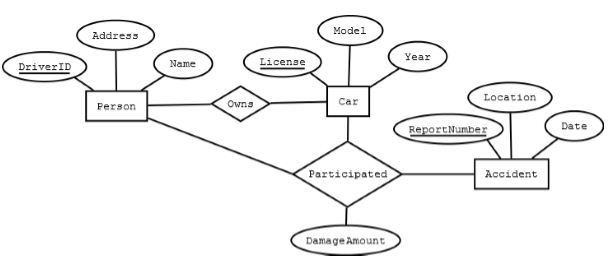
**1. Construct an E-R diagram for a car insurance company whose customers own one or more carseach. Each car has associated with it zero to any number of recorded accidents. Each insurancepolicy covers one or more cars, and has one or more premium payments associated with it. Eachpayment is for a particular period of time and has an associated due date and the date when thepayment was received**.



**2. What is E-R model? Draw an E-R Diagram for any Banking enterprise system.**

A. [ER diagram](https://www.geeksforgeeks.org/introduction-of-er-model/) is known as Entity-Relationship diagram. It is used to analyze to structure of the Database. It shows relationships between entities and their attributes. An ER model provides a means of communication.

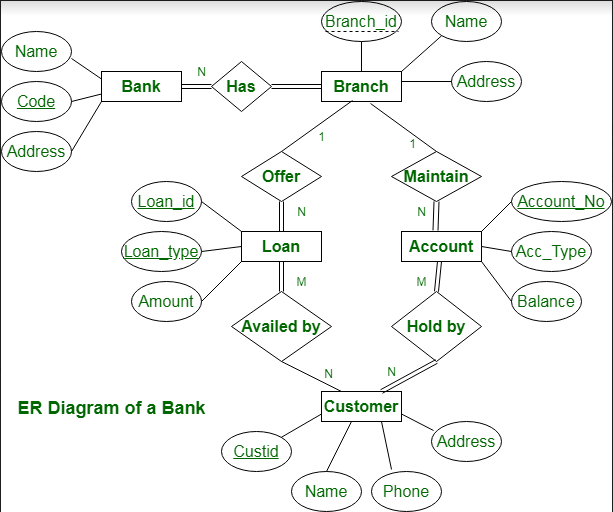
ER diagram of Bank has the following description : 

* Bank have Customer.
* Banks are identified by a name, code, address of main office.
* Banks have branches.
* Branches are identified by a branch \_no., branch \_name, address.
* Customers are identified by name, cust-id, phone number, address.
* Customer can have one or more accounts.
* Accounts are identified by account\_no., acc\_type, balance.
* Customer can avail loans.
* Loans are identified by loan\_id, loan\_type and amount.
* Account and loans are related to bank’s branch.

**Entities** and their **Attributes** are :

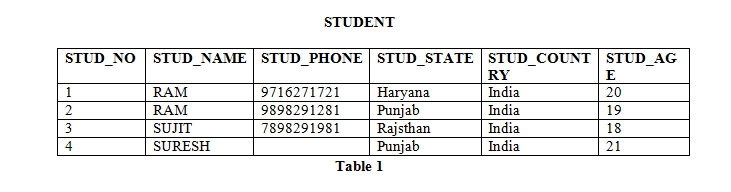
* **Bank Entity :** Attributes of Bank Entity are Bank Name, Code and Address.   
  Code is Primary Key for Bank Entity.
* **Customer Entity :** Attributes of Customer Entity are Customer\_id, Name, Phone Number and Address.   
  Customer\_id is Primary Key for Customer Entity.
* **Branch Entity :** Attributes of Branch Entity are Branch\_id, Name and Address.   
  Branch\_id is Primary Key for Branch Entity.
* **Account Entity :** Attributes of Account Entity are Account\_number, Account\_Type and Balance.   
  Account\_number is Primary Key for Account Entity.

