SC2207 Database

Lab Project 4 & 5

REP2 Group 1:

Lim Zhi Yong

Hui Xujia

Ethan Lee Kei Shen

Huang Caihong

Chong Huai Zhi

Bhati Nancy

| SQL DDL Commands for Table Creation   | USER\_ACCOUNT(UID, Gender, DOB, Name) | | --- | | CREATE TABLE USER\_ACCOUNT (  UID INT NOT NULL,   Gender TEXT NOT NULL,  DOB DATETIME NOT NULL,  Name TEXT NOT NULL,  PRIMARY KEY (UID) ); |  | SHOP(SID, Type, MID, Shop\_Manager\_ID, Shop\_Manager\_Name) | | --- | | CREATE TABLE SHOP(  SID INT NOT NULL,  Type TEXT NOT NULL,  MID INT NOT NULL,  Shop\_Manager\_ID INT NOT NULL,  Shop\_Manager\_Name TEXT NOT NULL,  FOREIGN KEY (MID) REFERENCES MALL(MID) ON DELETE CASCADE,  PRIMARY KEY(SID) ); |  | SHOPPING(SID, UID, Amount\_spent, Date\_time\_in, Date\_time\_out) | | --- | | CREATE TABLE SHOPPING (  SID INT NOT NULL,  UID INT NOT NULL,  Amount\_spent FLOAT NOT NULL,  Date\_time\_in DATETIME NOT NULL,  Date\_time\_out DATETIME NOT NULL,  PRIMARY KEY (SID, UID, Date\_time\_in),  FOREIGN KEY (SID) REFERENCES SHOP(SID) ON DELETE CASCADE,  FOREIGN KEY (UID) REFERENCES USER\_ACCOUNT(UID) ON DELETE CASCADE ); |  | DINING(UID, OID, Amount\_spent, Date\_time\_in, Date\_time\_out) | | --- | | CREATE TABLE DINING (  UID INT NOT NULL,  OID INT NOT NULL,  Amount\_spent FLOAT NOT NULL,  Date\_time\_in DATETIME NOT NULL,  Date\_time\_out DATETIME NOT NULL,  PRIMARY KEY (UID, OID, Date\_time\_in),  FOREIGN KEY (UID) REFERENCES USER\_ACCOUNT(UID) ON DELETE CASCADE,  FOREIGN KEY (OID) REFERENCES RESTAURANT\_OUTLET(OID) ON DELETE CASCADE ); |  | MALL(MID, Address, NumShops, CID) | | --- | | CREATE TABLE MALL (  MID INT NOT NULL,   Address NVARCHAR(255)TEXT NOT NULL,  NumShops INT NOT NULL,  CID INT NOT NULL,  FOREIGN KEY (CID) REFERENCES MALL\_MGMT\_COMPANY(CID) ON DELETE CASCADE,  PRIMARY KEY(MID) ); |  | MALL\_MGMT\_COMPANY(CID, Address) | | --- | | CREATE TABLE MALL\_MGMT\_COMPANY (  CID INT NOT NULL ,  Address TEXT NOT NULL,  PRIMARY KEY(CID) ); |  | RESTAURANT\_OUTLET(OID, Unit\_no, MID, RID, Type, Name, Floor) | | --- | | CREATE TABLE RESTAURANT\_OUTLET (  OID INT NOT NULL,   Unit\_no TEXT NOT NULL,   MID INT NOT NULL,   RID INT NOT NULL,   Type TEXT NOT NULL,   Name NVARCHAR(255) NOT NULL,   Floor INT NOT NULL,  FOREIGN KEY (MID) REFERENCES MALL(MID) ON DELETE CASCADE,  FOREIGN KEY (RID) REFERENCES RESTAURANT\_CHAIN(RID) ON DELETE CASCADE,  PRIMARY KEY(OID) ); |  | RESTAURANT\_CHAIN(RID, Address) | | --- | | CREATE