CMR
INSTITUTE
OF TECHNOLOGY

SCHEME OF EVALUATION

Internal Assessment Test 2 – Jan-Feb. 2024

Sub:	**DATABASE MANAGEMENT SYSTEMS**			**Code:**	21CS53
Date: 1/2/2024	Duration: 90 mins	Max Marks: 50	**Sem:** 5	**Branch:**	AIML

Note: ***Answer any five full questions.*** ***(5 X 10 = 50)***

Question No.		**Description**	**Marks Split up**		**Total Marks**
1.		✓ Steps for ER to relational mapping	10	10M	10M
2.	a)	✓ Constraints name of relational model	6	6M	10M
	b)	✓ Dynamic SQL ✓ Embedded SQL	2 2	4M	
3.	a)	✓ Trigger ✓ Assertion	5 5	10M	10M
4.		✓ Normalization ✓ First Normal Form ✓ Second Normal Form ✓ Third Normal Form	1 3 3 3	10M	10M
5.	a)	✓ Informal Design Guidelines	6M	6M	10M
	Bbb) b b)	i. Multivalued Dependency ii. Fourth Normal Form	2 2	$444M	
6.		✓ First Query ✓ Second Query ✓ Third Query ✓ Fourth Query ✓ Fifth Query	2 2 2 2 2		10M

USN | | | | | | | | | | |

Internal Assessment Test 2 – Jan-Feb 2024

Sub:	**Database Management Systems**					Sub Code:	**21CS53**	Branch:	**AIML**
Date:	**1/2/24**	Duration:	**90 minutes**	Max Marks:	**50**	Sem/Sec:	**V**		**OBE**

		Answer any FIVE FULL Questions	**MARKS**	**CO**	**RBT**
1		Briefly describe the steps involved in ER- to relational mapping algorithm.	[10]	2	L3
2	a	Explain the different relational model constraints.	[6]	2	L1
	b	What is Dynamic SQL and how is it different from embedded SQL?	[4]	3	L1
3		How trigger and assertion can be defined in SQL? Explain with example.	[10]	3	L1
4		Define Normalization.Explain 1NF, 2NF and 3NF with suitable examples for each.	[10]	4	L2
5	a	Explain the informal design guidelines used as measures to determine the quality of relation schema design.	[6]	4	L2
	b	Define multivalued dependency. Explan fourth normal form with an example.	[4]	4	L1
6		Consider the following schema for OrderDatabase SALESMAN (Salesman_id, Name, City, Commission) CUSTOMER (Customer_id, Cust_Name, City, Grade,Salesman_id) ORDERS (Ord_No, Purchase_Amt, Ord_Date, Customer_id, Salesman_id) Write SQL queries for the following: 1. Count the customers with grades above Bangalore's average. 2. Find the name and numbers of all salesmen who had more than one customer. 3. List all salesmen and indicate those who have and don't have customers in their cities (Use UNION operation.) 4.Create a view that finds the salesman who has the customer with the highest order of a day. 5.Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.	[10]	3	L3

CI **CCI** **HoD**

----------All the Best----------

Q1. Briefly describe the steps involved in ER- to relational mapping algorithm

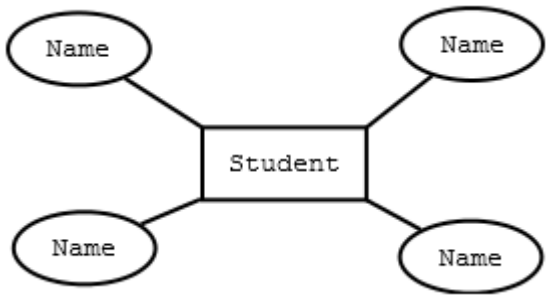
- Ans: ER Model when conceptualized into diagrams gives a good overview of entity-relationship, which is easier to understand.
- ER diagrams can be mapped to Relational schema using step by step procedure.
- Though all the ER constraints cannot be imported into Relational model but an approximate schema can be generated.