**Loan Entity :** Attributes of Loan Entity are Loan\_id, Loan\_Type and Amount.   
 Loan\_id is Primary Key for Loan Entity. [Relationships](https://www.geeksforgeeks.org/attributes-to-relationships-in-er-model/) are : 

* **Bank has Branches => 1 : N**  
  One Bank can have many Branches but one Branch can not belong to many Banks, so the relationship between Bank and Branch is one to many relationship.
* **Branch maintain Accounts => 1 : N**   
  One Branch can have many Accounts but one Account can not belong to many Branches, so the relationship between Branch and Account is one to many relationship.
* **Branch offer Loans => 1 : N**   
  One Branch can have many Loans but one Loan can not belong to many Branches, so the relationship between Branch and Loan is one to many relationship.
* **Account held by Customers => M : N**   
  One Customer can have more than one Accounts and also One Account can be held by one or more Customers, so the relationship between Account and Customers is many to many relationship.
* **Loan availed by Customer => M : N**   
  (Assume loan can be jointly held by many Customers).   
  One Customer can have more than one Loans and also One Loan can be availed by one or more Customers, so the relationship between Loan and Customers is many to many relationship.
* 

**3. What is meant by closure of F? Where F is the set of functional dependencies. Explaincomputing F+ with suitable examples**

* A. The Closure Of Functional Dependency means**the complete set of all possible attributes that can be functionally derived from given functional dependency** using the inference rules known as Armstrong’s Rules. If “F” is a functional dependency then closure of functional dependency can be denoted using “{F}+”.
* A functional dependency A->B in a relation holds if two tuples having same value of attribute A also have same value for attribute B. For Example, in relation STUDENT shown in table 1, Functional Dependencies
* STUD\_NO->STUD\_NAME, STUD\_NO->STUD\_PHONE **hold**
* but
* STUD\_NAME->STUD\_STATE **do not hold**



Functional Dependencies in a relation are dependent on the domain of the relation. Consider the STUDENT relation given in Table 1. 

* We know that STUD\_NO is unique for each student. So STUD\_NO->STUD\_NAME, STUD\_NO->STUD\_PHONE, STUD\_NO->STUD\_STATE, STUD\_NO->STUD\_COUNTRY and STUD\_NO -> STUD\_AGE all will be true.
* Similarly, STUD\_STATE->STUD\_COUNTRY will be true as if two records have same STUD\_STATE, they will have same STUD\_COUNTRY as well.
* For relation STUDENT\_COURSE, COURSE\_NO->COURSE\_NAME will be true as two records with same COURSE\_NO will have same COURSE\_NAME.
* **Functional Dependency Set:**Functional Dependency set or FD set of a relation is the set of all FDs present in the relation. For Example, FD set for relation STUDENT shown in table 1 is:
* { STUD\_NO->STUD\_NAME, STUD\_NO->STUD\_PHONE, STUD\_NO->STUD\_STATE, STUD\_NO->STUD\_COUNTRY,
* STUD\_NO -> STUD\_AGE, STUD\_STATE->STUD\_COUNTRY }
* **Attribute Closure:**Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.

**How to find attribute closure of an attribute set?**   
To find attribute closure of an attribute set: 

* Add elements of attribute set to the result set.
* Recursively add elements to the result set which can be functionally determined from the elements of the result set.

Using FD set of table 1, attribute closure can be determined as: 

(STUD\_NO)+ = {STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY, STUD\_AGE}

(STUD\_STATE)+ = {STUD\_STATE, STUD\_COUNTRY}

* If attribute closure of an attribute set contains all attributes of relation, the attribute set will be super key of the relation.
* If no subset of this attribute set can functionally determine all attributes of the relation, the set will be candidate key as well. For Example, using FD set of table 1,

(STUD\_NO, STUD\_NAME)+ = {STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY, STUD\_AGE}

(STUD\_NO)+ = {STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY, STUD\_AGE}

(STUD\_NO, STUD\_NAME) will be super key but not candidate key because its subset (STUD\_NO)+ is equal to all attributes of the relation. So, STUD\_NO will be a candidate key.

