOREIGN KEY (Publication_id) REFERENCES PUBLICATIONS(Publication_id)
)

CREATE TABLE STORE_STOCKS (
    Publication_id INT,
    Company_id INT,
    Store_pub_id INT,
    Quantity INT,
    Selling_price DECIMAL(5,2),
    PRIMARY KEY (Publication_id, Company_id),
    FOREIGN KEY (Company_id) REFERENCES BOOKSTORES (Company_id),
    FOREIGN KEY (Publication_id) REFERENCES PUBLICATIONS (Publication_id)
)


CREATE TABLE PRICE_HISTORY(
     Company_id INT,
     Publication_id INT,
     Startdate DATE,
     Enddate DATE DEFAULT null,
     Selling_price DECIMAL(5,2),
     PRIMARY KEY (Publication_id, Company_id, Startdate, Selling_price),
     FOREIGN KEY (Publication_id, Company_id) REFERENCES STORE_STOCKS
       (Publication_id, Company_id)
 )
```

```sql
CREATE TABLE ORDERS (
    Order_id int IDENTITY(1,1),
    Customer_id INT,
    Delivery_date DATE,
    Shipping_cost DECIMAL(5,2) DEFAULT 5,
    Total_quantity INT DEFAULT null,
    Total_price DECIMAL(5,2),
    Shipping_address VARCHAR(100),
    Order_status VARCHAR(50) DEFAULT 'being processed',
    Time_stamp DATE,
    PRIMARY KEY (Order_id),
    FOREIGN KEY (Customer_id) REFERENCES CUSTOMERS (Customer_id),
    CHECK (Shipping_cost> 0 AND Order_status IN('being processed', 'shipped',
      'delivered', 'returned'))
)

CREATE TABLE ORDER_DETAILS_BOOKS(
    Order_id INT,
    Publication_id INT,
    Company_id INT,
    Book_title VARCHAR(50),
    Book_quantity INT,
    Price DECIMAL(5,2),
    PRIMARY KEY (Order_id, Publication_id, Company_id),
    FOREIGN KEY (Order_id) REFERENCES ORDERS(Order_id),
    FOREIGN KEY (Publication_id, Company_id) REFERENCES STORE_STOCKS
      (Publication_id, Company_id),
    FOREIGN KEY (Publication_id, Book_title) REFERENCES BOOK_PUBLICATIONS
      (Publication_id, Book_title)
)

CREATE TABLE ORDER_DETAILS_MAGAZINES    (
    Order_id INT,
    Publication_id INT,
    Company_id INT,
    Magazine_title VARCHAR(50),
    Magazine_quantity INT,
    Price DECIMAL(5,2),
    PRIMARY KEY (Order_id, Publication_id, Company_id),
    FOREIGN KEY (Order_id) REFERENCES ORDERS(Order_id),
    FOREIGN KEY (Publication_id, Company_id) REFERENCES STORE_STOCKS
      (Publication_id, Company_id),
    FOREIGN KEY (Publication_id, Magazine_title) REFERENCES
       MAGAZINE_PUBLICATIONS(Publication_id, Magazine_title)
 )

 CREATE TABLE COMPLAINTS (
     Complaint_id INT IDENTITY(1,1),
     Order_id INT,
     Customer_id INT,
     Employee_id INT DEFAULT null,
     Complaint_status VARCHAR(50) DEFAULT 'pending',
     Pickup_date DATE DEFAULT null,
     Addressed_date DATE DEFAULT null,
     PRIMARY KEY (Complaint_id),
     FOREIGN KEY (Order_id) REFERENCES ORDERS (Order_id),
     FOREIGN KEY (Customer_id) REFERENCES CUSTOMERS (Customer_id),
     FOREIGN KEY (Employee_id) REFERENCES EMPLOYEES (Employee_id),
     UNIQUE(Order_id),
     CHECK (Complaint_status IN('pending', 'being handled', 'addressed'))
 )

 CREATE TABLE COMPLAINT_PUBLICATION(
     Complaint_id INT,
     Publication_id INT,
     PRIMARY KEY (Complaint_id),
     FOREIGN KEY (Complaint_id) REFERENCES COMPLAINTS (Complaint_id),
     FOREIGN KEY (Publication_id) REFERENCES PUBLICATIONS (Publication_id)
 )

