Question 1: What is Cryptography and what are the Encryption Methods?

Answer: Cryptography is associated with the process of converting ordinary plain text into unintelligible text and vice-versa. It is a method of storing and transmitting data in a particular form so that only those for whom it is intended can read and process it. Cryptography not only protects data from theft or alteration, but can also be used for user authentication.

Modern cryptography concerns with:

Confidentiality - Information cannot be understood by anyone.

Integrity - Information cannot be altered.

Non-repudiation - Sender cannot deny his/her intentions in the transmission of the information at a later stage

Authentication - Sender and receiver can confirm each

Cryptography is used in many applications like banking transactions cards, computer passwords, and e- commerce transactions.

Encryption Methods:

Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption; also called *symmetric encryption*. Primarily used for privacy and confidentiality.

Public Key Cryptography (PKC): Uses one key for encryption and another for decryption; also called *asymmetric encryption*. Primarily used for authentication, non-repudiation, and key exchange.

Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information, providing a digital fingerprint. Primarily used for message integrity.

Question 2: On entering a URL in a browser, explain the detailed procedure in which the request is handled by the browser and the result is obtained for the given search query.

Answer:

The browser checks the cache for a DNS record to find the corresponding IP address of User input address for example -- maps.google.com.

DNS(Domain Name System) is a database that maintains the name of the website (URL) and the particular IP address it links to. Every single URL on the internet has a unique IP address assigned to it. The IP address belongs to the computer which hosts the server of the website we are requesting to access. For example, www.google.com has an IP address of 209.85.227.104. So if you'd like, you can reach www.google.com by typing http://209.85.227.104 on your browser. DNS is a list of URLs, and their IP addresses, like how a phone book is a list of names and their corresponding phone numbers.

To find the DNS record, the browser checks four caches.

● First, it checks the browser cache. The browser maintains a repository of DNS records for a fixed duration for websites you have previously visited. So, it is the first place to run a DNS query.

● Second, the browser checks the OS cache. If it is not in the browser cache, the browser will make a system call (i.e., *gethostname* on Windows) to your underlying computer OS to fetch the record since the OS also maintains a cache of DNS records.

● Third, it checks the router cache. If it's not on your computer, the browser will communicate with the router that maintains its' own cache of DNS records.

● Fourth, it checks the ISP cache. If all steps fail, the browser will move on to the ISP. Your ISP maintains its' own DNS server, which includes a cache of DNS records, which the browser would check with the last hope of finding your requested URL.

If the requested URL is not in the cache, ISP's DNS server initiates a DNS query to find the IP address of the server that hosts maps.google.com.

As mentioned earlier, for my computer to connect with the server that hosts maps.google.com, I need the IP address of maps.google.com. The purpose of a DNS query is to search multiple DNS servers on the internet until it finds the correct IP address for the website. This type of search is called a recursive search since the search will repeatedly continue from a DNS server to a DNS server until it either finds the IP address we need or returns an error response saying it was unable to find it.

In this situation, we would call the ISP's DNS server a DNS recursor whose responsibility is to find the proper IP address of the intended domain name by asking other DNS servers on the internet for an answer. The other DNS servers are called name servers since they perform a DNS search based on the domain architecture of the website domain name.

Many website URLs we encounter today contain a third-level domain, a second-level domain, and a top-level domain. Each of these levels contains their own name server, which is queried during the DNS lookup process.

For maps.google.com, first, the DNS recursor will contact the root name server. The root name server will redirect it to the **.com** domain name server. **.com** name server will redirect it to the **google.com** name server. The **google.com** name server will find the matching IP address for maps.google.com in its' DNS records and return it to your DNS recursor, which will send it back to your browser.

These requests are sent using small data packets that contain information such as the content of the request and the IP address it is destined for (IP address of the DNS recursor). These packets travel through multiple networking equipment between the client and the server before it reaches the correct DNS server. This equipment use routing tables to figure out which way is the fastest possible way for the packet to reach its' destination. If these packets get lost, you'll get a request failed error. Otherwise, they will reach the correct DNS server, grab the correct IP address, and come back to your browser.

3. The browser initiates a TCP connection with the server.

Once the browser receives the correct IP address, it will build a connection with the server that matches the IP address to transfer information. Browsers use internet protocols to build such connections. There are several different internet protocols that can be used, but TCP is the most common protocol used for many types of HTTP requests.