**4. Discuss view serializability in transactions**.

A. [View serializability](https://www.geeksforgeeks.org/view-serializability-in-dbms-transactions/) is a concept that is used to compute whether schedules are View-Serializable or not. A schedule is said to be **View-Serializable**if it is view equivalent to a **Serial Schedule** (where no interleaving of transactions is possible).

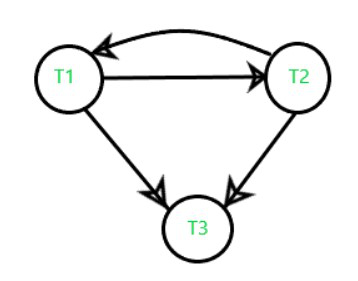
There may be some schedules that are not Conflict-Serializable but still gives a consistent result because the concept of [Conflict-Serializability](https://www.geeksforgeeks.org/conflict-serializability-in-dbms/) becomes limited when the [**Precedence Graph**](https://www.geeksforgeeks.org/precedence-graph-for-testing-Conflict-Serializability-in-dbms/) of a schedule contains a **loop/cycle.** In such a case we cannot predict whether a schedule would be consistent or inconsistent. As per the concept of Conflict-Serializability, We can say that a schedule is Conflict-Serializable (means serial and consistent) if its corresponding precedence graph does not have any loop/cycle.

But, what if a schedule’s precedence graph contains a cycle/loop and is giving consistent result/accurate result as a conflict serializable schedule is giving?   
So, to address such cases we brought the concept of**View-Serializability** because we did not want to confine the concept serializability only to Conflict-Serializability.

**Example :**Understanding View-Serializability first with a **Schedule S1 :**

|  |  |
| --- | --- |
|  |  |
| T1 | | T2 | T3 |
| a=100  **read(a)** | |  |  |
|  | | a=a-40  **write(a) //60** |  |
| a=a-40  **write(a) //20** | |  |  |
|  | |  | a=a-20  **write(a) //0** |

So, its Conflict Precedence Graph is as follows – 

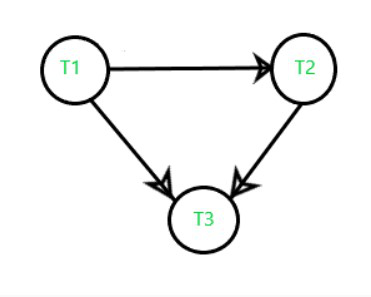


 The above graph contains cycle/loop which means it is not conflict-serializable but it does not mean that it cannot be consistent and equivalent to the serial schedule it may or may not be.

**LookSchedule S’1 :**   
In the above example if we do **swapping** among some transaction’s operation so our table will look like –

| T1 | T2 | T3 |
| --- | --- | --- |
| a=100  **read(a) //100** |  |  |
| a=a-40  **write(a) //60** |  |  |
|  | a=a-40  **write(a) //20** |  |
|  |  | a=a-20  **write(a) //0** |

Its Precedence Graph is as follows – 



Now, we see that the precedence graph of the second table does not contain any cycle/loop, which means it is conflict serializable (equivalent to serial schedule, consistent) and the final result is coming the same as the first table.

In the above example we understood that if a schedule is Conflict-serializable so we can easily predict that It would be – 

1. Equivalent to a serial schedule,
2. Consistent,
3. And also a View-Serializable.

**5 .Describe log-based recovery in brief.**

A. [Atomicity](https://www.geeksforgeeks.org/acid-properties-in-dbms/) property of DBMS states that either all the operations of transactions must be performed or none. The modifications done by an aborted transaction should not be visible to database and the modifications done by committed transaction should be visible.

To achieve our goal of atomicity, user must first output to stable storage information describing the modifications, without modifying the database itself. This information can help us ensure that all modifications performed by committed transactions are reflected in the database. This information can also help us ensure that no modifications made by an aborted transaction persist in the database.