ER Diagrams mainly comprised of:

- Entity and its attributes
- Relationship which is association among entities

Mapping Entity

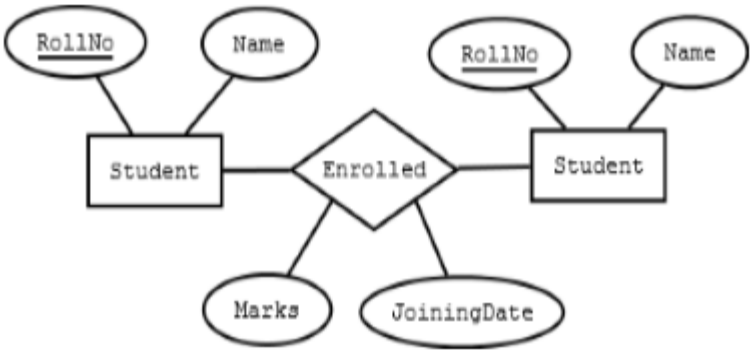
An entity is a real world object with some attributes.



- Create table for each entity
- Entity's attributes should become fields of tables with their respective data types.
- Declare primary key

Mapping relationship

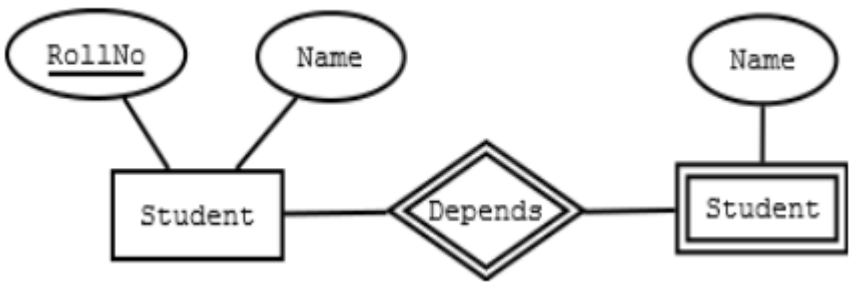
A relationship is association among entities.



- Create table for a relationship
- Add the primary keys of all participating Entities as fields of table with their respective data types.
- If relationship has any attribute, add each attribute as field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

Mapping Weak Entity Sets

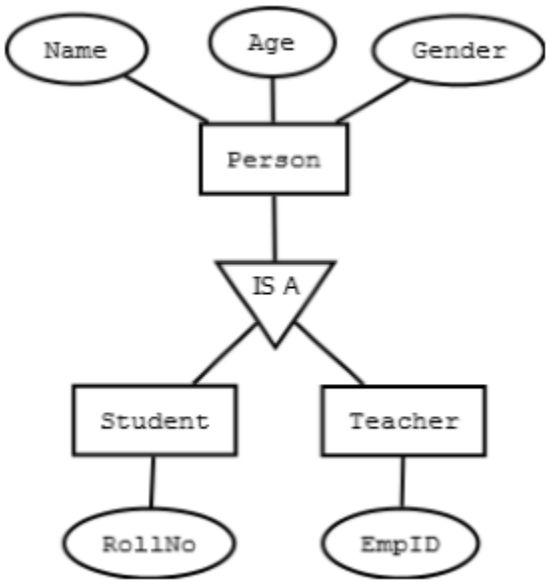
A weak entity sets is one which does not have any primary key associated with it.



- Create table for weak entity set
- Add all its attributes to table as field
- Add the primary key of identifying entity set
- Declare all foreign key constraints

Mapping hierarchical entities

ER specialization or generalization comes in the form of hierarchical entity sets.



- Create tables for all higher level entities
- Create tables for lower level entities
- Add primary keys of higher level entities in the table of lower level entities
- In lower level tables, add all other attributes of lower entities.
- Declare primary key of higher level table the primary key for lower level table

Q2(a). Explain the different relational model constraints.

Ans: 1. Domain Constraints

- Every domain must contain atomic values(smallest indivisible units) which means composite and multi-valued attributes are not allowed.
- We perform a datatype check here, which means when we assign a data type to a column we limit the values that it can contain. Eg. If we assign the datatype of attribute age as int, we can't give it values other than int datatype.

Example:

EID	Name	Phone
01	Bikash Dutta	123456789 234456678

Explanation: In the above relation, Name is a composite attribute and Phone is a multi-values attribute, so it is violating domain constraint.

