**1 & 3 Describe what attributes represent in an ER model and provide examples of simple, composite, single-valued, multi-valued, and derived attributes.   
(Review Question 12.3 in 5th edition/ 11.3 in 4th edition)**

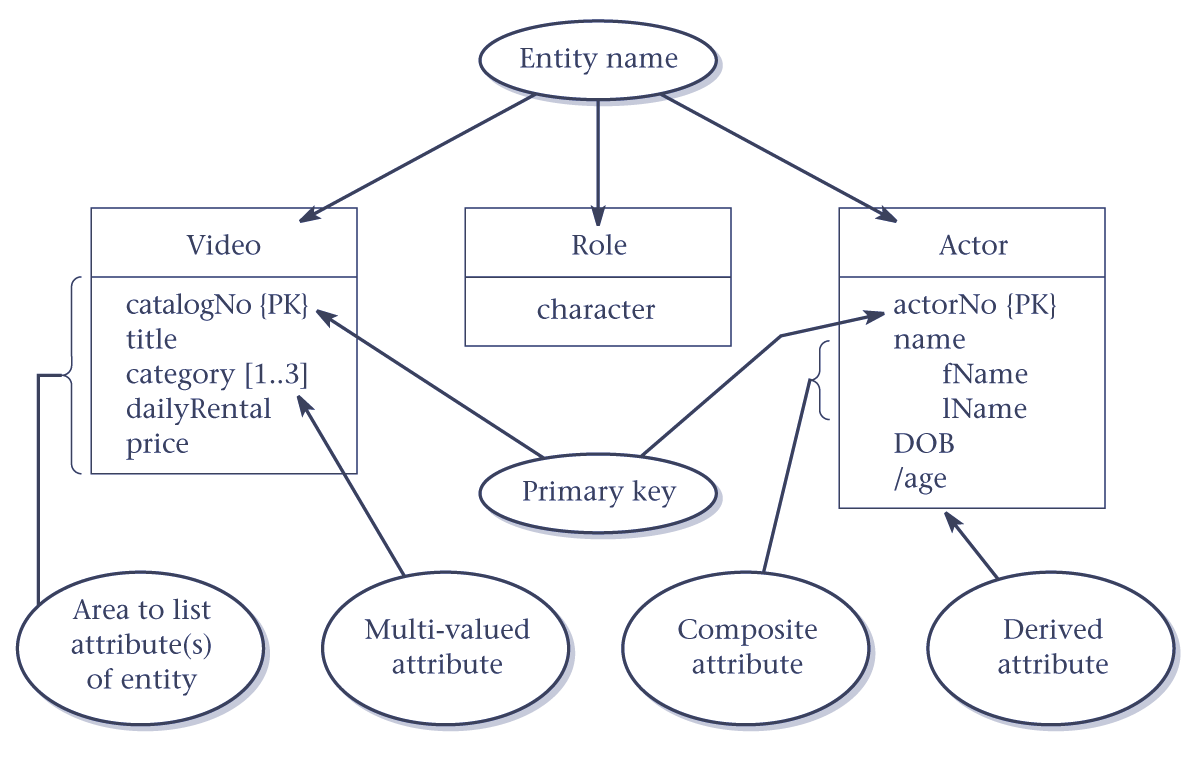
ANS: An attribute is a property of an entity or a relationship.

Attributes represent that we want to know about entities. For example, a Video entity may be described by the catalogNo, title, category, dailyRental and price attributes. These attributes hold values that describe each video occurrence and represent the main source of data stored in the database.

* 1. Simple attribute is an attribute composed of a single component. Simple attributes cannot be further subdivided. Example of simple attributes include the category and price attributes for a video.
  2. Composite attribute is an attribute composed of multiple components. Composite attributes can be further divided to yield smaller components with an independent existence. For example, the name attribute of the Member entity with the value ‘Don Nelson’ can be subdivided into fName (‘Don’) and lName (‘Nelson’).
  3. Single-valued attribute is an attribute that holds a single value for an entity occurrence. The majority of attributes are single-valued for a particular entity. For example, each occurrence of the Video entity has a single-value for the catalogNo attribute (for example, 207132), and therefore the catalogNo attribute is referred to as being single-valued.
  4. Multi-valued attribute is an attribute that holds multiple values for an entity occurrence. Some attributes have multiple values for a particular entity. For example, each occurrence of the Video entity may have multiple values for the category attribute (for example, ‘Children’ and ‘Comedy’), and therefore the category attribute in this case would be multi-valued. A multi-valued attribute may have a set of values with specified lower and upper limits. For example, the category attribute may have between one and three values.
  5. Derived attribute is an attribute that represents a value that is derivable from the value of a related attribute, or set of attributes, not necessarily in the same entity. Some attributes may be related for a particular entity. For example, the age of a member of staff (age) is derivable from the date of birth (DOB) attribute, and therefore the age and DOB attributes are related. We refer to the age attribute as a derived attribute, the value of which is derived from the DOB attribute.

**2 Describe how strong and weak entity types differ and provide an example of each.   
(Review Question 12.8 in 5th edition/ 11.8 in 4th edition)**

ANS: We can classify entities as being either strong or weak. A strong entity is not dependent on the existence of another entity for its primary key. A weak entity is partially or wholly dependent on the existence of another entity, or entities, for its primary key. For example, as we can distinguish one actor from all other actors and one video from all other videos without the existence of any other entity, Actor and Video are referred to as being strong entities. In other words, the Actor and Video entities are strong because they have their own primary keys. An example of a weak entity called Role, which represents characters played by actors in videos. If we are unable to uniquely identify one Role entity occurrence from another without the existence of the Actor and Video entities, then Role is referred to as being a weak entity. In other words, the Role entity is weak because it has no primary key of its own.