 CREATE TABLE COMPLAINT_BOOKSTORE(
     Complaint_id INT,
     Bookstore_id INT,
     PRIMARY KEY (Complaint_id),
     FOREIGN KEY (Complaint_id) REFERENCES COMPLAINTS (Complaint_id),
     FOREIGN KEY (Bookstore_id) REFERENCES BOOKSTORES (Company_id)
 )
```

```sql
CREATE TABLE RATINGS (
    Rating_id INT,
    Order_id INT,
    Publication_id INT,
    Comment VARCHAR(200) DEFAULT null,
    No_of_stars DECIMAL(2,1),
    Rate_date DATE,
    PRIMARY KEY (Rating_id),
    FOREIGN KEY (Order_id) REFERENCES ORDERS (Order_id),
    FOREIGN KEY (Publication_id) REFERENCES PUBLICATIONS (Publication_id),
    CHECK (No_of_stars <= 5)
)
```

II - SQL Statements for queries in Appendix B:

Query 1:

Find the average price of “Harry Porter Finale” on Ahamazon from 1 August 2022 to 31 August 2022.

Query Statements:

```
SELECT CAST(AVG(Selling_price) AS DECIMAL(5,2)) AS AvgPriceOfHP
FROM BOOK_PUBLICATIONS AS BP, PRICE_HISTORY AS PH
WHERE BP.Publication_ID = PH.Publication_ID AND
BP.Book_Title = 'Harry Porter Finale' AND
PH.Startdate BETWEEN '2022-08-01' AND '2022-08-31' AND
PH.Enddate BETWEEN '2022-08-01' AND '2022-08-31'
```

Query Brief Explanation:

1) The query uses inner join to combine the BOOK_PUBLICATIONS and PRICE_HISTORY tables.
2) The WHERE clause filters the results to only include the publication with the book title "Harry Porter Finale" on Ahamazon from 1 August 2022 to 31 August 2022.
3) The AVG function is used to find the average selling price of the book during this time period.

Query Output:

	AvgPriceOfHP
1	30.00

Query 2:

Find publications that received at least 10 ratings of “5” in August 2022, and rank them by their average ratings.

Query Statements:

```
SELECT COUNT(No_of_stars) AS NoOf5StarRating, Publication_id, RANK() OVER ( ORDER
  BY AVG(No_of_stars)) AS Ranking
FROM RATINGS AS R1
WHERE R1.Rate_date BETWEEN '2022-08-01' AND '2022-08-31' AND
R1.No_of_stars = 5.0
GROUP BY Publication_id
HAVING COUNT(No_of_stars) >= 10;
```

Query Brief Explanation:

1) The query uses the COUNT and AVG aggregate function to count the number of stars in the Ratings table and compute the average number of stars.
2) The query uses the rank and order by ranking.
3) The WHERE clause filters the results to only include ratings of "5" in August 2022.
4) The GROUP BY clause is used to group the results by publication ID.
5) The HAVING clause is used to filter the results to only include publications that received at least 10 ratings of "5".

Query Output:

	NoOf5StarRating	Publication_id	Ranking
1	11985	10	1

Query 3:

For all publications purchased in June 2022 that have been delivered, find the average time from the ordering date to the delivery date.

Query Statements:

```
SELECT order_pubs.Publication_id, AVG(DATEDIFF(DAY, ORDERS.Time_stamp,
  ORDERS.Delivery_date)) AS Avg_del_time
FROM ORDERS,
(SELECT Order_id, Publication_id
FROM ORDER_DETAILS_BOOKS
UNION
SELECT Order_id, Publication_id
FROM ORDER_DETAILS_MAGAZINES) AS order_pubs
WHERE ORDERS.Time_stamp BETWEEN '2022-06-01' AND '2022-06-30' AND
  ORDERS.Order_status = 'delivered' AND ORDERS.Order_id = order_pubs.Order_id
GROUP BY order_pubs.Publication_id
```

Query Brief Explanation:

1) The query uses AVG aggregate function and DATEDIFF function to find the average time from the ordering date(Time_stamp) to the delivery time in days.
2) The DATEDIFF function is to calculate the difference between the delivery date and timestamp date in days.
3) The query uses union to combine the ORDERS_DETAILS_BOOKS, and ORDERS_DETAILS_MAGAZINES tables as order_pubs.
4) The query uses an inner join where the ORDERS.Order_id = order_pubs.Order_id.
5) The WHERE filters the results to only include orders that were delivered in June 2022 and have a status of "delivered".
6) The GROUP BY is used to group the results by publication_id.

Query Output:

	Publication_id	Avg_del_time
1	1	164
2	3	130
3	5	3
4	6	97
5	7	3
6	9	164
7	10	83

Query 4:

Let us define the “latency” of an employee by the average that he/she takes to process a complaint. Find the employee with the smallest latency.

Query Statements:

```
SELECT TOP(1) Employee_name, Latency_days
FROM (SELECT DATEDIFF(DAY, Pickup_date, Addressed_date) AS Latency_days,
  Employee_id
      FROM COMPLAINTS) AS c , EMPLOYEES AS empl
WHERE Latency_days IS NOT null AND c.Employee_id = empl.Employee_id
ORDER BY Latency_days
```

Query Brief Explanation:

1) The query begins with the SELECT statement which is used to retrieve data from the database, and TOP(1) specifies that we want to retrieve the top 1 record.
2) The SELECT statement also specifies that we want to retrieve two columns: Employee_name and Latency_days
3) The FROM clause in the query specifies that we want to retrieve data from two tables: COMPLAINTS and EMPLOYEES
4) The query uses a subquery in the FROM clause to calculate Latency_days for each employee who handled a complaint. The DATEDIFF function is used to calculate the number of days between the Pickup_date and Addressed_date columns in the COMPLAINTS table. The AS keyword is used to alias the result of the subquery c, which is then used as a table in the main query.
5) The WHERE clause is used to filter the results of the query. In this case, we are selecting only those rows where the Latency_dats is not null and where the Employee_id from the COMPLAITNS table matches the employee_od in the EMPLOYEES table.
6) Finally, the ORDER BY clause is used to sort the results of the query by the Latency_days column in ascending order.

Query Output:

	Employee_name	Latency_days
1	Jack	0

Query 5:

Produce a list that contains (i) all publications published by Nanyang Publisher Company, and (ii) for each of them, the number of bookstores on Ahamazon that sell them.

Query Statements:

```
SELECT PUBLICATIONS.Publication_id, COUNT(DISTINCT STORE_STOCKS.Store_pub_id) AS
   Num_of_bookstores
FROM (PUBLICATIONS
      INNER JOIN
      BOOK_PUBLICATIONS
      ON PUBLICATIONS.Publication_id = BOOK_PUBLICATIONS.Publication_id)
      INNER JOIN
      STORE_STOCKS
      ON PUBLICATIONS.Publication_id = STORE_STOCKS.Publication_id
WHERE PUBLICATIONS.Publisher = 'Nanyang Publisher Company'
GROUP BY PUBLICATIONS.Publication_id
```

Query Brief Explanation:

1) To retrieve the Publication_ID and the number of bookstores that sell each publication, we use the inner join between Publications and Book_Publications first, this ensures that only publications that are books are retrieved.
2) Next, to get the information on which bookstores sell each book we use the inner join between store_stocks and the resulting table from the previous step.
3) Then, we use the where keyword to filter publications based on the specified publisher name.
4) Finally, we group by publication ID and utilise the COUNT function to count the number of distinct id values for each publication. This gives us our desired result.

Query Output:

	Publication_id	Num_of_bookstores
1	1	1
2	4	1
3	6	2

Query 6:

Find bookstores that made the most revenue in August 2022.

Query Statements:

```
SELECT TOP(1) *
FROM ((SELECT STORE_STOCKS.Company_ID, SUM(ORDER_DETAILS_BOOKS.Price *
  ORDER_DETAILS_BOOKS.Book_Quantity) AS Revenue
FROM STORE_STOCKS, ORDER_DETAILS_BOOKS, ORDERS
WHERE STORE_STOCKS.Publication_ID = ORDER_DETAILS_BOOKS.Publication_ID AND
STORE_STOCKS.Company_ID = ORDER_DETAILS_BOOKS.Company_ID AND
ORDERS.Order_ID = ORDER_DETAILS_BOOKS.Order_ID AND
ORDERS.Time_stamp BETWEEN '2022-08-01' AND '2022-08-31'
GROUP BY STORE_STOCKS.Company_ID)