2. Key Constraints or Uniqueness Constraints

- These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
- A relation can have multiple keys or candidate keys(minimal superkey), out of which we choose one of the keys as the primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes.
- Null values are not allowed in the primary key, hence Not Null constraint is also part of the key constraint.

Example:

EID	Name	Phone
01	Bikash	6000000009
02	Paul	9000090009
01	Tuhin	9234567892

Explanation: In the above table, EID is the primary key, and the first and the last tuple have the same value in EID ie 01, so it is violating the key constraint.

3. Entity Integrity Constraints

- Entity Integrity constraints say that no primary key can take a NULL value, since using the primary key we identify each tuple uniquely in a relation.

Example:

EID	Name	Phone
01	Bikash	9000900099
02	Paul	600000009
NULL	Sony	9234567892

Explanation: In the above relation, EID is made the primary key, and the primary key can't take NULL values but in the third tuple, the primary key is null, so it is violating Entity Integrity constraints.

4. Referential Integrity Constraints

- The Referential integrity constraint is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.
- This constraint is enforced through a foreign key, when an attribute in the foreign key of relation R1 has the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.
- The values of the foreign key in a tuple of relation R1 can either take the values of the primary key for some tuple in relation R2, or can take NULL values, but can't be empty.

Example:

EID	Name	DNO
01	Divine	12
02	Dino	22
04	Vivian	14

DNO	Place
12	Jaipur
13	Mumbai
14	Delhi

Explanation: In the above tables, the DNO of Table 1 is the foreign key, and DNO in Table 2 is the primary key. DNO = 22 in the foreign key of Table 1 is not allowed because DNO = 22 is not defined in the primary key of table 2. Therefore, Referential integrity constraints are violated here.

Q2 (b) What is Dynamic SQL and how is it different from embedded SQL?

Ans : Embedded SQL

Embedded SQL is a method that combines SQL with a high−level programming language's features. It enables programmers to put SQL statements right into the source code files used to set up an application. Database operations may be carried out effortlessly by developers by adding SQL statements to the application code. The source code files having embedded SQL statements should be preprocessed before compilation because of the issue of interpretation of SQL statements by the high−level programming languages in embedded SQL. The terms EXEC SQL and END_EXEC must be used before and after each SQL statement in the source code file. In embedded SQL, host variables play a key role. These variables serve as an intermediary for data transfer between the application and the database. There are two different kinds of host variables: input host variables that provide data to the database and output host variables that receive that data.

Example

This example shows the embedded code written in C++, retrieving the customer id, and name from the database.

Input

Student

id	roll_no	name	address
1	21	monu	gonda

Example

```
int main(){
/* connecting to database */
EXEC SQL CONNECT student;

/* declaring variables */
EXEC SQL BEGIN DECLARE SECTION;
int id;
int roll_no;
char name[10];
char address[30];
EXEC SQL END DECLARE SECTION;

/* set up error processing */
EXEC SQL WHENEVER SQLWARNING DO display_warning();
EXEC SQL WHENEVER SQLERROR STOP;
EXEC SQL WHENEVER NOT FOUND GOTO lbl_no_records;

/* Execute the sql query */
EXEC SQL SELECT * FROM STUDENT WHERE STUDENT_ID = :STD_ID;

/* Display result */
printf("id: %d", id);
printf("name: %d", name);
exit();
```

Dynamic SQL

Dynamic SQL involves the creation and execution of SQL statements at runtime. Dynamic SQL allows developers to generate SQL statements dynamically based on runtime conditions or user input. By combining changeable data, conditions, and dynamic database or column names, developers may quickly construct SQL queries using dynamic SQL. Because of its adaptability, dynamic SQL is a good choice when the SQL statements need to change in response to evolving needs or user inputs. Dynamic SQL queries are built at execution time so the system chooses how to access the database and conduct the SQL queries. Performance could be affected as a result of this lack of preparation because the system must create an execution plan on the spot. Dynamic SQL, however, provides outstanding adaptability and versatility.