Diagrammatic representation of attributes for the Video, Role and Actor attributes

Strong entities are sometimes referred to as parent, owner, or dominant entities and weak entities as child, dependent, or subordinate entities.

**4 What are the differences between materialization and pipelining?**

ANS:

The difference between materialization and pipelining are as follows: The results of intermediate relational algebra operations are written temporarily to disk. This process is known as materialization: the output of one operation is stored in a temporary relation for processing by the next operation. An alternative approach is to pipeline the results of one operation to another operation without creating a temporary relation to hold the intermediate result. Clearly, if we can use pipelining we can save on the cost of creating temporary relations and reading the results back in again. So Materialization is the process of creating a temporary relation as the output of the stage which is passed to the next stage as input where as pipelining deals with projection and selection.

**5 The deadlock state can be changed back to stable state by using \_\_\_\_\_\_\_\_\_\_\_\_\_ statement.**

* 1. COMMIT
  2. ROLLBACK
  3. SAVEPOINT
  4. DEADLOCK

ANS: B

**6 Describe, with examples, the types of problem that can occur in a multi-user environment when concurrent access to the database is allowed.**ANS: The types of problem that can occur in a multi-user environment when concurrent access to the database is allowed are as follows:

* lost update problem
* uncommitted dependency
* inconsistent analysis problem

To illustrate these problems, we use a simple bank account relation that contains the DreamHome staff account balances. In this context, we are using the transaction as the unit of concurrency control.

An apparently successfully completed update operation by one user can be overridden by another user. This is known as the lost update problem. Let’s suppose a transaction T1 is executing concurrently with transaction T2. T1 is withdrawing £10 from an account with balance balx, initially £100, and T2 is depositing £100 into the same account. If these transactions are executed serially, one after the other with no interleaving of operations, the final balance would be £190 no matter which transaction is performed first. Transactions T1 and T2 start at nearly the same time, and both read the balance as £100. T2 increases balx by £100 to £200 and stores the update in the database. Meanwhile, transaction T1 decrements its copy of balx by £10 to £90 and stores this value in the database, overwriting the previous update, and thereby ‘losing’ the £100 previously added to the balance. The loss of T2’s update is avoided by preventing T1 from reading the value of balx until after T2’s update has been completed.

The uncommitted dependency problem occurs when one transaction is allowed to see the intermediate results of another transaction before it has committed. An uncommitted dependency that causes an error, using the same initial value for balance balx as in the previous example. Here, transaction T4 updates balx to £200, but it aborts the transaction so that balx should be restored to its original value of £100. However, by this time transaction T3 has read the new value of balx (£200) and is using this value as the basis of the £10 reduction, giving a new incorrect balance of £190, instead of £90. The value of balx read by T3 is called dirty data, giving rise to the alternative name, the dirty read problem.

The reason for the rollback is unimportant; it may be that the transaction was in error, perhaps crediting the wrong account. The effect is the assumption by T3 that T4’s update completed successfully, although the update was subsequently rolled back. This problem is avoided by preventing T3 from reading balx until after the decision has been made to either commit or abort T4’s effects.

The two problems in these examples concentrate on transactions that are updating the database and their interference may corrupt the database. However, transactions that only

read the database can also produce inaccurate results if they are allowed to read partial results of incomplete transactions that are simultaneously updating the database. We illustrate this with the next example.

The problem of inconsistent analysis occurs when a transaction reads several values from the database but a second transaction updates some of them during the execution of the first. For example, a transaction that is summarizing data in a database (for example, totaling balances) will obtain inaccurate results if, while it is executing, other transactions are updating the database. One example is illustrated in Figure 20.6, in which a summary transaction T6 is executing concurrently with transaction T5. Transaction T6 is totaling the balances of account x (£100), account y (£50), and account z (£25). However, in the meantime, transaction T5 has transferred £10 from balx to balz, so that T6 now has the wrong result (£10 too high). This problem is avoided by preventing transaction T6 from reading balx and balz until after T5 has completed its updates.

Another problem can occur when a transaction T rereads a data item it has previously read but, in between, another transaction has modified it. Thus, T receives two different values for the same data item. This is sometimes referred to as a nonrepeatable (or fuzzy) read. A similar problem can occur if transaction T executes a query that retrieves a set of tuples from a relation satisfying a certain predicate, re-executes the query at a later time but finds that the retrieved set contains an additional (phantom) tuple that has been inserted by another transaction in the meantime. This is sometimes referred to as a phantom read.

**7 3NF essentially identifies**

* 1. 1-\* relationships
  2. \* - \* relationships
  3. 1-1 relationships
  4. None of the above

ANS: A

**8 Explain what is meant by a transaction. Why are transactions important units of operation in a DBMS?**

ANS: Transaction is a sequence of operations that performs logical unit of operations on a database which include one or more operations as insert, delete, update and modification. Transaction is an executing program; the boundary of a transaction is defined as the explicit begin of transaction and end of transaction.

Transaction is the executing program that executes the operations on the database as read(x), write(x) so the transaction plays a vital role among database operations.