

GUI and Database Management System

Q.1.a Write advantages of DBMS over file system. (05)

Ans: Following are advantages of DBMS over file system:

1] Data isolation:

Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

2] Integrity problems:

The data values stored in the database must satisfy certain types of consistency constraint. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

3] Atomicity problems:

A computer system, like any other mechanical or electrical device, is subject to failure. In Many applications, it is difficult that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer \$50 from account A to account

B. If a system failure occurs during the execution of the program, it is possible that the \$50 was removed from account A but was not credited to account B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur or that neither occur. That is, the funds transfer must be atomic it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

4] Concurrent-access anomalies:

For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data.

5] Security problems:

Not every user of the database system should be able to access all the data. For example, in A banking system, payroll personnel need to see only that part of the database that has Information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult.

Q.1.b. Explain Murphy's law of GUI design with the help of example. (05)

Ans:

Murphy's law of GUI design:

- Law of Maximised Misunderstanding: It states that, "If something can possibly be misunderstood, it will be."

- Corollary: If it cannot possibly be misunderstood, it will still be misunderstood.

- User Identification Theorem: Everyone involved with software development accept the fact that GUIs should be best design to suit the users. Users can be developers , clients or actual end users. Hence , the software should be designed in such way that all the requirements are satisfied.

Example:

You can not make two-pin plug symmetrical and label it; "THIS WAY UP" if it matters which way it is plugged in, then you make the design asymmetrical.

GUI and Database Management System

Q.1.c. **What is view? How it is created and stored?** (04)

Ans:

View: Any relation that is not part of the logical model, but is made visible to a user as virtual relation is called a view.

While working with database, it is not advisory to show the entire logical model. Due to some security considerations it may be hidden from users.

Consider person who needs to know customer's loan number and branch but has no need to see loan amount. This person should see relation, in relational algebra.

Π [customer_name,loan_no,branch_name](borrower x account);

Example for create view and storage:

- Syntax for create view:

CREATE VIEW [view name] as SELECT column_name from Table_name WHERE column_name=expression list;

Example: create view V_Book as

Select title, author_name
from book;

If we want to show the contents of view then it is possible with SQL as:

Select * from V_Book;

Q.1.d. **Explain following controls of Visual Basic:** (06)

Ans:

i] Listbox:

It displays list of items from which the user can select the required one. By default, only 1 column is displayed but multiple columns can also be displayed.

ii] Combo Box:

It is expandable list box. It is collection of text box and textbox properties.

There are three types:

1] Dropdown Combo(style 0)

2] simple Combo(style 1)

3] Dropdown List(style 2)

iii] Check Box:

It is used to select one or more options. A selected checkbox shows the selection with checkmarks and checkboxes are never mutually exclusive. So users can select one or more checkboxes even if those checkboxes reside in same frame or of same form.

Q.2.a. **Explain following relational algebra operators with suitable example.** (10)

Ans:

I] select:

The **select** operation selects tuples that satisfy a given predicate. We use the lowercase

GUI and Database Management System

Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ . The argument relation is in parentheses after the σ . Thus, to select those tuples of the loan relation where the branch is "Perryridge," we write $\sigma_{\text{branch-name} = \text{"Perryridge"}}(\text{loan})$. If the loan relation is there then the relation that results from the preceding query.

We can find all tuples in which the amount lent is more than \$1200 by writing $\sigma_{\text{amount} > 1200}(\text{loan})$

In general, we allow comparisons using =, \neq , <, \leq , >, \geq in the selection predicate. of more than \$1200 made by the Perryridge branch, we write

ii] Project Operation

Suppose we want to list all loan numbers and the amount of the loans, but do not care about the branch name.

The **project** operation allows us to produce this relation.