To transfer data packets between your computer(client) and the server, it is important to have a TCP connection established. This connection is established using a process called the TCP/IP

three-way handshake. This is a three-step process where the client and the server exchange SYN(synchronize) and ACK(acknowledge) messages to establish a connection.

a. The client machine sends a SYN packet to the server over the internet, asking if it is open for new connections.

b. If the server has open ports that can accept and initiate new connections, it'll respond with an ACKnowledgment of the SYN packet using a SYN/ACK packet.

c. The client will receive the SYN/ACK packet from the server and will acknowledge it by sending an ACK packet.

Then a TCP connection is established for data transmission!

4. The browser sends an HTTP request to the webserver.

Once the TCP connection is established, it is time to start transferring data! The browser will send a GET request asking for maps.google.com web page. If you're entering credentials or submitting a form, this could be a POST request. This request will also contain additional information such as browser identification (*User-Agent* header), types of requests that it will accept (*Accept* header), and connection headers asking it to keep the TCP connection alive for additional requests. It will also pass information taken from cookies the browser has in store for this domain.

5. The server handles the request and sends back a response.

The server contains a webserver (i.e., Apache, IIS) that receives the request from the browser and passes it to a request handler to read and generate a response. The request handler is a program (written in ASP.NET, PHP, Ruby, etc.) that reads the request, its' headers, and cookies to check what is being requested and also update the information on the server if needed. Then it will assemble a response in a particular format (JSON, XML, HTML).

6. The server sends out an HTTP response.

The server response contains the web page you requested as well as the status code, compression type (*Content-Encoding)*, how to cache the page (*Cache-Control*), any cookies to set, privacy information, etc.

7. The browser displays the HTML content (for HTML responses, which is the most common).

The browser displays the HTML content in phases. First, it will render the bare bone HTML skeleton. Then it will check the HTML tags and send out GET requests for additional elements on the web page, such as images, CSS stylesheets, JavaScript files, etc. These static files are cached by the browser, so it doesn't have to fetch them again the next time you visit the page. In the end, you'll see maps.google.com appearing on your browser.

That's it!

Question 3: How will you create persistent connections between the server and the client?

Answer: A persistent connection (HTTP persistent connection) is a network communication channel that remains open for further HTTP requests and responses rather than closing after a single exchange.

HTTP has a persistent connection function that allows the channel to remain open rather than be closed after a requested exchange of data. TCP is a connection-oriented protocol: It starts a connection after confirmation from both ends that they are available and open to a data exchange. In a non-persistent connection, the channel closes when one host signals that it wants to end communications or when a certain amount of time has elapsed with no exchange of data. To maintain a persistent connection, TCP keep-alive packets are sent to prevent the connection from timing out.

An open connection is faster for frequent data exchanges. Communications overhead is saved by leaving a connection open rather than opening and closing sessions for each request. Persistent connections can also be used with APIs to enable servers to push data to clients. Other benefits of persistent connections include reduced network congestion, latency and CPU and memory usage due to the lower number of connections; errors can also be reported without closing the connection.

Persistent connections added for HTTP 1.0 used an extra header to request the client keep the connection alive; HTTP 1.1 assumes all connections to be persistent unless otherwise specified. HTTP 2.0 expands the persistent connection to enable numerous requests and returns of data to be exchanged simultaneously over a single connection.

Persistent connections are also called HTTP keep-alive or HTTP connection reuse.

Keep-Alive Operation

Keep-alive is deprecated and no longer documented in the current HTTP/1.1 specification. However, keep-alive handshaking is still in relatively common use by browsers and servers, so HTTP implementors should be prepared to interoperate with it. We'll take a quick look at keep-alive operation now. Refer to older versions of the HTTP/1.1 specification (such as RFC 2068) for a more complete explanation of keep-alive handshaking.

Clients implementing HTTP/1.0 keep-alive connections can request that a connection be kept open by including the Connection: Keep-Alive request header.

If the server is willing to keep the connection open for the next request, it will respond with the same header in the response (see Figure 4-14). If there is no Connection: keep-alive header in the response, the client assumes that the server does not support keep-alive and that the server will close the connection when the response message is sent back.

Question 4 : What is Multithreading? What is the difference between a thread and a process?

Answer : Multithreading:

A thread is a path which is followed during a program's execution. Majority of programs written now a days run as a single thread. Lets say, for example a program is not capable of reading keystrokes while making drawings. These tasks cannot be executed by the program at the same time. This problem can be solved through multitasking so that two or more tasks can be executed simultaneously.

Multitasking is of two types: Processor based and thread based. Processor based multitasking is totally managed by the OS, however multitasking through multithreading can be controlled by the programmer to some extent.

