Homework

1- **Imperative programming** is a type of programming paradigm that describes how the program executes. Developers are more concerned with how to get an answer step by step. It comprises the sequence of command imperatives. In this, the order of execution is very important and uses both mutable and immutable data. Fortran, Java, C, C++ programming languages are examples of imperative programming.

Declarative programming is a type of programming paradigm that describes what programs to be executed. Developers are more concerned with the answer that is received. It declares what kind of results we want and leave programming language aside focusing on simply figuring out how to produce them. In simple words, it mainly focuses on end result. It expresses the logic of computation. Miranda, Erlang, Haskell, Prolog are a few popular examples of declarative programming.

Imperative Programming	Declarative Programming
In this, programs specify how it is to be done.	In this, programs specify what is to be done.
It simply describes the control flow of computation.	It simply expresses the logic of computation.
Its main goal is to describe how to get it or accomplish it.	Its main goal is to describe the desired result without direct dictation on how to get it.
Its advantages include ease to learn and read, the notional model is simple to understand, etc.	Its advantages include effective code, which can be applied by using ways, easy extension, high level of abstraction, etc.
Its type includes procedural programming, object-oriented programming, parallel processing approach.	Its type includes logic programming and functional programming.
In this, the user is allowed to make decisions and commands to the compiler.	In this, a compiler is allowed to make decisions.
It has many side effects and includes mutable variables as compared to declarative programming.	It has no side effects and does not include any mutable variables as compared to imperative programming.
It gives full control to developers that are very important in low-level programming.	It may automate repetitive flow along with simplifying code struc

Via : https://www.geeksforgeeks.org/difference-between-imperative-and-declarative-programming/

2- An **index** is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements.

Indexes are objects that take up extra space. For this reason, creating an index for key-values that we will not filter will take up unnecessary space on the disk. The disk space occupied for the places to be filtered is insignificant compared to the gain in performance.

3- **Normalization** is the process of organizing the data in the database. It is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Example of 1NF , Via : https://www.javatpoint.com/dbms-first-normal-form

1NF :

- It must contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Example of 2NF , Via : https://www.javatpoint.com/dbms-second-normal-form

2NF :

- In the 2NF, relation must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

> {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Example of 3NF, Via : https://www.javatpoint.com/dbms-third-normal-form

3NF :

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

4- **Object-Relational Mapping(ORM)** is a technique that lets you query and manipulate data from a database using an object-oriented paradigm.

Pros and Cons

Using ORM saves a lot of time because:

- DRY: You write your data model in only one place, and it's easier to update, maintain, and reuse the code.
- A lot of stuff is done automatically, from database handling to I18N.
- It forces you to write MVC code, which, in the end, makes your code a little cleaner.
- You don't have to write poorly-formed SQL (most Web programmers really suck at it, because SQL is treated like a "sub" language, when in reality it's a very powerful and complex one).
- Sanitizing; using prepared statements or transactions are as easy as calling a method.

Using an ORM library is more flexible because:

- It fits in your natural way of coding (it's your language!).
- It abstracts the DB system, so you can change it whenever you want.
- The model is weakly bound to the rest of the application, so you can change it or use it anywhere else.
- It lets you use OOP goodness like data inheritance without a headache.

But ORM can be a pain:

- You have to learn it, and ORM libraries are not lightweight tools;
- You have to set it up. Same problem.
- Performance is OK for usual queries, but a SQL master will always do better with his own SQL for big projects.
- It abstracts the DB. While it's OK if you know what's happening behind the scene, it's a trap for new programmers that can write very greedy statements, like a heavy hit in a `for` loop.

Via : https://stackoverflow.com/questions/1279613/what-is-an-orm-how-does-it-work-and-how-should-i-use-one

5- A **Domain Specific Language** is a tool for programming solutions to a specific, narrow set of problems. It's more specialized than a general-purpose programming language, which can be used to program solutions for many kinds of problems.

The process for creating a DSL is:

- Have a problem
- Decompose the problem into smaller parts to be solved, and solve it
- Have a bunch more similar problems
- Decompose those, too, and solve them
- Realize that many of the problems have similar decompositions - there are things you're doing over and over again
- Decide to think of the common parts of the problem decompositions as "primitives" of a class of problems
- Write a tool to help you manipulate those primitives

6- A **long-lived transaction** is a transaction that spans multiple database transactions. The transaction is considered "long-lived" because its boundaries must, by necessity of business logic, extend past a single database transaction. A long-lived transaction can be thought of as a sequence of database transactions grouped to achieve a single atomic result.

A common example is a multi-step sequence of requests and responses of an interaction with a user through a web client.

A long-lived transaction creates challenges of concurrency control and scalability.

A chief strategy in designing long-lived transactions is optimistic concurrency control with versioning.

7- A **thread pool** is a software design pattern for achieving concurrency of execution in a computer program. Often also called a replicated workers or worker-crew model, a thread pool maintains multiple threads waiting for tasks to be allocated for concurrent execution by the supervising program. By maintaining a pool of threads, the model increases performance and avoids latency in execution due to frequent creation and destruction of threads for short-lived tasks.The number of available threads is tuned to the computing resources available to the program, such as a parallel task queue after completion of execution.

8- **Scalability** is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.

Vertical scaling keeps your existing infrastructure but adds computing power. Your existing pool of code does not need to change — you simply need to run the same code on machines with better specs. By scaling up, you increase the capacity of a single machine and increase its throughput. Vertical scaling allows data to live on a single node, and scaling spreads the load through CPU and RAM resources for your machines.

Horizontal scaling simply adds more instances of machines without first implementing improvements to existing specifications. By scaling out, you share the processing power and load balancing across multiple machines.

9- **Data Replication** : The primary server node copies data onto secondary server nodes. This can help increase data availability and act as a backup, in case if the primary server fails.

Sharding : Handles horizontal scaling across servers using a shard key. This means that rather than copying data holistically, sharding copies pieces of the data (or “shards”) across multiple replica sets. These replica sets work together to utilize all of the data.

Think of it like a pizza. With replication, you are making a copy of a complete pizza pie on every server. With sharding, you’re sending pizza slices to several different replica sets. Combined together, you have access to the entire pizza pie.