The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relation is a set, any duplicate rows are eliminated. Projection is denoted by the uppercase Greek letter pi (π). We list those attributes that we wish to appear in the result as a subscript to π . The argument relation follows in parentheses. Thus, we write the query to list all loan numbers and the amount of the loan as

$\pi_{\text{loan-number, amount}}(\text{loan})$

Figure shows the relation that results from this query.

loan-number	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

Figure Loan number and the amount of the loan.

iii] The Natural-Join Operation

It is often desirable to simplify certain queries that require a Cartesian product. Usually, a query that involves a Cartesian product includes a selection operation on the result of the Cartesian product.

The *natural join* is a binary operation that allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the "join" symbol \bowtie . The natural-join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

iv] Division operation:

It is not primitive operation but useful for expressing queries. It is binary operation and suitable for queries that include the phrase 'for all'.

Let A have two attributes x and y and B has only one attribute y; if $A \div B$ contains all x tuples in B, there is an xy tuple in A.

Example: $\pi_{\text{sno}}(A \div B)$;

GUI and Database Management System

v] The Cartesian-Product Operation

The Cartesian-product operation, denoted by a cross (\times), allows us to combine information from any two relations. We write the Cartesian product of relations r_1 and r_2 as $r_1 \times r_2$.

Q.2.b. **What is transaction? Discuss state diagram and properties of transaction.** (10)

Ans:

A **transaction** is a **unit** of program execution that accesses and possibly updates various data items. Usually, a transaction is initiated by a user program written in high-level data manipulation language or programming language (for example, SQL, COBOL, C, C++, or Java), where it is delimited by statements (or function calls) of the form **begin transaction** and **end transaction**. The transaction consists of all operations executed between the **begin transaction** and **end transaction**. To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

Transaction State:

- . **Active**, the initial state; the transaction stays in this state while it is executing
- . **Partially committed**, after the `.nal` statement has been executed
- . **Failed**, after the discovery that normal execution can no longer proceed.
- . **Aborted**, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
- . **Committed**, after successful completion

The state diagram corresponding to a transaction appears in Figure We say that a transaction has committed only if it has entered the committed state. Similarly, we say that a transaction has aborted only if it has entered the aborted state.

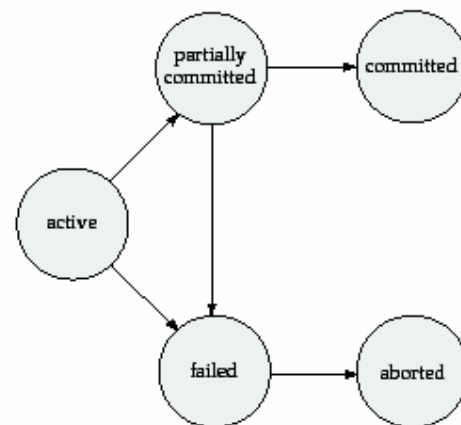


Figure State diagram of a transaction.

- . **Consistency:** The consistency requirement here is that the sum of A and B be unchanged by the execution of the transaction. Without the consistency requirement, money could be created or destroyed by the transaction! It can be verified easily that, if the database is consistent before an execution of the transaction, the database remains consistent after the execution of the transaction.

GUI and Database Management System

. **Atomicity:** Suppose that, just before the execution of transaction T_i the values of accounts A and B are \$1000 and \$2000, respectively. Now suppose that, during the execution of transaction T_i , a failure occurs that prevents T_i from completing its execution successfully. Examples of such failures include power failures, hardware failures, and software errors. Further, suppose that the failure happened after the write(A) operation but before the write(B) operation.

. **Durability:** The durability property guarantees that, once a transaction complete successfully, all the updates that it carried out on the database persist, even if there is a system failure after the transaction completes execution.

. **Isolation:** Even if the consistency and atomicity properties are ensured for each transaction if several transactions are executed concurrently, their operations may interleaved undesirable way, resulting in an inconsistent state.