Steps to use Dynamic SQL

Step 1: Declare two variables

```
DECLARE
@var1 NVARCHAR(MAX),
@var2 NVARCHAR(MAX);
```

Step 2: Set the value of the first variable as table_name

```
SET @var1 = N'table_name';
```

Step 3: Select statement is added to table_name to create dynamic SQL

```
SET @var2= N'SELECT * FROM ' + @var1;
```

Step 4: Use the second variable to run the sp_executesql

Q3 How trigger and assertion can be defined in SQL? Explain with example.

Solution

What are Assertions?

When a constraint involves 2 (or) more tables, the table constraint mechanism is sometimes hard and results may not come as expected. To cover such situation SQL supports the creation of assertions that are constraints not associated with only one table. And an assertion statement should ensure a certain condition will always exist in the database. DBMS always checks the assertion whenever modifications are done in the corresponding table.

Syntax –

CREATE ASSERTION [assertion_name]

CHECK ([condition]);

Example –

CREATE TABLE sailors (sid int,sname varchar(20), rating int,primary key(sid),

CHECK(rating >= 1 AND rating <=10)

CHECK((select count(s.sid) from sailors s) + (select count(b.bid)from boats b)<100));

In the above example, we enforcing CHECK constraint that the number of boats and sailors should be less than 100. So here we are able to CHECK constraints of two tablets simultaneously.

2. What are Triggers?

A trigger is a database object that is associated with the table, it will be activated when a defined action is executed for the table. The trigger can be executed when we run the following statements:

1. INSERT
2. UPDATE
3. DELETE

And it can be invoked before or after the event.

Syntax –

create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]

Example –

```
create trigger t1  before  UPDATE on sailors

for each row

begin

  if new.age>60 then

    set new.age=old.age;

  else

    set new.age=new.age;

  end if;

end;

$
```

Q4. Define Normalization.Explain 1NF, 2NF and 3NF with suitable examples for each.

Ans. Normalization is the process of organizing the data and the attributes of a database. It is performed to reduce the data redundancy in a database and to ensure that data is stored logically. Data redundancy in DBMS means having the same data but at multiple places. It is necessary to remove data redundancy because it causes anomalies in a database which makes it very hard for a database administrator to maintain it.

First Normal Form (1NF)

A relation is in 1NF if every attribute is a single-valued attribute or it does not contain any multi-valued or composite attribute, i.e., every attribute is an atomic attribute. If there is a composite or multi-valued attribute, it violates the 1NF. To solve this, we can create a new row for each of the values of the multi-valued attribute to convert the table into the 1NF.

Let's take an example of a relational table <EmployeeDetail> that contains the details of the employees of the company.

<EmployeeDetail>

Employee Code	Employee Name	Employee Phone Number
101	John	98765623,998234123
101	John	89023467
102	Ryan	76213908
103	Stephanie	98132452

Here, the Employee Phone Number is a multi-valued attribute. So, this relation is not in 1NF.

To convert this table into 1NF, we make new rows with each Employee Phone Number as a new row as shown below:

<EmployeeDetail>

Employee Code	Employee Name	Employee Phone Number
101	John	998234123
101	John	98765623
101	John	89023467
102	Ryan	76213908
103	Stephanie	98132452

Second Normal Form (2NF)

The normalization of 1NF relations to 2NF involves the elimination of partial dependencies. A partial dependency in DBMS exists when any non-prime attributes, i.e., an attribute not a part of the candidate key, is not fully functionally dependent on one of the candidate keys.

For a relational table to be in second normal form, it must satisfy the following rules:

1. The table must be in first normal form.
2. It must not contain any partial dependency, i.e., all non-prime attributes are fully functionally dependent on the primary key.

If a partial dependency exists, we can divide the table to remove the partially dependent attributes and move them to some other table where they fit in well.

Let us take an example of the following <EmployeeProjectDetail> table to understand what is partial dependency and how to normalize the table to the second normal form:

<EmployeeProjectDetail>

Employee Code	Project ID	Employee Name	Project Name
101	P03	John	Project103
101	P01	John	Project101
102	P04	Ryan	Project104
103	P02	Stephanie	Project102

In the above table, the prime attributes of the table are Employee Code and Project ID. We have partial dependencies in this table because Employee Name can be determined by Employee Code and Project Name can be determined by Project ID. Thus, the above relational table violates the rule of 2NF.