**Log and log records –**  
The log is a sequence of log records, recording all the update activities in the database. In a stable storage, logs for each transaction are maintained. Any operation which is performed on the database is recorded is on the log. Prior to performing any modification to database, an update log record is created to reflect that modification.

An update log record represented as: <Ti, Xj, V1, V2> has these fields:

1. **Transaction identifier:** Unique Identifier of the transaction that performed the write operation.
2. **Data item:** Unique identifier of the data item written.
3. **Old value:** Value of data item prior to write.
4. **New value:** Value of data item after write operation.

Other type of log records are:

1. **<Ti start>**: It contains information about when a transaction Ti starts.
2. **<Ti commit>**: It contains information about when a transaction Ti commits.
3. **<Ti abort>**: It contains information about when a transaction Ti aborts.

**Undo and Redo Operations –**  
Because all database modifications must be preceded by creation of log record, the system has available both the old value prior to modification of data item and new value that is to be written for data item. This allows system to perform redo and undo operations as appropriate:

1. **Undo:** using a log record sets the data item specified in log record to old value.
2. **Redo:** using a log record sets the data item specified in log record to new value.

**The database can be modified using two approaches –**

1. **Deferred Modification Technique:** If the transaction does not modify the database until it has partially committed, it is said to use deferred modification technique.
2. **Immediate Modification Technique:** If database modification occur while transaction is still active, it is said to use immediate modification technique.

**Recovery using Log records –**  
After a system crash has occurred, the system consults the log to determine which transactions need to be redone and which need to be undone.

1. Transaction Ti needs to be undone if the log contains the record <Ti start> but does not contain either the record <Ti commit> or the record <Ti abort>.
2. Transaction Ti needs to be redone if log contains record <Ti start> and either the record <Ti commit> or the record <Ti abort>.

**Use of Checkpoints –**

When a system crash occurs, user must consult the log. In principle, that need to search the entire log to determine this information. There are two major difficulties with this approach:

1. The search process is time-consuming.
2. Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will cause recovery to take longer.

To reduce these types of overhead, user introduce checkpoints. A log record of the form <checkpoint L> is used to represent a checkpoint in log where L is a list of transactions active at the time of the checkpoint. When a checkpoint log record is added to log all the transactions that have committed before this checkpoint have <Ti commit> log record before the checkpoint record. Any database modifications made by Ti is written to the database either prior to the checkpoint or as part of the checkpoint itself. Thus, at recovery time, there is no need to perform a redo operation on Ti.

After a system crash has occurred, the system examines the log to find the last <checkpoint L> record. The redo or undo operations need to be applied only to transactions in L, and to all transactions that started execution after the record was written to the log. Let us denote this set of transactions as T. Same rules of undo and redo are applicable on T as mentioned in Recovery using Log records part.

Note that user need to only examine the part of the log starting with the last checkpoint log record to find the set of transactions T, and to find out whether a commit or abort record occurs in the log for each transaction in T. For example, consider the set of transactions {T0, T1, . . ., T100}. Suppose that the most recent checkpoint took place during the execution of transaction T67 and T69, while T68 and all transactions with subscripts lower than 67 completed before the checkpoint. Thus, only transactions T67, T69, . . ., T100 need to be considered during the recovery scheme. Each of them needs to be redone if it has completed (that is, either committed or aborted); otherwise, it was incomplete, and needs to be undone.

**6**. Draw ER diagram for university database consisting four entities Student, Department, Class andFaculty. Student has a unique id, the student can enroll for multiple classes and has a most onemajor. Faculty must belong to department and faculty can teach multiple classes. Each class istaught by only faculty. Every student will get grade for the class he/she has enrolled.

**A**. Drawing of ER model of university database application considering the constraints −

* A university has many departments.
* Each department has multiple instructors (one person is HOD). Here the HOD refers to the head of department.
* An instructor belongs to only one department.
* Each department offers multiple courses, each subject is taught by a single instructor.
* A student may enroll for many courses offered by different departments.

Step 1 − Identifying the entity sets.