Q.3.a. *Elaborate design consideration of GUI.* (10)

Ans:

- Learn about project architecture and environment.
- Perform interviews and site visits.
- Collect and document design considerations.
- Select type of interface and main window.
- Draw conceptual model.
- Draw navigation model.
- Do some reality checks.
- Complete the project GUI standard.

Q.3.b. *Discuss Option explicit statement with example.* (10)

Ans:

Option explicit statement forces explicit declaration of all variables in file.

Option Explicit {on/off}.

In this,

on: optional. Enables option Explicit checking. If on and of is not specified then the default is on.

Off: optional. Disables Option Explicit checking.

Example:

Option Explicit on

Dim thisVar As Integer

ThisVar = 10

Thisint = 10

Q.4.a. *What do you understand by deadlocks in database system. Explain how it is prevented.* (10)

Ans:

GUI and Database Management System

Deadlocks:

A system is in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set. More precisely, there exists a set of waiting transactions $\{T_0, T_1, \dots, T_n\}$ such that T_0 is waiting for a data item that T_1 holds, and T_1 is waiting for a data item that T_2 holds, and \dots , and T_{n-1} is waiting for a data item that T_n holds, and T_n is waiting for a data item that T_0 holds. None of the transactions can make progress in such a situation.

There are two principal methods for dealing with the deadlock problem.

1] **deadlock prevention** protocol to ensure that the system will *never* enter a deadlock state.

There are two approaches to deadlock prevention. One approach ensures that no cyclic waits can occur by ordering the requests for locks, or requiring all locks to be acquired together. The other approach is closer to deadlock recovery, and performs transaction rollback instead of waiting for a lock, whenever the wait could potentially result in a deadlock.

Another approach for preventing deadlocks is to impose an ordering of all data items, and to require that a transaction lock data items only in a sequence consistent with the ordering. We have seen one such scheme in the tree protocol, which uses a partial ordering of data items.

Two different deadlock prevention schemes using timestamps have been proposed:

1. The wait–die scheme is a non-preemptive technique. When transaction T_i requests a data item currently held by T_j , T_i is allowed to wait only if it has a timestamp smaller than that of T_j (that is, T_i is older than T_j). Otherwise, T_i is rolled back (dies).

2. The wound–wait scheme is a preemptive technique. It is a counterpart to the wait–die scheme. When transaction T_i requests a data item currently held by T_j , T_i is allowed to wait only if it has a timestamp larger than that of T_j (that is, T_i is younger than T_j). Otherwise, T_j is rolled back (T_j is *wounded* by T_i).

2] **deadlock detection and deadlock recovery scheme** can allow the system to enter a deadlock state, and then try to recover by using a deadlock detection and deadlock recovery schema

Recovery from Deadlock

When a detection algorithm determines that a deadlock exists, the system must recover from the deadlock. The most common solution is to roll back one or more transactions to break the deadlock. Three actions need to be taken:

1. Selection of a victim. Given a set of deadlocked transactions, we must determine which transaction (or transactions) to roll back to break the deadlock. We should roll back those transactions that will incur the minimum cost. Unfortunately, the term *minimum cost* is not a precise one.

2. Rollback. Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back. The simplest solution is a total rollback restart it.

3. Starvation. In a system where the selection of victims is based primarily on cost factors, it may happen that the same transaction is always picked as a victim.

GUI and Database Management System

Q.4.b. Explain any one Timestamp based protocol.

(10)

Ans:

The Timestamp-Ordering Protocol:

The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows:

1. Suppose that transaction T_i issues read(Q).

a. If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and T_i is rolled back.

b. If $TS(T_i) \leq W\text{-timestamp}(Q)$, then the read operation is executed, and $R\text{timestamp}(Q)$ is set to the maximum of $R\text{-timestamp}(Q)$ and $TS(T_i)$.

2. Suppose that transaction T_i issues write(Q).

a. If $TS(T_i) < R\text{-timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced. Hence, the system rejects the write operation and rolls T_i back.

b. If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of Q . Hence, the system rejects this write operation and rolls T_i back.

c. Otherwise, the system executes the write operation and sets $W\text{-timestamp}(Q)$ to $TS(T_i)$.