The concept of multi-threading needs proper understanding of these two terms – a process and a thread. A process is a program being executed. A process can be further divided into independent units known as threads.

A thread is like a small light-weight process within a process. Or we can say a collection of threads is what is known as a process.

Applications –
Threading is used widely in almost every field. Most widely it is seen over the internet now days where we are using transaction processing of every type like recharges, online transfer, banking etc. Threading is a segment which divide the code into small parts that are of very light weight and has less burden on CPU memory so that it can be easily worked out and can achieve goal in desired field. The concept of threading is designed due to the problem of fast and regular changes in technology and less the work in different areas due to less application. Then as says “need is the generation of creation or innovation” hence by following this approach human mind develop the concept of thread to enhance the capability of programming.

The Difference between Process and thread

The major differences between a process and a thread are given as follows −

Comparison Basis	Process	Thread
Definition	A process is a program under execution i.e an active program.	A thread is a lightweight process that can be managed independently by a scheduler.
Context switching time	Processes require more time for context switching as they are more heavy.	Threads require less time for context switching as they are lighter than processes.
Memory Sharing	Processes are totally independent and don't share memory.	A thread may share some memory with its peer threads.
Communication	Communication between processes requires more time than between threads.	Communication between threads requires less time than between processes .
Blocked	If a process gets blocked, remaining processes can continue execution.	If a user level thread gets blocked, all of its peer threads also get blocked.
Resource Consumption	Processes require more resources than threads.	Threads generally need less resources than processes.
Dependency	Individual processes are independent of each other.	Threads are parts of a process and so are dependent.
Data and Code sharing	Processes have independent data and code segments.	A thread shares the data segment, code segment, files etc. with its peer threads.
Treatment by OS	All the different processes are treated separately by the operating system.	All user level peer threads are treated as a single task by the operating system.
Time for creation	Processes require more time for creation.	Threads require less time for creation.
Time for termination	Processes require more time for termination.	Threads require less time for termination.

Question 5 : Explain Indexing in DBMS.

Answer : Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types –

Primary Index – Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.

Secondary Index – Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.

Clustering Index – Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types –

Dense Index

Sparse Index

Ordered Indexing is of two types –

Dense Index

Sparse Index

Dense Index

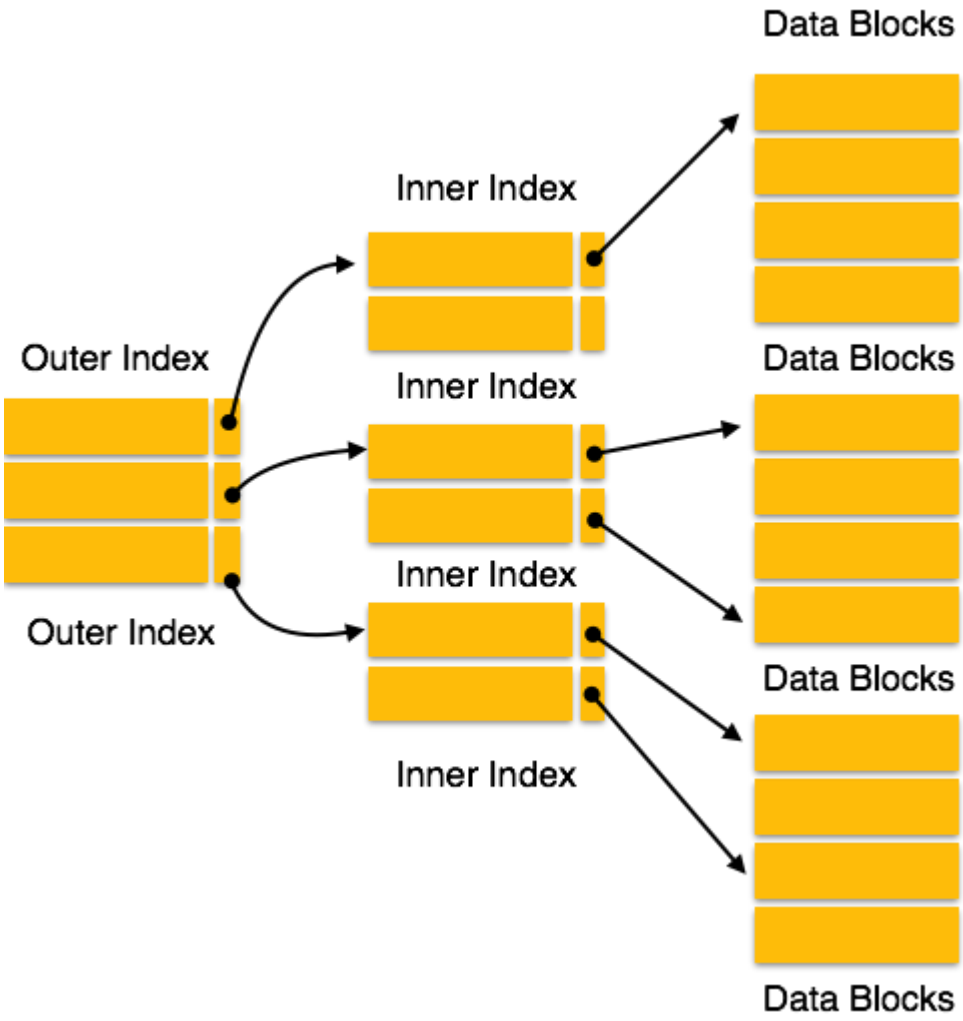
In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.