TABLE RESTAURANT\_CHAIN (  RID INT NOT NULL,   Address TEXT NOT NULL,  PRIMARY KEY(RID) ); |      | DAY\_PACKAGE (DID, Description, VID, UID, Date, ProductID) | | --- | | CREATE TABLE DAY\_PACKAGE (  DID INT NOT NULL,  Description TEXT,  VID INT NOT NULL,  UID INT NOT NULL,  Date DATETIME NOT NULL,  ProductID INT NOT NULL,  FOREIGN KEY (VID) REFERENCES VOUCHER(VID),  FOREIGN KEY (UID) REFERENCES USER\_ACCOUNT(UID),  PRIMARY KEY(DID) ); |      | COMPLAINT (CID,Text,Status,Filed\_date\_time,UID) | | --- | | CREATE TABLE COMPLAINT (  CID INT NOT NULL,   Text TEXT NOT NULL,   Status TEXT NOT NULL,   Filed\_date\_time DATETIME NOT NULL,   UID INT NOT NULL,  FOREIGN KEY (UID) REFERENCES USER\_ACCOUNT(UID),  PRIMARY KEY(CID) ); |      | COMPLAINTS\_ON\_RESTAURANT (CID, OID) | | --- | | CREATE TABLE COMPLAINTS\_ON\_RESTAURANT (  CID INT NOT NULL,   OID INT NOT NULL,   PRIMARY KEY (CID, OID),  FOREIGN KEY (CID) REFERENCES COMPLAINT(CID),  FOREIGN KEY (OID) REFERENCES RESTAURANT\_OUTLET(OID) ); |      | COMPLAINTS\_ON\_SHOP(CID, SID) | | --- | | CREATE TABLE COMPLAINTS\_ON\_SHOP (  CID INT NOT NULL,   SID INT NOT NULL,   PRIMARY KEY (CID, SID),  FOREIGN KEY (CID) REFERENCES COMPLAINT(CID),  FOREIGN KEY (SID) REFERENCES SHOP(SID) ); |      | VOUCHER(VID, Date\_issued, Description, Status, Expiry\_date) | | --- | | CREATE TABLE VOUCHER (  VID INT NOT NULL,   Date\_issued DATETIME NOT NULL,   Description TEXT,  Status TEXT NOT NULL,   Expiry\_date DATETIME NOT NULL,  PRIMARY KEY(VID) ); |      | PURCHASE\_VOUCHER(VID, Purchase\_discount, UID, Date\_time) | | --- | | CREATE TABLE PURCHASE\_VOUCHER (  VID INT NOT NULL,   Purchase\_discount FLOAT NOT NULL,   UID INT NOT NULL,   Date\_time DATETIME NOT NULL,  FOREIGN KEY (VID) REFERENCES VOUCHER(VID) ON DELETE CASCADE,  FOREIGN KEY (UID) REFERENCES USER\_ACCOUNT(UID) ON DELETE CASCADE,  PRIMARY KEY(VID) ); |  | DINE\_VOUCHER(VID, Cash\_discount, UID, Date\_time) | | --- | | CREATE TABLE DINE\_VOUCHER (  VID INT NOT NULL,   Cash\_discount FLOAT NOT NULL,   UID INT NOT NULL,   Date\_time DATETIME NOT NULL,  FOREIGN KEY (VID) REFERENCES VOUCHER(VID) ON DELETE CASCADE,  FOREIGN KEY (UID) REFERENCES USER\_ACCOUNT(UID) ON DELETE CASCADE,  PRIMARY KEY(VID) ); |      | GROUP\_VOUCHER(VID, Group\_size, Group\_discount, UID, Date\_time) | | --- | | CREATE TABLE GROUP\_VOUCHER (  VID INT NOT NULL,   Group\_size INT NOT NULL,   Group\_discount