If a transaction T_i is rolled back by the concurrency-control scheme as result of issuance of either a read or write operation, the system assigns it a new timestamp and restarts it.

The timestamp-ordering protocol ensures conflict serializability. This is because conflicting operations are processed in timestamp order. protocol ensures freedom from deadlock, since no transaction ever waits.

Q.5.a. For the given employee database give an expression in SQL for the following:

(10)

Employee(empname, street, city)

Works(empname, company_name, salary)

Company(Company_name, city)

Manages(empname, manager_name)

Ans:

i] modify the database so that 'Ram' now lives in Mumbai.

Update Employee Set city='Mumbai' where empname = 'Ram';

ii] Give all employees of satyam a 50 percent raise.

Update Works Set salary = 1.50 * salary where Company_name = 'satyam';

iii] list all the employees who lives in same cities as their managers.

Select empname from Employee e, (select city from employee, manages where employee.empname = manages.manager_name) as Mngr(city) where e.city=mngr.city;

GUI and Database Management System

Q.5.b. Define serializability? Explain conflict and view serializability.

(10)

Ans:

Serializability

The database system must control concurrent execution of transactions, to ensure that the database state remains consistent. Before examine how the database

T ₁	T ₂
read(A)	read(A)
A := A - 50	temp := A * 0.1
	A := A - temp
	write(A)
	read(B)
write(A)	
read(B)	
B := B + 50	
write(B)	B := B + temp
	write(B)

Figure a concurrent schedule.

Conflict Serializability

Let us consider a schedule S in which there are two consecutive instructions I_i and I_j , of transactions T_i and T_j , respectively ($i \neq j$). If I_i and I_j refer to different data items, then can swap I_i and I_j without affecting the results of any instruction in the schedule. However, if I_i and I_j refer to the same data item Q , then the order of the two steps may matter. Since for dealing with only read and write instructions, there are four cases that need to consider:

1. $I_i = \text{read}(Q)$, $I_j = \text{read}(Q)$. The order of I_i and I_j does not matter, since the same value of Q is read by T_i and T_j , regardless of the order.
2. $I_i = \text{read}(Q)$, $I_j = \text{write}(Q)$. If I_i comes before I_j , then T_i does not read the value of Q that is written by T_j in instruction I_j . If I_j comes before I_i , then T_i reads the value of Q that is written by T_j . Thus, the order of I_i and I_j matters.
3. $I_i = \text{write}(Q)$, $I_j = \text{read}(Q)$. The order of I_i and I_j matters for reasons similar to those of the previous case.
4. $I_i = \text{write}(Q)$, $I_j = \text{write}(Q)$. Since both instructions are write operations, the order of these instructions does not affect either T_i or T_j . However, the value obtained by the next $\text{read}(Q)$ instruction of S is affected, since the result of only the latter of the two write instructions is preserved in the database. If there is no other $\text{write}(Q)$ instruction after I_i and I_j in S , then the order of I_i and I_j directly affects the final value of Q in the database state that results from schedule S .

Thus, only in the case where both I_i and I_j are read instructions does the relative order of their execution not matter.

GUI and Database Management System

We say that T_i and T_j conflict if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.

View Serializability

In this section, consider a form of equivalence that is less stringent than conflict equivalence, but that, like conflict equivalence, is based on only the read and write operations of transactions.

The schedules S and S_* are said to be view equivalent if three conditions are met:

1. For each data item Q , if transaction T_i reads the initial value of Q in schedule S , then transaction T_i must, in schedule S_* , also read the initial value of Q .
2. For each data item Q , if transaction T_i executes $read(Q)$ in schedule S , and if that value was produced by a $write(Q)$ operation executed by transaction T_j , then the $read(Q)$ operation of transaction T_i must, in schedule S_* , also read the value of Q that was produced by the same $write(Q)$ operation of transaction T_j .
3. For each data item Q , the transaction (if any) that performs the final $write(Q)$ operation in schedule S must perform the final $write(Q)$ operation in schedule S_* .