UNION ALL

(SELECT STORE_STOCKS.Company_ID, SUM(ORDER_DETAILS_MAGAZINES.Price *
  ORDER_DETAILS_MAGAZINES.Magazine_Quantity) AS Revenue
FROM STORE_STOCKS, ORDER_DETAILS_MAGAZINES, ORDERS
WHERE STORE_STOCKS.Publication_ID = ORDER_DETAILS_MAGAZINES.Publication_ID AND
STORE_STOCKS.Company_id = ORDER_DETAILS_MAGAZINES.Company_id AND
ORDERS.Order_id = ORDER_DETAILS_MAGAZINES.Order_id AND
ORDERS.Time_stamp BETWEEN '2022-08-01' AND '2022-08-31'
GROUP BY STORE_STOCKS.Company_ID)) as new

ORDER BY Revenue DESC
```

Query Brief Explanation:

1) The query uses UNION ALL to combine the results of two subqueries, one for book sales and one for magazine sales.
2) Each subquery is similar to the previous query, but with different JOINs to the ORDERS tables.
3) Each subquery uses WHERE clause which filters the Time_stamp of the orders being placed in August 2022.
4) The subqueries select Company_ID and the revenue generated by each sale.
5) The query uses GROUP BY to group the results by Company_ID.
6) SUM aggregate function used to calculate the total revenue for each store.
7) The ORDER BY clause is used to sort the results in descending order of total revenue, so the bookstore with the highest revenue will be listed first.
8) Lastly, we select the top 1 record of the result which is the Company_ID that made the most revenue in August 2022.

Query Output:

	Company_ID	Revenue
1	5	75.00

Query 7:

For customers that made the most number of complaints, find the most expensive publication he/she has ever purchased.

Query Statements:

```
SELECT TOP(1) order_pubs.Publication_id, order_pubs.Price
FROM ORDERS,
(SELECT TOP(1) Customer_id, COUNT(Complaint_id) AS complaintsmade
FROM COMPLAINTS
GROUP BY Customer_id
ORDER BY complaintsmade DESC) AS cust_compl,
(SELECT Order_id, Publication_id, Company_id, Price
FROM ORDER_DETAILS_BOOKS
UNION
SELECT Order_id, Publication_id, Company_id, Price
FROM ORDER_DETAILS_MAGAZINES) AS order_pubs
WHERE ORDERS.Customer_id = cust_compl.Customer_id AND
ORDERS.Order_id = order_pubs.Order_id
ORDER BY order_pubs.Price DESC
```

Query Brief Explanation:

1) The query begins with the SELECT statement which is used to retrieve data from the database. The first line of the statement specifies that we want to retrieve the top one record.
2) The SELECT statement also specifies that we want to retrieve two columns: Publication_id and Price.
3) The FROM clause in the query specifies that we want to retrieve data from three tables: ORDERS, COMPLAINTS, ORDER_DETAILS_BOOKS, and ORDER_DETAILS_MAGAZINES.
4) The query uses a subquery in the FROM clause to identify the customer with the most complaints. The TOP keyword is used to select only the first row from the subquery. The COUNT function is used to count the number of complaints for each customer, and the GROUP BY clause groups the results by Customer_id. The AS keyword is used to alias the result of the subquery as cust_compl, which is then used as a table in the main query.
5) The query uses another subquery in the FROM clause to retrieve the details of each publication ordered by customers.
6) The UNION keyword is used to combine the results of two subqueries that

retrieve order details for books and magazines respectively. The AS keyword is used to alias the result of the subquery as order_pubs, which is then used as a table in the main query.

7) The WHERE clause is used to filter the results of the query. In this case, we are selecting only those rows where the Customer_id in the ORDERS table matches the Customer_id in the cust_compl table, and where the Order_id in the ORDERS table matches the Order_id in the order_pubs table.
8) Finally, the ORDER BY clause is used to sort the results of the query by the Price column in descending order. The TOP keyword is used to retrieve only the first row of the sorted results.

Query Output:

	Publication_id	Price
1	3	40.00

Query 8:

Find publications that have never been purchased by any customer in July 2022, but are the top 3 most purchased publications in August 2022.

Query Statements:

```
SELECT TOP(3) WITH TIES bought_aug.Publication_id, SUM(bought_aug.Qty) AS Qty
FROM
(SELECT DISTINCT order_pubs.Publication_id
FROM
(SELECT Order_id
FROM ORDERS
WHERE Time_stamp NOT BETWEEN '2022-07-01' AND '2022-07-31') AS orders,
(SELECT Order_id, Publication_id, Company_id, Book_quantity AS Qty
FROM ORDER_DETAILS_BOOKS
UNION
SELECT Order_id, Publication_id, Company_id, Magazine_quantity AS Qty
FROM ORDER_DETAILS_MAGAZINES) AS order_pubs
WHERE orders.Order_id = order_pubs.Order_id) AS not_bought_jul,
(SELECT order_pubs.Publication_id, order_pubs.Qty
FROM
(SELECT Order_id
FROM ORDERS
WHERE Time_stamp BETWEEN '2022-08-01' AND '2022-08-31') AS orders,
(SELECT Order_id, Publication_id, Company_id, Book_quantity AS Qty
FROM ORDER_DETAILS_BOOKS
UNION
SELECT Order_id, Publication_id, Company_id, Magazine_quantity AS Qty
FROM ORDER_DETAILS_MAGAZINES) AS order_pubs
WHERE orders.Order_id = order_pubs.Order_id) AS bought_aug
WHERE not_bought_jul.Publication_id = bought_aug.Publication_id
GROUP BY bought_aug.Publication_id
ORDER BY Qty DESC
```

Query Brief Explanation:

1) The query select all Order IDs that did not have a timestamp between July 1st and July 31st and then selecting the publication IDs of the corresponding books and magazines from the ORDER_DETAILS_BOOKS and ORDER_DETAILS_MAGAZINES tables.
2) The query then defines another subquery (bought_aug) that selects the publication IDs and quantities of all books and magazines that were purchased in August 2022.
3) It does this by selecting all Order IDs that have a timestamp between August 1st and August 31st and then selecting the publication IDs and quantities of

the corresponding books and magazines from the ORDER_DETAILS_BOOKS and ORDER_DETAILS_MAGAZINES tables.

4) The main query joins the two subqueries on the Publication ID column and groups the results by Publication ID.
5) It then uses the SUM aggregate function to sum up the quantities of each publication that were purchased in August
6) It uses ORDER BY which sort the results in descending order by quantity.
7) Finally, it uses the TOP clause with the WITH TIES option to return the top three publications with the same quantity as the third-highest quantity.

Query Output:

	Publication_id	Qty
1	5	3
2	9	2
3	10	1
4	1	1

Query 9:

Find publications that are increasingly being purchased over at least 3 months.

Query Statements:

```
WITH GroupedByMonthPublications AS (
    SELECT
        Publication_id,
        DATEFROMPARTS(YEAR(Time_stamp), MONTH(Time_stamp), 1) AS Month_and_year,
        COUNT(*) AS Orders_count
    FROM
        ORDERS
        JOIN (
            SELECT Order_id, Publication_id
            FROM ORDER_DETAILS_BOOKS
            UNION ALL
            SELECT Order_id, Publication_id
            FROM ORDER_DETAILS_MAGAZINES
        )
        AS COMBINED
        ON ORDERS.Order_id = COMBINED.Order_id

    GROUP BY
        Publication_id,
        DATEFROMPARTS(YEAR(Time_stamp), MONTH(Time_stamp), 1)
),
RankedPublications AS (
    SELECT
        Publication_id,
        Month_and_year,
        Orders_count,
        DENSE_RANK() OVER (PARTITION BY Publication_id
                           ORDER BY Month_and_year)
                    AS PublicationRank
        /*
            DENSE_RANK() assigns a rank to each Publication_id based on its
              Month_and_year
        */
    FROM
        GroupedByMonthPublications
)
SELECT
    PUBLICATIONS.Publication_id,
    PUBLICATIONS.Publication_title

FROM
    (
            SELECT Publication_id, Magazine_title as Publication_title
            FROM MAGAZINE_PUBLICATIONS
            UNION ALL
            SELECT Publication_id, Book_title
            FROM BOOK_PUBLICATIONS
        ) AS PUBLICATIONS
    JOIN (
```