FLOAT NOT NULL,   UID INT NOT NULL,   Date\_time DATETIME NOT NULL,  FOREIGN KEY (VID) REFERENCES VOUCHER(VID) ON DELETE CASCADE,  FOREIGN KEY (UID) REFERENCES USER\_ACCOUNT(UID) ON DELETE CASCADE,  PRIMARY KEY(VID) ); |      | PACKAGE\_VOUCHER(VID, Package\_discount, UID, Date\_time) | | --- | | CREATE TABLE PACKAGE\_VOUCHER (  VID INT NOT NULL,   Package\_discount FLOAT NOT NULL,   UID INT NOT NULL,   Date\_time DATETIME NOT NULL,  FOREIGN KEY (VID) REFERENCES VOUCHER(VID) ON DELETE CASCADE,  FOREIGN KEY (UID) REFERENCES USER\_ACCOUNT(UID) ON DELETE CASCADE,  PRIMARY KEY(VID) ); |  | RECOMMENDATION(NID, Valid\_period, Date\_issued, MID, Dine\_voucher, Purchase\_voucher, DID, OID) | | --- | | CREATE TABLE RECOMMENDATION (  NID INT NOT NULL,   Valid\_period INT NOT NULL,   Date\_issued DATETIME NOT NULL,   MID INT NOT NULL,   Dine\_voucher INT NOT NULL,   Purchase\_voucher INT NOT NULL,   DID INT NOT NULL,   OID INT NOT NULL,  FOREIGN KEY (MID) REFERENCES MALL(MID),  FOREIGN KEY (Dine\_voucher) REFERENCES DINE\_VOUCHER(VID),  FOREIGN KEY (Purchase\_voucher) REFERENCES PURCHASE\_VOUCHER(VID),  FOREIGN KEY (DID) REFERENCES DAY\_PACKAGE(DID),  FOREIGN KEY (OID) REFERENCES RESTAURANT\_OUTLET(OID),  PRIMARY KEY(NID) ); |      | USER\_RELATED\_USER(UID1, UID2, Type, GID) | | --- | | CREATE TABLE USER\_RELATED\_USER (  UID1 INT NOT NULL,   UID2 INT NOT NULL,   Type TEXT NOT NULL,  GID INT NOT NULL,   PRIMARY KEY (UID1, UID2),  FOREIGN KEY (UID1) REFERENCES USER\_ACCOUNT(UID),  FOREIGN KEY (UID2) REFERENCES USER\_ACCOUNT(UID) ); |      | USER\_USE\_RECOMMENDATION(UID, NID) | | --- | | CREATE TABLE USER\_USE\_RECOMMENDATION (  UID INT NOT NULL,   NID INT NOT NULL,   PRIMARY KEY (UID, NID),  FOREIGN KEY (UID) REFERENCES USER\_ACCOUNT(UID),  FOREIGN KEY (NID) REFERENCES RECOMMENDATION(NID) ); |      | DAY\_PACKAGE\_HAS\_MALL(DID, MID) | | --- | | CREATE TABLE DAY\_PACKAGE\_HAS\_MALL (  DID INT NOT NULL,   MID INT NOT NULL,   PRIMARY KEY (DID, MID),  FOREIGN KEY (DID) REFERENCES DAY\_PACKAGE(DID),  FOREIGN KEY (MID) REFERENCES MALL(MID) ); |      | DAY\_PACKAGE\_HAS\_RESTAURANT\_OUTLET(DID, OID) | | --- | | CREATE TABLE DAY\_PACKAGE\_HAS\_RESTAURANT\_OUTLET (  DID INT NOT NULL,   OID INT NOT NULL,   PRIMARY KEY (DID, OID),  FOREIGN KEY (DID) REFERENCES DAY\_PACKAGE(DID),  FOREIGN KEY (OID) REFERENCES RESTAURANT\_OUTLET(OID) ); | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