Q.6.a. **Explain rules for converting ER model to relational model.** (10)

Ans:

There are some basic guidelines for converting ER model to relational model

- 1] Each entity set is describe as relation in which it contains separate column for each of its attribute.
- 2] Each relationship set is represented as set of relations which corresponds to all entities which are the part of that relationship set.
- 3] Each weak entity set represented by relation where it contains columns as all its own attributes as well as attributes from another entity that act as primary key for that weak entity set.
- 4] for each generalization there is relation from converting ISA hierarchy into relation.

Q.6.b. **Draw ER diagram for banking enterprise** (10)

Ans:

1] Account:

Account_no	Balance
------------	---------

2] Branch:

Branch_name	Branch_city	Assets
-------------	-------------	--------

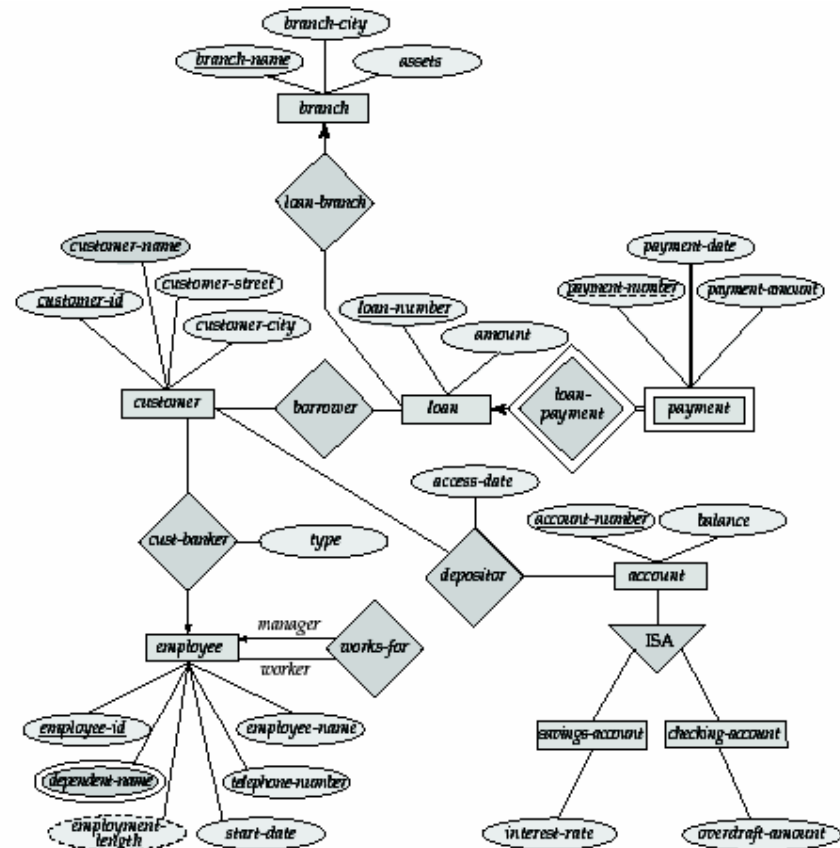
3] Customer:

Cust_name	Cust_city	Cust_street
-----------	-----------	-------------

GUI and Database Management System

4] loan:

Loan_no	Amount
---------	--------



Q.7. Write note on:

Ans:

I] Weak entity set:

An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed a weak entity set.

