LIBRARY

```
CREATE TABLE PUBLISHER
(NAME VARCHAR2 (20) PRIMARY KEY, PHONE INTEGER,
ADDRESS VARCHAR2 (20));
```

```
CREATE TABLE BOOK
(BOOK_ID INTEGER PRIMARY KEY,
TITLE VARCHAR2 (20),
PUB_YEAR VARCHAR2 (20),
PUBLISHER_NAME REFERENCES PUBLISHER (NAME) ON DELETE
CASCADE);
```

```
CREATE TABLE BOOK_AUTHORS
(AUTHOR_NAME VARCHAR2 (20),
BOOK_ID REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID,
AUTHOR_NAME));
```

```
CREATE TABLE LIBRARY_BRANCH (BRANCH_ID INTEGER PRIMARY KEY, BRANCH_NAME
VARCHAR2 (50), ADDRESS VARCHAR2 (50));
```

```
CREATE TABLE BOOK_COPIES
(NO_OF_COPIES INTEGER,
BOOK_ID REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, BRANCH_ID REFERENCES
LIBRARY_BRANCH (BRANCH_ID) ON CASCADE,
PRIMARY KEY (BOOK_ID, BRANCH_ID));
```

```
CREATE TABLE CARD
(CARD_NO INTEGER PRIMARY KEY);
```

```
CREATE TABLE BOOK_LENDING (DATE_OUT DATE,
DUE_DATE DATE,
BOOK_ID REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, BRANCH_ID REFERENCES
LIBRARY_BRANCH (BRANCH_ID) ON CASCADE,
CARD_NO REFERENCES CARD (CARD_NO) ON DELETE CASCADE, PRIMARY KEY
(BOOK_ID, BRANCH_ID, CARD_NO));
```

Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each branch, etc.

```
SELECT B.BOOK_ID, B.TITLE, B.PUBLISHER_NAME, A.AUTHOR_NAME, C.NO_OF_COPIES,
L.BRANCH_ID
FROM BOOK B, BOOK_AUTHORS A, BOOK_COPIES C, LIBRARY_BRANCH L WHERE
B.BOOK_ID=A.BOOK_ID
AND B.BOOK_ID=C.BOOK_ID
AND L.BRANCH_ID=C.BRANCH_ID;
```

Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun 2017.

```
SELECT CARD_NO
FROM BOOK_LENDING
WHERE DATE_OUT BETWEEN '01-JAN-2017' AND '01-JUL-2017' GROUP BY CARD_NO
HAVING COUNT (*)>3;
```

Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.

```
DELETE FROM BOOK WHERE BOOK_ID=3;
```

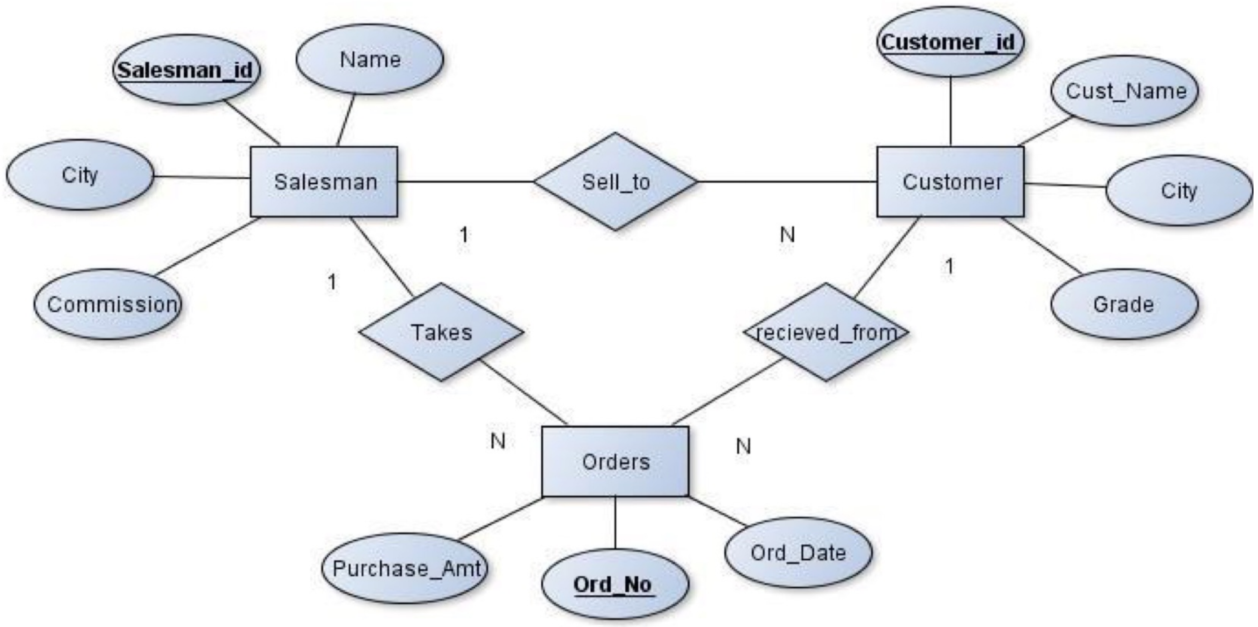
Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.

```
CREATE VIEW V_PUBLICATION AS SELECT PUB_YEAR
FROM BOOK;
```