SQL Statements to Solve the Queries in Appendix B

| 1. Find the most popular day packages(product/service) where all participants are related to one another as either family members or members of the same club. |
| --- |
| SELECT productid, count(productid) AS COUNT FROM day\_package dpb WHERE (dpb.did IN  (SELECT did  FROM day\_package dp0  WHERE productid IN  (SELECT dp1.productid  FROM day\_package dp1  GROUP BY dp1.productid  EXCEPT  (SELECT dp2.productid  FROM day\_package dp2  JOIN day\_package dp3 ON dp2.date = dp3.date  AND dp2.productid = dp3.productid  AND dp2.uid <> dp3.uid  JOIN user\_related\_user uru ON ((uru.uid1 = dp2.uid  AND uru.uid2 = dp3.uid)  OR (uru.uid2 = dp2.uid  AND uru.uid1 = dp3.uid))  WHERE cast(uru.type AS varchar(255)) NOT IN ('family', 'club')  GROUP BY dp2.productid)) ))  AND (did NOT IN  (SELECT dp4.did  FROM day\_package dp4  EXCEPT  (SELECT dp2.did  FROM day\_package dp2  JOIN day\_package dp3 ON dp2.[date] = dp3.[date]  AND dp2.productid = dp3.productid  AND dp2.uid <> dp3.uid  GROUP BY dp2.did))) GROUP BY productid ORDER BY count(productid) DESC; |
| Query Output:    Brief explanation:   1. The main idea is to first, find all the day packages and MINUS (EXCEPT) the day packages whereby some participants are neither ‘family’ nor ‘club’. The results of this subquery would be the ProductID of the day packages whereby all participants are either ‘family’ or ‘club’. 2. We then count the number of tuples in DAY\_PACKAGE whereby ProductID is equal to the ones found in (1), but exclude those day packages whereby there are only one person in it, and order them in descending order. Although the question is asking for the “MOST”, we did not use the TOP 1 keyword because we wanted our query to show 3-5 tuples as output.   Assumptions made:   1. Popularity is based on how many people signed up for the day package. 2. Tuples in the DAY-PACKAGE table with the same ProductID, will mean the same day package. However, they may have different dates. 3. Different dates would represent different “groups”, one day package may have numerous groups. 4. There can only be 1 group going for the day package on each calendar date. 5. As long as everyone in the group is either ‘family’ or ‘club’, it will add to the popularity count. 6. For groups with only 1 person in it, we will exclude it from the count. |