The prime attributes in DBMS are those which are part of one or more candidate keys.

To remove partial dependencies from this table and normalize it into second normal form, we can decompose the <EmployeeProjectDetail> table into the following three tables:

<EmployeeDetail>

Employee Code	Employee Name
101	John
101	John
102	Ryan
103	Stephanie

<EmployeeProject>

Employee Code	Project ID
101	P03
101	P01
102	P04
103	P02

<ProjectDetail>

Project ID	Project Name
P03	Project103
P01	Project101
P04	Project104
P02	Project102

Thus, we've converted the <EmployeeProjectDetail> table into 2NF by decomposing it into <EmployeeDetail>, <ProjectDetail> and <EmployeeProject> tables. As you can see, the above tables satisfy the following two rules of 2NF as they are in 1NF and every non-prime attribute is fully dependent on the primary key.

The relations in 2NF are clearly less redundant than relations in 1NF. However, the decomposed relations may still suffer from one or more anomalies due to the transitive dependency. We will remove the transitive dependencies in the Third Normal Form.

Third Normal Form (3NF)

The normalization of 2NF relations to 3NF involves the elimination of transitive dependencies in DBMS.

A functional dependency X -> Z is said to be transitive if the following three functional dependencies hold:

- X -> Y
- Y does not -> X
- Y -> Z

For a relational table to be in third normal form, it must satisfy the following rules:

1. The table must be in the second normal form.
2. No non-prime attribute is transitively dependent on the primary key.
3. For each functional dependency X -> Z at least one of the following conditions hold:

- X is a super key of the table.
- Z is a prime attribute of the table.

If a transitive dependency exists, we can divide the table to remove the transitively dependent attributes and place them to a new table along with a copy of the determinant.

Let us take an example of the following <EmployeeDetail> table to understand what is transitive dependency and how to normalize the table to the third normal form:

<EmployeeDetail>

Employee Code	Employee Name	Employee Zipcode	Employee City
101	John	110033	Model Town
101	John	110044	Badarpur
102	Ryan	110028	Naraina
103	Stephanie	110064	Hari Nagar

The above table is not in 3NF because it has Employee Code -> Employee City transitive dependency because:

- Employee Code -> Employee Zipcode
- Employee Zipcode -> Employee City

Also, Employee Zipcode is not a super key and Employee City is not a prime attribute.

To remove transitive dependency from this table and normalize it into the third normal form, we can decompose the <EmployeeDetail> table into the following two tables:

<EmployeeDetail>

Employee Code	Employee Name	Employee Zipcode
101	John	110033
101	John	110044
102	Ryan	110028
103	Stephanie	110064

<EmployeeLocation>

Employee Zipcode	Employee City
110033	Model Town
110044	Badarpur
110028	Naraina
110064	Hari Nagar

Thus, we've converted the <EmployeeDetail> table into 3NF by decomposing it into <EmployeeDetail> and <EmployeeLocation> tables as they are in 2NF and they don't have any transitive dependency.

The 2NF and 3NF impose some extra conditions on dependencies on candidate keys and remove redundancy caused by that. However, there may still exist some dependencies that cause redundancy in the database. These redundancies are removed by a more strict normal form known as BCNF

Q5(a). Explain the informal design guidelines used as measures to determine the quality of relation schema design.
Ans.

INFORMAL DESIGN GUIDELINES FOR RELATIONAL
SCHEMA 1.Semantics of the Attributes
2.Reducing the Redundant Value in Tuples.
3.Reducing Null values in Tuples.
4.Dissallowing spurious Tuples.

1. **Semantics of the Attributes**
Whenever we are going to form relational schema there should be some meaning among the attributes.This meaning is called semantics.This semantics relates one attribute to another with some relation.
2. **Reducing the Redundant Value in Tuples**
Mixing attributes of multiple entities may cause problems Information is stored redundantly wasting storage Problems with update anomalies Insertion anomalies Deletion anomalies Modification anomalies.
3. Reducing Null values in Tuples.
Note: Relations should be designed such that their tuples will have as few NULL values as possible Attributes that are NULL frequently could be placed in separate relations (with the primary key) Reasons for nulls: attribute not applicable or invalid attribute value unknown (may exist) value known to exist, but unavailable

4. Disallowing spurious Tuples
Bad designs for a relational database may result in erroneous results for certain JOIN operations The "lossless join" property is used to guarantee meaningful results for join operations Note: The relations should be designed to satisfy the lossless join condition. No spurious tuples should be generated by doing a natural-join o

Q5(b) Define multivalued dependency. Explan fourth normal form with an example.
Ans.