The entity set has multiple instances in a given business scenario.

As per the given constraints the entity sets are as follows −

* Department
* Course
* Student
* Instructor

Head of the Department (HOD) is not an entity set. It is a relationship between the instructor and department entities.

### Step 2 − Identifying the attributes for the given entities

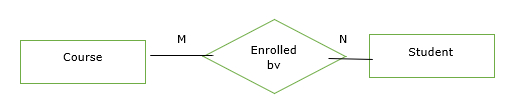
* Department − the relevant attributes are department Name and location.
* Course − The relevant attributes are courseNo, course Name, Duration, and prerequisite.
* Instructor − The relevant attributes are Instructor Name, Room No, and telephone number.
* Student − The relevant attributes are Student No, Student Name, and date of birth.

### Step 3 − Identifying the Key attributes

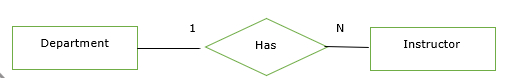
* Department Name is the key attribute for Department.
* CourseNo is the key attribute for Course entity.
* Instructor Name is the key attribute for the Instructor entity.
* StudentNo is the key attribute for Student entities.

### Step 4 − Identifying the relationship between entity sets

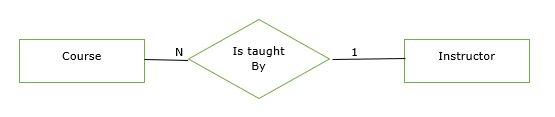
* The department offers multiple courses and each course belongs to only one department, hence cardinality between department and course if one to many.
*  One course is enrolled by multiple students and one student for multiple courses. Hence, relationships are many to many.



* One department has multiple instructors and one instructor belongs to one and only one department, hence the relationship is one to many.



* One course is taught by only one instructor but one instructor teaches many courses hence the relationship between course and instructor is many to one.



The relationship between instructor and student is not defined because of the following reasons −

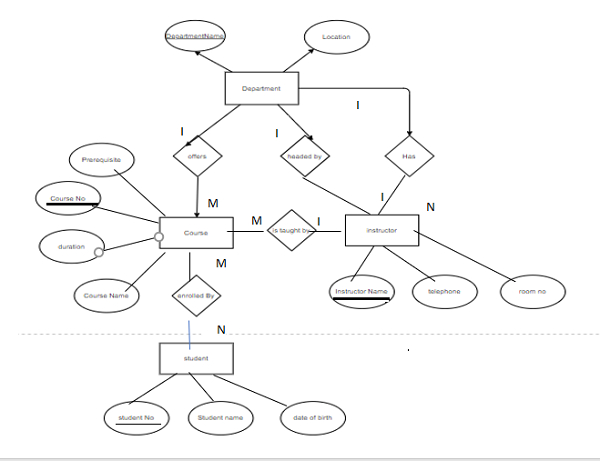
* There is no significance in the relationship.
* We can always derive this relationship indirectly through course and instructors, and course and student.

### Step 5 − Complete ER model

The complete ER Model is as follows −

Construct an Entity-Relationship diagram for a online shopping systems such as Jabong/Flipkart.

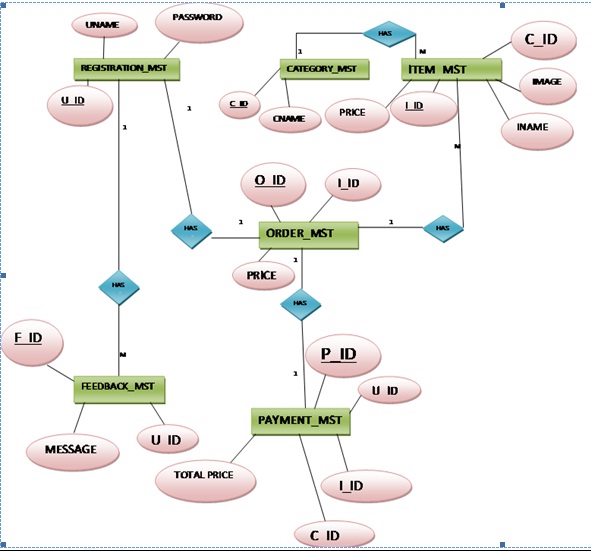
Quote your assumptions and list the requirements considered by you for conceptual database

design for the above system.