| 2. Find families who frequently shopped and dined together, with or without day packages. As part of your output, indicate whether these families use day packages or not. “frequently” means at least 50% of the time. |
| --- |
| SELECT total\_count\_go\_out.uid,  us.name,  total\_count\_go\_out.count\_together,  total\_count\_go\_out.count\_total,  total\_count\_go\_out.percentage,  total\_count\_dp.count\_dp FROM  (SELECT count\_all\_go\_out.uid,  count\_total,  count\_together,  cast(sq10.count\_together AS FLOAT) / cast(count\_all\_go\_out.count\_total AS FLOAT) AS percentage  FROM  (SELECT UID,  sum(all\_count) AS count\_total  FROM (  (SELECT UID,  count(\*) AS all\_count  FROM shopping  GROUP BY UID)  UNION  (SELECT UID,  count(\*) AS all\_count  FROM dining  GROUP BY UID)) AS ss1  GROUP BY UID) AS count\_all\_go\_out  LEFT OUTER JOIN  (SELECT UID,  sum(count\_together) AS count\_together  FROM (  (SELECT UID,  count(date\_time\_in) AS count\_together  FROM  (SELECT DISTINCT d1.\*  FROM dining d1  JOIN dining d2 ON d1.uid <> d2.uid  AND d1.oid = d2.oid  AND d1.date\_time\_in = d2.date\_time\_in  JOIN (  (SELECT uru1.uid2,  uru1.gid  FROM user\_related\_user uru1  JOIN user\_related\_user uru2 ON uru1.gid = uru2.gid  WHERE cast(uru1.type AS varchar(255)) IN ('family'))  UNION  (SELECT uru1.uid1,  uru1.gid  FROM user\_related\_user uru1  JOIN user\_related\_user uru2 ON uru1.gid = uru2.gid  WHERE cast(uru2.type AS varchar(255)) IN ('family'))) AS sq1 ON d1.uid = sq1.uid2  JOIN (  (SELECT uru1.uid2,  uru1.gid  FROM user\_related\_user uru1  JOIN user\_related\_user uru2 ON uru1.gid = uru2.gid  WHERE cast(uru1.type AS varchar(255)) IN ('family'))  UNION  (SELECT uru1.uid1,  uru1.gid  FROM user\_related\_user uru1  JOIN user\_related\_user uru2 ON uru1.gid = uru2.gid  WHERE cast(uru2.type AS varchar(255)) IN ('family'))) AS sq2 ON d2.uid = sq2.uid2  WHERE sq1.gid = sq2.gid ) AS ss1  GROUP BY ss1.uid)  UNION ALL  (SELECT ss2.uid,  count(\*) AS count\_together  FROM  (SELECT DISTINCT s1.uid  FROM shopping s1  JOIN shopping s2 ON s1.uid <> s2.uid  AND s1.sid = s2.sid  AND s1.date\_time\_in = s2.date\_time\_in  JOIN (  (SELECT uru1.uid2,  uru1.gid  FROM user\_related\_user uru1  JOIN user\_related\_user uru2 ON uru1.gid = uru2.gid  WHERE cast(uru1.type AS varchar(255)) IN ('family'))  UNION  (SELECT uru1.uid1,  uru1.gid  FROM user\_related\_user uru1  JOIN user\_related\_user uru2 ON uru1.gid = uru2.gid  WHERE cast(uru2.type AS varchar(255)) IN ('family'))) AS sq1 ON s1.uid = sq1.uid2  JOIN (  (SELECT uru1.uid2,  uru1.gid  FROM user\_related\_user uru1  JOIN user\_related\_user uru2 ON uru1.gid = uru2.gid  WHERE cast(uru1.type AS varchar(255)) IN ('family'))  UNION  (SELECT uru1.uid1,  uru1.gid  FROM user\_related\_user uru1  JOIN user\_related\_user uru2 ON uru1.gid = uru2.gid  WHERE cast(uru2.type AS varchar(255)) IN ('family'))) AS sq2 ON s2.uid = sq2.uid2  WHERE sq1.gid = sq2.gid) AS ss2  GROUP BY ss2.uid)) AS sss1  GROUP BY UID) AS sq10 ON sq10.uid = count\_all\_go\_out.uid) total\_count\_go\_out LEFT OUTER JOIN  (SELECT UID,  sum(dp\_count) AS count\_dp  FROM (  (SELECT si1.uid,  count(\*) AS dp\_count  FROM  (SELECT dg.\*  FROM dining dg  JOIN day\_package dp ON dg.uid = dp.uid  AND convert(date, dg.[date\_time\_in]) = convert(date, dp.[date])) AS si1  GROUP BY si1.uid)  UNION ALL  (SELECT si2.uid,  count(\*) AS dp\_count  FROM  (SELECT sp.\*  FROM shopping sp  JOIN day\_package dp ON sp.uid = dp.uid  AND convert(date, sp.[date\_time\_in]) = convert(date, dp.[date])) AS si2  GROUP BY si2.uid)) AS ss1  GROUP BY UID) AS total\_count\_dp  ON total\_count\_go\_out.uid = total\_count\_dp .uid  JOIN USER\_ACCOUNT us ON us.UID = total\_count\_go\_out.uid  WHERE total\_count\_go\_out.percentage > 0.5; |
| Query Output:    Brief explanation:  (We are focusing on user 5, 6 and 7 for this example.)  User 5, 6 and 7 belong to the same family.  User 5 dines together with user 7 on 31 March 2024.  User 5 shops together with user 6 on 31 March 2024.  User 5 shops together with user 7 on 31 March 2024.  User 7 shops alone 3 times.  Therefore we should expect the results to return user 5 and user 6 only.   1. Count\_total represents the total number of shopping/dining activities done by the user (be it alone, or with family). Count\_together represents the total number of shopping/dining activities done by the user WITH family. Percentage is Count\_together DIVIDED by Count\_total. 2. We first have to JOIN the SHOPPING table with USER\_RELATED\_USER table to find out if users are shopping with their family members. 3. We repeat (1) for the DINING table as well. 4. Afterwards, we sum up the frequencies whereby each user dines/shops together with their family. 5. We then join the resulting table with another table where we summed up the total number of times each user dined/shopped (either alone or with a family). 6. By dividing (3) with (4), we are able to filter out the users who shop/dine with their families more than 50% of the time. 7. We join the above with the DAY\_PACKAGE table again, so that for each user that dines/shops with their family more than 50% of the time, we find out if they are using day package (when count\_dp is greater than 0, it mean day package is used).   Assumptions:   1. Since one family has multiple members, and each member may either join the family, or shop/dine alone, it would be more appropriate to return the users within the families rather than the families itself. |