Multilevel Index

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

Question 6: What is normalization and denormalization and why do we need it?

Answer :

Normalization:

Normalization is the method used in a database to reduce the data redundancy and data inconsistency from the table. It is the technique in which Non-redundancy and consistency data are stored in the set schema. By using normalization the number of tables is increased instead of decreased.

Denormalization:

Denormalization is also the method which is used in a database. It is used to add the redundancy to execute the query quickly. It is a technique in which data are combined to execute the query quickly. By using denormalization the number of tables is decreased which oppose to the normalization.

BASIS FOR COMPARISON	NORMALIZATION	DENORMALIZATION
Basic	Normalization is the process of creating a set schema to store non-redundant and consistent data.	Denormalization is the process of combining the data so that it can be queried speedily.
Purpose	To reduce the data redundancy and inconsistency.	To achieve the faster execution of the queries through introducing redundancy.
Used in	OLTP system, where the emphasize is on making the insert, delete and update anomalies faster and storing the quality data.	OLAP system, where the emphasis is on making the search and analysis faster.
Data integrity	Maintained	May not retain
Redundancy	Eliminated	Added
Number of tables	Increases	Decreases
Disk space	Optimized usage	Wastage

Question 7 : Difference between INNER and OUTER JOIN.

Answer :

Inner Join

An inner join focuses on the commonality between two tables. When using an inner join, there must be at least some matching data between two (or more) tables that are being compared. An inner join searches tables for matching or overlapping data. Upon finding it, the inner join combines and returns the information into one new table.

Example of Inner Join

Let's consider a common scenario of two tables: product prices and quantities. The common information in the two tables is product name, so that is the logical column to join the tables ***on***. There are some products that are common in the two tables; others are unique to one of the tables and don't have a match in the other table.

An inner join on *Products* returns information about only those products that are common in both tables.

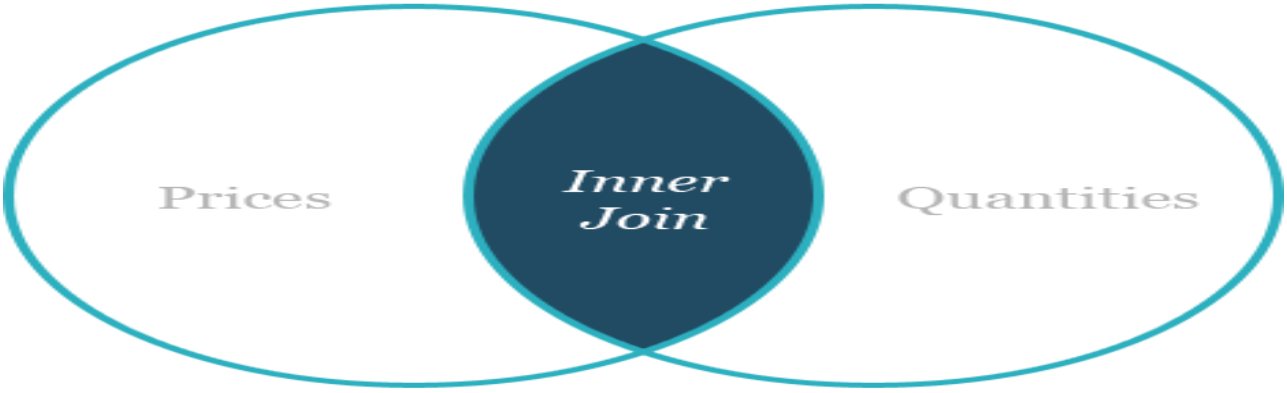


TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices INNER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR INNER JOIN

PRODUCT	PRICE	QUANTITY
Potatoes	$3	45
Avocados	$4	63
Kiwis	$2	19
Onions	$1	20
Melons	$5	66

Outer Join

An outer join returns a set of records (or rows) that include what an inner join would return but also includes other rows for which no corresponding match is found in the other table.