Multivalued dependency would occur whenever two separate attributes in a given table happen to be independent of each other. And yet, both of these depend on another third attribute. The multivalued dependency contains at least two of the attributes dependent on the third attribute. This is the reason why it always consists of at least three of the attributes.

Example

Suppose that there is a car manufacturing company that produces two of the colours in the market, i.e., red and yellow of each of their models, every year.

CAR_MODEL	MANUF_MONTH	COLOUR
S2011	JAN	Yellow
S2001	FEB	Red
S3001	MAR	Yellow
S3001	APR	Red
S4006	MAY	Yellow
S4006	JUN	Red

In this case, the columns COLOUR and MANUF_MONTH are dependent on CAR_MODEL, and they are independent of each other. Thus, we can call both of these columns multivalued. These are, as a result, dependent on CAR_MODEL. Here is a representation of the dependencies we discussed above:

CAR_MODEL → → MANUF_MONTH

CAR_MODEL → → COLOUR

We can read this as “CAR_MODEL multidetermined MANUF_MONTH” and “CAR_MODEL multidetermined COLOUR”.

Fourth Normal Form (4NF)

The Fourth Normal Form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF, and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Properties

A relation R is in 4NF if and only if the following conditions are satisfied:

1. It should be in the Boyce-Codd Normal Form (BCNF).
2. The table should not have any Multi-valued Dependency.

A table with a multivalued dependency violates the normalization standard of the Fourth Normal Form (4NF) because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this up to 4NF, it is necessary to break this information into two
tables.

Example: Consider the database table of a class that has two relations R1 contains student ID(SID) and student name (SNAME) and R2 contains course id(CID) and course name (CNAME).

Table R1

SID	SNAME
S1	A
S2	B

Table R2

CID	CNAME
C1	C
C2	D

When their cross-product is done it resulted in multivalued dependencies.

Table R1 X R2

SID	SNAME	CID	CNAME
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

Multivalued dependencies (MVD) are:
SID->->CID; SID->->CNAME; SNAME->->CNAME

6. Consider the following schema for OrderDatabase

SALESMAN (Salesman_id, Name, City, Commission)

CUSTOMER (Customer_id, Cust_Name, City, Grade,Salesman_id)

ORDERS (Ord_No, Purchase_Amt, Ord_Date, Customer_id, Salesman_id)

Write SQL queries for the following:

1. Count the customers with grades above Bangalore's average.
2. Find the name and numbers of all salesmen who had more than one customer.
3. List all salesmen and indicate those who have and don't have customers in their cities (Use UNION operation.)
4. Create a view that finds the salesman who has the customer with the highest order of a day.
5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.

Ans.

1. Count the customers with grades above Bangalore's average

```
select grade,count(distinct c_id)from customer group by grade having grade>(select avg(grade) from customer where
city='Banglore');
```

2. Find the name and numbers of all salesman who had more thanone customer.

```
select salesmanid,name from salesman S where(select count(*) from customer C where C.salesmanid=S.salesmanid)>1;
```

3. List all the salesman and indicate those who have and don't havecustomers in their cities (Use UNION operation.)

```
select S.salesmanid,S.name,C.c_name,S.commission from salesman S,customer C where S.city=c.city union select
S.salesmanid,S.name,'No match',S.commission from salesman S where city not in(Select city from customer)order by 1 asc;
```

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

```
create view V_salesman as select O.order_date,S.salesmanid,S.name from salesman S,orders O where
S.salesmanid=O.salesmanid and O.purchase_amt=(Select max(purchase_amt) from orders C);
```

5. Demonstrate the DELETE operation by removing salesman withid 1000. All his orders must also be deleted.

```
delete from customer where salesmanid=1000;
delete salesmanid=1000;from orders where salesmanid=1000;
delete from salesman where
```