| 3. What are the most popular recommendations from the app regarding malls? |
| --- |
| SELECT [mid],  count(\*) AS recommendation\_count FROM recommendation GROUP BY [mid] ORDER BY recommendation\_count DESC; |
| Query Output:    Brief explanation:   1. GROUP BY MID means that the count of recommendations will be calculated for each unique Mall ID (MID). 2. After grouping them based on MID, we will count the number of tuples within each MID, this will give us the number of recommendations for each MID. 3. ORDER BY sorts the results based on the ‘Recommendation\_Count’ in descending order with the highest number of recommendations appearing at the top of the result set. |

| 4. Compulsive shoppers are those who have visited a certain mall more than 5 times within a certain period of time. Find the youngest compulsive shoppers and the amount they spent in total during December 2023. |
| --- |
| SELECT TOP 1 U.UID,  U.Name,  U.DOB,  Subquery2.TOTAL\_SPENT,  Subquery2.SHOPPING\_COUNT,  Subquery2.MID FROM  (SELECT \*  FROM  (SELECT SP.UID,  S.MID,  COUNT(S.MID) AS SHOPPING\_COUNT,  SUM(SP.Amount\_spent) AS TOTAL\_SPENT  FROM SHOPPING SP  JOIN SHOP S ON SP.SID = S.SID  WHERE SP.Date\_time\_in BETWEEN '2023-12-01' AND '2023-12-31'  GROUP BY SP.UID,  S.MID) AS Subquery1  WHERE SHOPPING\_COUNT > 4 ) AS Subquery2 JOIN USER\_ACCOUNT U ON Subquery2.UID = U.UID ORDER BY U.DOB DESC; |
| Query Output:    Brief explanation:   1. The first subquery (Subquery1) involves joining SHOP and SHOPPING so that we will be able to get the MID. Using the WHERE clause, we filtered it down by the date and GROUP BY UID and MID so that each tuple will represent the number of times a user visits a mall within the given period and the SUM function will give us the total amount spent by the user. 2. The second subquery (Subquery2) uses another WHERE clause to filter out only those tuples in Subquery1 that have a count of more than 4. 3. On the outermost query, we JOIN Subquery2 with USER\_ACCOUNT table so that we have access to the DOB of the user. By setting ORDER BY U.DOB DESC, we are able to obtain the YOUNGEST shopper that has shopped more than 5 times at a mall during the given period of time. |