```
        SELECT

            Publication_id,
            Month_and_year,
            Orders_count
        FROM
            RankedPublications
        WHERE
            PublicationRank <= 3
        )
        AS SortedPublications
        /* SortedPublications contains list of Publication_ids that satisfy our
          query */
    ON PUBLICATIONS.Publication_id = SortedPublications.Publication_id
GROUP BY
    PUBLICATIONS.Publication_id,
    PUBLICATIONS.Publication_title
HAVING
    COUNT(*) = 3
ORDER BY /* Just for tidiness */
    PUBLICATIONS.Publication_id
```

Query Brief Explanation:

1) A new table, GroupedByMonthPublications is created which joins the the ORDERS Table with ORDER_DETAILS_BOOKS and ORDER_DETAILS_MAGAZINES to retrieve the publication_id along with its corresponding date of order, Time_stamp.
2) Then we extract the month from Time_stamp
3) Then we create another new table, RankedPublications which uses the DENSE_RANK function to rank publications by its Time_stamp for each unique publication_id.
4) Then we create another table SortedPublications which will select publications that have at least 3 consecutive months based on its ranking in RankedPublications. All tuples in SortedPublications contain the publication_id which satisfy the query now
5) Lastly, match these publication_id with their corresponding titles in MAGAZINE_PUBLICATIONS and BOOK_PUBLICATIONS

Additional Thoughts:

When using DATEFROMPARTS to group our dates, we cannot simply group by month as we also need to consider years which would be useful for consecutive months such as

from dec-22 to feb-23. Else if our database contains data of several years, GroupBy will group all the months from different years together which would produce a wrong result.

Query Output:

	Publication_id	Publication_title
1	3	SG50
2	4	Just for laughs
3	5	How to read
4	9	Fashion

III - Printout of table records:

R1: PUBLICATIONS

	Publication_id	Avg_rating	Publisher	Year_of_publication
1	1	5.0	Nanyang Publisher Company	2020
2	2	5.0	Singapore Publisher Company	2010
3	3	4.5	Ngee Ann Publisher Company	2020
4	4	3.5	Nanyang Publisher Company	2022
5	5	4.0	Random Publisher Company	2018
6	6	4.0	Nanyang Publisher Company	2019
7	7	3.0	Random Publisher Company	2001
8	8	5.0	Ngee Ann Publisher Company	2023
9	9	3.5	Ngee Ann Publisher Company	2021
10	10	4.0	Singapore Publisher Company	2023

R2: BOOK_PUBLICATIONS

	Publication_id	Book_title
1	1	NTU history
2	3	SG50
3	4	Just for laughs
4	5	How to read
5	6	Harry Porter...

R3: MAGAZINE_PUBLICATIONS

	Publication_id	Magazine_title	Issue_number
1	2	Daily news	1
2	7	Gossip	1
3	8	Growing	2
4	9	Fashion	1
5	10	Sport	5

R4: EMPLOYEES

	Employee_id	Employee_name	Monthly_salary
1	1	Jack	10000.00
2	2	Bob	7000.00
3	3	Denise	10000.00
4	4	Harry	5000.00
5	5	Jane	4000.00
6	6	Jennie	5000.00
7	7	Lisa	5500.00

R5: ORDERS

	Order_id	Customer_id	Delivery_date	Shipping_cost	Total_quantity	Total_price
1	1	1	2022-12-01	50.00	12	305.00
2	2	4	2022-06-25	20.00	15	200.00
3	3	4	2022-10-01	20.00	6	190.00
4	4	3	2023-04-03	10.00	1	40.00
5	5	3	2023-04-01	5.00	2	30.00
6	6	2	2023-04-03	5.00	2	65.00