Create a view of all books and its number of copies that are currently available in the Library.

```
CREATE VIEW V_BOOKS AS
SELECT B.BOOK_ID, B.TITLE, C.NO_OF_COPIES
FROM BOOK B, BOOK_COPIES C, LIBRARY_BRANCH L WHERE B.BOOK_ID=C.BOOK_ID
AND C.BRANCH_ID=L.BRANCH_ID;
```

SALESMAN



```
CREATE TABLE SALESMAN (SALESMAN_ID NUMBER (4),
NAME VARCHAR2 (20),
CITY VARCHAR2 (20), COMMISSION VARCHAR2 (20),
PRIMARY KEY
(SALESMAN_ID))
;
CREATE TABLE CUSTOMER1 (CUSTOMER_ID NUMBER (4),
CUST_NAME VARCHAR2 (20),
CITY VARCHAR2 (20),
GRADE NUMBER (3),
PRIMARY KEY (CUSTOMER_ID),
SALESMAN_ID REFERENCES SALESMAN (SALESMAN_ID)
ON DELETE SET NULL);
CREATE TABLE ORDERS (ORD_NO NUMBER (5),
PURCHASE_AMT NUMBER (10, 2), ORD_DATE DATE,
PRIMARY KEY (ORD_NO),
CUSTOMER_ID REFERENCES CUSTOMER1
(CUSTOMER_ID) ON DELETE CASCADE,

SALESMAN_ID REFERENCES SALESMAN (SALESMAN_ID) ON DELETE CASCADE);
```

Count the customers with grades above Bangalore’s average.

```
SELECT GRADE, COUNT (DISTINCT CUSTOMER_ID) FROM CUSTOMER1
GROUP BY GRADE
HAVING GRADE > (SELECT AVG(GRADE) FROM CUSTOMER1
WHERE CITY='BANGALORE');
```

Find the name and numbers of all salesmen who had more than one customer.

```
SELECT SALESMAN_ID, NAME FROM SALESMAN A
WHERE 1 < (SELECT COUNT (*)
FROM CUSTOMER1
WHERE SALESMAN_ID=A.SALESMAN_ID);
```

List all salesmen and indicate those who have and don’t have customers in their cities (Use UNION operation.)

```
SELECT SALESMAN.SALESMAN_ID, NAME, CUST_NAME, COMMISSION FROM SALESMAN,
CUSTOMER1
WHERE SALESMAN.CITY = CUSTOMER1.CITY
UNION
SELECT SALESMAN_ID, NAME, 'NO MATCH', COMMISSION FROM SALESMAN
WHERE NOT CITY = ANY
(SELECT CITY FROM CUSTOMER1)
ORDER BY 2 DESC;
```