**7.** **Construct an Entity-Relationship diagram for a online shopping systems such as Jabong/Flipkart.Quote your assumptions and list the requirements considered by you for conceptual databasedesign for the above system.** **conceptual databasedesign for the above system.**

**A.** E-R diagram means Entity Relationship diagram. Entity means object of system, generally we refer entity as database table , the e-r diagram represent the relationship between each table of database. E-R diagram represent entity with attributes, attributes is a properties of entity. If we assume entity is a database table then all the columns of table are treat as attributes.

* Entities are represented by rectangles.
* Attributes are represented by ellipses.



**8.** **Explain deadlock with suitable scheduling examples.**

**A.*****Deadlock***is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



**Deadlock can arise if**the **following four conditions hold simultaneously (Necessary Conditions)**

***Mutual Exclusion:*** Two or more resources are non-shareable (Only one process can use at a time)   
***Hold and Wait:***A process is holding at least one resource and waiting for resources.   
***No Preemption:*** A resource cannot be taken from a process unless the process releases the resource.   
***Circular Wait:*** A set of processes are waiting for each other in circular form.

**Methods for handling deadlock**   
There are three ways to handle deadlock   
**1) Deadlock prevention or avoidance:**

**Prevention:**

The idea is to not let the system into a deadlock state. This system will make sure that above mentioned four conditions will not arise. These techniques are very costly so we use this in cases where our priority is making a system deadlock-free.  
One can zoom into each category individually, Prevention is done by negating one of above mentioned necessary conditions for deadlock. Prevention can be done in four different ways:

       1. Eliminate mutual exclusion                                        3. Allow preemption

       2. Solve hold and Wait                                                   4. Circular wait Solution

**Avoidance:**  
Avoidance is kind of futuristic. By using the strategy of “Avoidance”, we have to make an assumption. We need to ensure that all information about resources that the process will need is known to us before the execution of the process. We use Banker’s algorithm (Which is in turn a gift from Dijkstra) to avoid deadlock.

In prevention and avoidance, we get correctness of data but performance decreases.

**2) Deadlock detection and recovery:**If Deadlock prevention or avoidance is not applied to the software then we can handle this by deadlock detection and recovery. which consist of two phases:

1. In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
2. If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.

In Deadlock detection and recovery, we get the correctness of data but performance decreases.

**3) Deadlock ignorance:** If a deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take. we use the ostrich algorithm for deadlock ignorance.

In Deadlock, ignorance performance is better than the above two methods but the correctness of data .

**9.** **The relational database schema is given below.**

**employee (person-name, street, city)**

**works (person-name, company-name, salary)**

**company (company-name, city)**

**manages (person-name, manager-name)**

**Write the relational algebra expressions for the given queries**

**1. Find the names of all employees who work for First Bank Corporation.**

**2. Find the names and cities of residence of all employees who work for First Bank Corporation.**

**3. Find the names, street address, and cities of residence of all employees who work for FirstBank Corporation and earn more than $10,000 per annum.**

**4. Find the names of all employees in this database who do not work for First Bank Corporation.**

**A.1.** Find the names of all employees in the database who do not work for 'First Bank Corporation'.

Query:

select person-name from works where company-name = 'First Bank Corporation’

2, **Find the names and cities of residence of all employees who work for First Bank Corporation.**

select person-name and city from works where company-name = 'First Bank Corporation’

3.Find the names, street address, and cities of residence for all employees who work for 'First Bank Corporation' and earn more than $10,000.

Query:

select employee.employee-name, employee.street, employee.city from employee, works where employee.employee-name=works.employee-name and company-name = 'First Bank Corporation' and salary > 10000