| 5. Find users who have dined in all the restaurants in some malls, but have never dined in any restaurants in some other malls. |
| --- |
| SELECT us.uid,  [name] FROM user\_account us JOIN  (SELECT [uid]  FROM  (SELECT distinct(sq3.uid)  FROM  (SELECT sq1.uid,  sq1.mid,  count(DISTINCT sq1.oid) AS dine\_in\_same\_mall  FROM  (SELECT dg.uid,  dg.oid,  ro.mid  FROM dining AS dg  JOIN restaurant\_outlet AS ro ON dg.oid = ro.oid) AS sq1  GROUP BY sq1.uid,  sq1.mid) AS sq3  JOIN  (SELECT ro.mid,  count(ro.oid) AS number\_of\_restaurants  FROM restaurant\_outlet ro  GROUP BY ro.mid) AS sq2 ON sq3.mid = sq2.mid  AND sq2.number\_of\_restaurants = sq3.dine\_in\_same\_mall  GROUP BY sq3.uid) AS s1  EXCEPT SELECT distinct(sq13.uid)  FROM  (SELECT sq11.uid,  sq11.mid,  count(DISTINCT sq11.oid) AS dine\_in\_same\_mall  FROM  (SELECT dg.uid,  dg.oid,  ro.mid  FROM dining AS dg  JOIN restaurant\_outlet AS ro ON dg.oid = ro.oid) AS sq11  GROUP BY sq11.uid,  sq11.mid) AS sq13  JOIN  (SELECT ro.mid,  count(ro.oid) AS number\_of\_restaurants  FROM restaurant\_outlet ro  GROUP BY ro.mid) AS sq12 ON sq13.mid = sq12.mid  AND sq12.number\_of\_restaurants <> sq13.dine\_in\_same\_mall  GROUP BY sq13.uid) AS s2 ON us.uid = s2.uid; |
| Query Output:    Brief explanation:  (How we insert tuples into the database)  MallA has Restaurant 1 and 2  MallB has Restaurant 3 and 4  MallC has Restaurant 5 and 6  MallD has Restaurant 7 and 8  User1 dined in Restaurants 1, 2, 3, 4, 5 and 6.  User2 dined in Restaurants 1, 2, 3, 4 and 5.  User3 dined in Restaurants 1, 2, 1, 2, 1 and 2 again.  In this example, User1 and User3 should be the results of the query, but not User2 because he dined in Restaurant5 and not Restaurant6.   1. The main idea is to find UIDs that have dined in every restaurant in the mall and MINUS (EXCEPT) UIDs that have dined in less than the number of restaurants in the mall. 2. To achieve the above, we first JOIN DINING with RESTAURANT\_OUTLET so we have access to the MID. This MID will then be used to JOIN with RESTAURANT\_OUTLET again, but this time it has computed the number of restaurants for each MID using the GROUP BY MID and COUNT function. 3. The result of the above is essentially the DINING table with the addition of the corresponding number of restaurants in the mall the restaurant belongs to. 4. We then GROUP BY UID and MID, such that we will be able to find the number of DISTINCT restaurants each user has dined in each mall, and check if that is EQUAL to the number of restaurants in the same mall. This gives us the first part mentioned in (1). 5. We repeat steps 2-4, except this time we find the UIDs whereby the number of DISTINCT restaurants dined in each mall is NOT EQUAL to the number of restaurants in the same mall. This gives us the second part mentioned in (2). 6. We do an EXCEPT between (4) and (5) to obtain the UIDs of those that have dined in all the restaurants in some malls, but have never dined in any restaurants in another mall. 7. The outermost query is simply to JOIN the immediate subquery with the USER\_ACCOUNT table so that we have access to the user’s name. |

| 6. What are the top 3 highest earning malls and restaurants? |
| --- |
| Top 3 highest earning malls  SELECT top 3 m.mid,  m.address,  sum(sp.amount\_spent) AS total\_amount\_spent FROM mall m JOIN shop sh ON m.mid = sh.mid JOIN shopping sp ON sh.sid = sp.sid GROUP BY m.mid,  m.address ORDER BY total\_amount\_spent DESC; |
| Top 3 highest earning restaurants  SELECT top 3 r.name,  sum(d.amount\_spent) AS total\_amount\_spent FROM restaurant\_outlet r JOIN dining d ON r.oid = d.oid GROUP BY r.name ORDER BY total\_amount\_spent DESC; |
| Query Output:    Brief explanation:  Malls   1. We join SHOPPING with SHOP so that we are able to identify which mall the shopping has taken place in, then we JOIN it with MALL so that we are able to get the Address of the mall. We then use GROUP BY MID and Address and SUM function to find how much money is spent shopping in each MID. 2. With the TOP 3 and ORDER BY DESC keywords, we are able to find the top 3 highest earning malls.   Restaurants   1. We join DINING with RESTAURANT\_OUTLET so that we are able to get the Name of the restaurant. We then use GROUP BY Name and SUM function to find how much money is spent dining in each restaurant. 2. With the TOP 3 and ORDER BY DESC keywords, we are able to find the top 3 highest earning restaurants. |