Create a view that finds the salesman who has the customer with the highest order of a day.

```
CREATE VIEW ELITSALESMAN AS
SELECT B.ORD_DATE, A.SALESMAN_ID, A.NAME FROM SALESMAN A, ORDERS B

 WHERE A.SALESMAN_ID = B.SALESMAN_ID
 AND B.PURCHASE_AMT=(SELECT MAX (PURCHASE_AMT)
 FROM ORDERS C
 WHERE C.ORD_DATE = B.ORD_DATE);
```

Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.
Use ON DELETE CASCADE at the end of foreign key definitions while creating child table orders and then execute the following:
Use ON DELETE SET NULL at the end of foreign key definitions while creating child table customers and then executes the following:

```
DELETE FROM SALESMAN WHERE SALESMAN_ID=1000;
```

Actor

```
CREATE TABLE ACTOR ( ACT_ID NUMBER (3), ACT_NAME VARCHAR (20),
ACT_GENDER CHAR (1), PRIMARY KEY (ACT_ID));
CREATE TABLE DIRECTOR ( DIR_ID NUMBER (3), DIR_NAME VARCHAR (20),
DIR_PHONE NUMBER (10), PRIMARY KEY (DIR_ID));
CREATE TABLE MOVIES
( MOV_ID NUMBER (4),
MOV_TITLE VARCHAR (25),
MOV_YEAR NUMBER (4),
MOV_LANG VARCHAR (12),
DIR_ID NUMBER (3),
PRIMARY KEY (MOV_ID),
FOREIGN KEY (DIR_ID) REFERENCES DIRECTOR (DIR_ID));

CREATE TABLE MOVIE_CAST ( ACT_ID NUMBER (3),
MOV_ID NUMBER (4),
ROLE VARCHAR (10),
PRIMARY KEY (ACT_ID, MOV_ID),
FOREIGN KEY (ACT_ID) REFERENCES ACTOR (ACT_ID), FOREIGN KEY (MOV_ID)
REFERENCES MOVIES (MOV_ID));
CREATE TABLE RATING (
MOV_ID NUMBER (4),
REV_STARS VARCHAR (25),
PRIMARY KEY (MOV_ID),
FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));
```

List the titles of all movies directed by ‘Hitchcock’.

```
SELECT MOV_TITLE
FROM MOVIES
WHERE DIR_ID IN (SELECT DIR_ID
FROM DIRECTOR
WHERE DIR_NAME = ‘HITCHCOCK’);
```

Find the movie names where one or more actors acted in two or more movies.

```
SELECT MOV_TITLE
FROM MOVIES M, MOVIE_CAST MV
WHERE M.MOV_ID=MV.MOV_ID AND ACT_ID IN (SELECT ACT_ID
FROM MOVIE_CAST GROUP BY ACT_ID HAVING COUNT (ACT_ID)>1)
GROUP BY MOV_TITLE HAVING COUNT (*)>1;
```

List all actors who acted in a movie before 2000 and also in a movie after 2015 (use JOIN operation).

```
SELECT ACT_NAME, MOV_TITLE, MOV_YEAR

FROM ACTOR A JOIN MOVIE_CAST C
ON A.ACT_ID=C.ACT_ID JOIN MOVIES M
ON C.MOV_ID=M.MOV_ID
WHERE M.MOV_YEAR NOT BETWEEN 2000 AND 2015;
OR
SELECT A.ACT_NAME, A.ACT_NAME, C.MOV_TITLE, C.MOV_YEAR FROM
ACTOR A, MOVIE_CAST B, MOVIES C
WHERE A.ACT_ID=B.ACT_ID
AND B.MOV_ID=C.MOV_ID
AND C.MOV_YEAR NOT BETWEEN 2000 AND 2015;
```

Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.

```
SELECT MOV_TITLE, MAX (REV_STARS) FROM MOVIES
INNER JOIN RATING USING (MOV_ID) GROUP BY MOV_TITLE
HAVING MAX (REV_STARS)>0 ORDER BY MOV_TITLE;
```

Update rating of all movies directed by ‘Steven Spielberg’ to 5
KL

```
UPDATE RATING
SET REV_STARS=5
WHERE MOV_ID IN (SELECT MOV_ID FROM MOVIES
WHERE DIR_ID IN (SELECT DIR_ID FROM DIRECTOR
WHERE DIR_NAME = ‘STEVEN SPIELBERG’));
```