(07)

Specialization:

An entity set may include subgroupings of entities that are distinct in some way from other entities in the set. For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinctive entity groupings. Consider an entity set person, with attributes name, street, and city. A person may be further classified as one of the following:

GUI and Database Management System

. customer

. employee

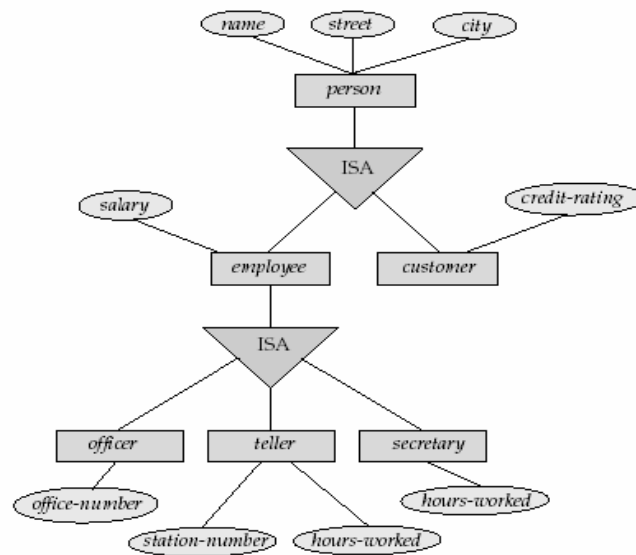
Each of these person types is described by a set of attributes that includes all attribute entity set person plus possibly additional attributes. For example, customer entities may be described further by the attribute customer-id, whereas employee entities may be described further by the attributes employee-id and salary. The process of designating subgroupings within an entity set is called specialization.

In terms of an E-R diagram, specialization is depicted by a triangle component labelled ISA. The label ISA stands for "is a".

Generalization:

The refinement from an initial entity set into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicit. The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher-level entity set on the basis of common features.

This commonality can be expressed by generalization, which is a containment relationship that exists between a *higher-level* entity set and one or more *lower-level* entity sets. The *person* entity set is the super class of the *customer* and *employee* subclasses. For all practical purposes, generalization is a simple inversion of specialization.



ii] Total participation:

The participation of an entity set E in a relationship set R is said to be **total** if every entity in E participates in at least one relationship in R . If only some entities in E participate in relationships in R .

(06)

partial participation:

The participation of entity set E in relationship R is said to be **partial participation**.

Unique key:

It is also called 'alternate key'. It is similar to primary key except it accepts null values, so that records can still be entered submitting null values to this attribute.

GUI and Database Management System

primary key:

The primary key is the key chosen by the database designer as the main identifier for all records in that relation

iii] Logbased recovery mechanism

(07)

Update log record describes single database write. The fields are as follow:

Transaction identifier: It is unique identifier of transaction that performs the write operation. Data item identifier: It is unique identifier of data item written. It is location on disk of data item.

Old value: The value of data item prior to the write.

New value: The value that data item will have after write.

We denote various types of log records as:

<Ti start> : Ti started

<Ti commit> : Ti committed.

<Ti abort> : Ti aborted

There are two techniques which ensures transaction atomicity :

Deferred database modification:

Transaction atomicity is ensure by creating log for each database modification but all the write operations are delayed till transaction partially commits.

Partial Commit: A transaction is said to be partially committed when final action of the transaction has been executed.

immediate database modification:

It allows database modification to be output to database while transaction is still in active state. Data modification written by active transaction are called 'uncommitted modifications'. In immediate database modification, the old value field of the log record is requires because the modifications are made before transactions commits s if system crash or transaction failure take place the changes which are made need to undone.

Checkpoints based recovery mechanism:

If system crashes or failure occurs, there is need to consult the log record to decide which transaction needs to be redone and which need to undone. In this three are two overheads:

searching the entire log records is time consuming.

Most of transactions need to redone have already written their updates into the database so make time increment for recovery.

Hence to reduce time taken by recovery and to increase efficiency we use checkpoints.

Along with maintaining log during execution of transaction the system periodic perform checkpoints, which includes the following set of actions:

Output all log records current residing in main memory onto the storage.

output all modify buffer blocks to disk.