7	7	6	2023-04-05	5.00	1	15.00
8	8	1	2023-04-03	5.00	1	15.00
9	9	6	2022-05-01	15.00	1	30.00
10	10	5	2022-07-25	10.00	1	10.00
11	11	5	2022-07-17	25.00	1	15.00
12	12	4	2022-02-01	25.00	3	65.00
13	13	2	2022-06-28	10.00	5	70.00
14	14	2	2022-07-06	30.00	3	90.00
15	15	2	2022-08-06	30.00	3	90.00
16	16	3	2022-08-10	30.00	3	90.00
17	17	1	2022-09-30	30.00	3	90.00
18	18	1	2022-09-24	30.00	3	90.00
19	19	6	2022-09-30	30.00	3	90.00

R6: ORDER_DETAILS_BOOKS

	Order_id	Publication_id	Company_id	Book_title	Book_quantity	Price
1	1	1	2	NTU History	1	50.00
2	1	3	1	SG50	2	35.00
3	1	3	3	SG50	2	40.00
4	2	4	2	Just for laughs	5	10.00
5	3	3	3	SG50	1	40.00
6	3	6	3	Harry Porter F...	5	30.00
7	4	3	3	SG50	1	40.00
8	6	1	2	NTU History	1	50.00
9	9	6	3	Harry Porter F...	1	30.00
10	10	4	2	Just for laughs	1	10.00
11	12	3	1	SG50	1	35.00
12	12	4	2	Just for laughs	1	10.00
13	13	5	5	How to read	2	25.00
14	14	5	5	How to read	4	25.00
15	15	5	5	How to read	2	25.00
16	16	5	5	How to read	1	25.00
17	17	5	5	How to read	2	25.00
18	18	5	5	How to read	2	25.00
19	19	5	5	How to read	2	25.00

R7: ORDER_DETAILS_MAGAZINES

	Order_id	Publication_id	Company_id	Magazine_title	Magazine_quantity	Price
1	1	9	1	Fashion	2	15.00
2	1	10	7	Sport	5	15.00
3	2	10	7	Sport	10	15.00
4	5	9	1	Fashion	2	15.00
5	6	9	1	Fashion	1	15.00
6	7	10	7	Sport	1	15.00
7	8	9	1	Fashion	1	15.00
8	11	9	1	Fashion	1	15.00
9	12	2	5	Daily news	1	20.00
10	13	7	7	Gossip	1	15.00
11	13	10	7	Sport	2	15.00
12	14	8	1	Growing	3	30.00

R8: CUSTOMERS

	Customer_id	Customer_name
1	1	Mike
2	2	Zack
3	3	Alice
4	4	Ray
5	5	Bob
6	6	Dorry

R9: RATINGS

	Rating_id	Order_id	Publication_id	Comment	No_of_stars	Rate_date
1	1	1	1	very informative, thumbs up	5.0	2022-12-02
2	2	1	9	NULL	4.0	2022-12-02
.						
.						
119...	11997	2	10	Comment - 11997	5.0	2022-08-04
119...	11998	2	10	Comment - 11998	5.0	2022-08-04
119...	11999	2	10	Comment - 11999	5.0	2022-08-04

R10: STORE_STOCKS

	Publication_id	Company_id	Store_pub_id	Quantity	Selling_price
1	1	2	1	20	50.00
2	2	5	1	20	20.00
3	3	1	1	20	35.00
4	3	3	1	25	40.00
5	4	2	4	40	10.00
6	5	5	2	20	25.00
7	6	3	5	30	30.00
8	6	7	10	35	33.00
9	7	7	3	15	15.00
10	8	1	3	25	30.00
11	9	1	2	20	15.00
12	10	7	1	30	15.00

R11: PRICE_HISTORY

Company_id	Publication_id	Startdate	Enddate	Selling_price
1	3	2021-01-04	2021-01-07	18.00
1	3	2021-04-01	2021-07-01	18.00
1	3	2021-07-01	2022-11-01	23.00
1	3	2022-11-01	NULL	25.00
3	6	2022-08-01	2022-08-02	30.00
3	6	2022-08-03	2022-08-04	40.00
3	6	2022-08-30	2022-08-31	20.00
7	10	2022-04-01	NULL	15.00
7	10	2022-04-01	2023-01-20	18.00

R12: COMPLAINTS

	Complaint_id	Order_id	Customer_id	Employee_id	Complaint_status	Pickup_date
1	1	1	1	3	addressed	2022-12-03
2	2	2	4	1	addressed	2022-06-26
3	3	12	4	7	addressed	2022-02-05
4	4	3	4	2	addressed	2022-10-01
5	5	14	2	1	addressed	2022-07-06
6	6	10	5	6	being handled	2022-11-07
7	7	13	2	3	being handled	2022-07-01
8	8	9	6	1	pending	NULL
9	9	11	5	4	pending	NULL
10	11	4	3	7	addressed	2022-12-28

R13: COMPLAINT_PUBLICATION

	Complaint_id	Publication_id
1	1	3
2	2	10
3	3	4
4	4	6
5	9	9

R14: COMPLAINT_BOOKSTORE

	Complaint_id	Bookstore_id
1	5	1
2	6	2
3	7	7
4	8	3
5	11	3

R15: BOOKSTORE

	Company_id
1	1
2	2
3	3
4	5
5	7