There are three types of outer joins:

Left Outer Join (or Left Join)

Right Outer Join (or Right Join)

Full Outer Join (or Full Join)

Each of these outer joins refers to the part of the data that is being compared, combined, and returned. Sometimes nulls will be produced in this process as some data is shared while other data is not.

Left Outer Join

A left outer join will return all the data in Table 1 and all the shared data (so, the inner part of the Venn diagram example), but only corresponding data from Table 2, which is the right join.

Left Join Example

In our example database, there are two products — oranges and tomatoes — on the 'left' (*Prices* table) that do not have a corresponding entry on the 'right' (Quantities table). In a left join, these rows are included in the result set with a NULL in the Quantity column. The other rows in the result are the same as the inner join.

Right Outer Join

A right outer join returns Table 2's data and all the shared data, but only corresponding data from Table 1, which is the left join.

Right Join Example

Similar to the left join example, the output of a right outer join includes all rows of the inner join and two rows — broccoli and squash — from the 'right' (*Quantities* table) that do not have matching entries on the left.

Full Outer Join

A full outer join, or full join, which is *not* supported by the popular MySQL database management system, combines and returns *all* data from two or more tables, regardless of whether there is shared information. Think of a full join as simply duplicating all the specified information, but in one table, rather than multiple tables. Where matching data is missing, nulls will be produced.

Question 8: What is Cache? Where does cache lie in an Operating System? Difference between Cache and HashMap.

Answer: In computing, a **cache** is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere. A *cache hit* occurs when the requested data can be found in a cache, while a *cache miss* occurs when it cannot. Cache

hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests that can be served from the cache, the faster the system performs.

Cache memory lies on the path between the CPU and the main memory. It facilitates the transfer of data between the processor and the main memory at the speed which matches to the speed of the processor.

Purpose of Cache:

The fast speed of the cache memory makes it extremely useful.

It is used for bridging the speed mismatch between the fastest CPU and the main memory.

It does not let the CPU performance suffer due to the slower speed of the main memory.

A hash maps a set of objects to a set of integers. If the hash is any good, that set of integers will be uniformly distributed over their range.

A cache stores copies of expensive-to-access resources in much faster local storage. Some caches uses hashing functions as part of the lookup process, but it's not required.

Question 9 : How will you analyze 'process out of memory exception' in your node js application?

Answer: This error occurs when the memory allocated for the execution of your application is less than the required memory by your application. You will get an error like below when you run your application.

To solve this issue you need to run your application by increasing the memory limit by using the command **`--max-old-space-size`**. By default the memory limit of Node.js is 512 mb. Lets say I run the above code using the command **`node --max-old-space-size=512 example5.js`**, it will give me the same error - FATAL ERROR: CALL_AND_RETRY_LAST Allocation failed - process out of memory as shown below.

Now when I increase the memory and let say run using the command **`node --max-old-space-size=1024 example5.js`** it will again give the same exception.

Again we will increase the memory and try to run at 2000 MB using the command node --max-old-space-size=2000 example5.js, this time we will not get any exception and the program will run.

Question 10: If RAM size is 4GB, if 4 processes of size 2GB are launched! What happens?

Answer :

Nothing weird happen. All processes will run simultaneously because RAM based on BUS structure so when a process is initiated it required some memory which it kept using until it requires so If we run 4 process of 2 GB each in 4GB RAM it run parallel as no process requires full 4GB at a same time. NO process is as bigger as 4GB.

This is HIGHLY platform dependent. On many 32bit OS's, no single process can ever use more than 2GB of memory, regardless of the physical memory installed or virtual memory allocated.

For example, my work computers use 32bit Linux with PAE (Physical Address Extensions) to allow it to have 16GB of RAM installed. The 2GB per process limit still applies however. Having the extra RAM simply allows me to have more individual processes running. 32bit Windows is the same way.

64bit OS's are more of a mixed bag. 64bit Linux will allow individual processes to map memory well in excess of 32GB (but again, varies from Kernel to Kernel). You will be limited only by the amount of Swap (Linux virtual memory) you have. 64bit Windows is a complete crap shoot. Certain versions will only allow 2GB per process, but most will allow >32GB limited only by the amount of Page File the user has allocated.