SQL Statements to Solve the Additional Queries given in class

| 7. Find restaurants that are more popular for young adults (18 to 35 years old) than for older adults (50 years or older). |
| --- |
| SELECT SQ.OID,  RO.Name,  SQ.YOUNG\_ADULTS,  SQ.OLD\_ADULTS FROM  (SELECT COALESCE(YOUNGQUERY.OID, OLDQUERY.OID) AS OID,  COALESCE(YOUNG\_ADULTS, 0) AS YOUNG\_ADULTS,  COALESCE(OLD\_ADULTS, 0) AS OLD\_ADULTS  FROM  (SELECT d.OID,  COUNT(\*) AS YOUNG\_ADULTS  FROM DINING D  JOIN USER\_ACCOUNT UA ON D.UID = UA.UID  WHERE DATEDIFF(YEAR, UA.DOB, GETDATE()) BETWEEN 18 AND 35  GROUP BY d.OID) AS YOUNGQUERY  FULL OUTER JOIN  (SELECT d.OID,  COUNT(\*) AS OLD\_ADULTS  FROM DINING D  JOIN USER\_ACCOUNT UA ON D.UID = UA.UID  WHERE DATEDIFF(YEAR, UA.DOB, GETDATE()) >= 50  GROUP BY d.OID) OLDQUERY ON YOUNGQUERY.OID = OLDQUERY.OID) AS SQ JOIN RESTAURANT\_OUTLET RO ON SQ.OID = RO.OID WHERE SQ.YOUNG\_ADULTS > SQ.OLD\_ADULTS ORDER BY SQ.OLD\_ADULTS DESC; |
| Query Output:    Brief explanation:   1. In subquery YOUNGQUERY, we find the count of diners who are aged 18 to 35, and group them based on the restaurant’s OID. 2. In subquery OLDQUERY, we find the count of diners who are aged 50 and above, and group them based on the restaurant’s OID. 3. We then do a FULL OUTER JOIN between these 2 queries, and assign all NULL values as 0 so that a comparison can be done in the later steps. This joined query will be SQ. 4. SQ is then joined with RESTAURANT\_OUTLET table so that we have access to the restaurant’s names. 5. Lastly, we add in a WHERE clause whereby we filter the output based on the restaurants that have a higher YOUNG\_ADULT count than the OLD\_ADULT count. |

| Printout of all table records   | USER\_ACCOUNT(UID, Gender, DOB, Name) | | --- | |  |  | SHOP(SID, Type, MID, Shop\_Manager\_ID, Shop\_Manager\_Name) | | --- | |  |  | SHOPPING(SID, UID, Amount\_spent, Date\_time\_in, Date\_time\_out) | | --- | |  |  | DINING(UID, OID, Amount\_spent, Date\_time\_in, Date\_time\_out) | | --- | |  |  | MALL(MID, Address, NumShops, CID) | | --- | |  |  | MALL\_MGMT\_COMPANY(CID, Address) | | --- | |  |  | RESTAURANT\_OUTLET(OID, Unit\_no, MID, RID, Type, Name, Floor) | | --- | |  |  | RESTAURANT\_CHAIN(RID, Address) | | --- | |  |      | DAY\_PACKAGE (DID, Description, VID, UID, Date, ProductID) | | --- | |  |      | COMPLAINT (CID,Text,Status,Filed\_date\_time,UID) | | --- | |  |      | COMPLAINTS\_ON\_RESTAURANT (CID, OID) | | --- | |  |      | COMPLAINTS\_ON\_SHOP(CID, SID) | | --- | |  |      | VOUCHER(VID, Date\_issued, Description, Status, Expiry\_date) | | --- | |  |      | PURCHASE\_VOUCHER(VID, Purchase\_discount, UID, Date\_time) | | --- | |  |  | DINE\_VOUCHER(VID, Cash\_discount, UID, Date\_time) | | --- | |  |      | GROUP\_VOUCHER(VID, Group\_size, Group\_discount, UID, Date\_time) | | --- | |  |      | PACKAGE\_VOUCHER(VID, Package\_discount, UID, Date\_time) | | --- | |  |  | RECOMMENDATION(NID, Valid\_period, Date\_issued, MID, Dine\_voucher, Purchase\_voucher, DID, OID) | | --- | |  |      | USER\_RELATED\_USER(UID1, UID2, Type, GID) | | --- | |  |      | USER\_USE\_RECOMMENDATION(UID, NID) | | --- | |  |      | DAY\_PACKAGE\_HAS\_MALL(DID, MID) | | --- | |  |      | DAY\_PACKAGE\_HAS\_RESTAURANT\_OUTLET(DID, OID) | | --- | |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |