EXPLAIN THE FOLLOWING:

1.ENTITIES:

An entity in a database is a container designed to store and delineate information important to the goals of a project. Learn how entities differ from attributes and why relationships between entities are the key to a good database or can also be defined as

- **.**An entity is an object that exists. It doesn't have to do anything; it just has to exist. In database administration, an **entity** can be a single thing, person, place, or object. Data can be stored about such entities. A design tool that allows database administrators to view the relationships between several entities is called the **entity relationship diagram (ERD)**.

In database administration, only those things about which data will be captured or stored is considered an entity. If you aren't going to capture data about something, there's no point in creating an entity in a database.

If you're creating a database of your employees, examples of entities you may have include employees and health plan enrollment.

In Entities we have something called attributes

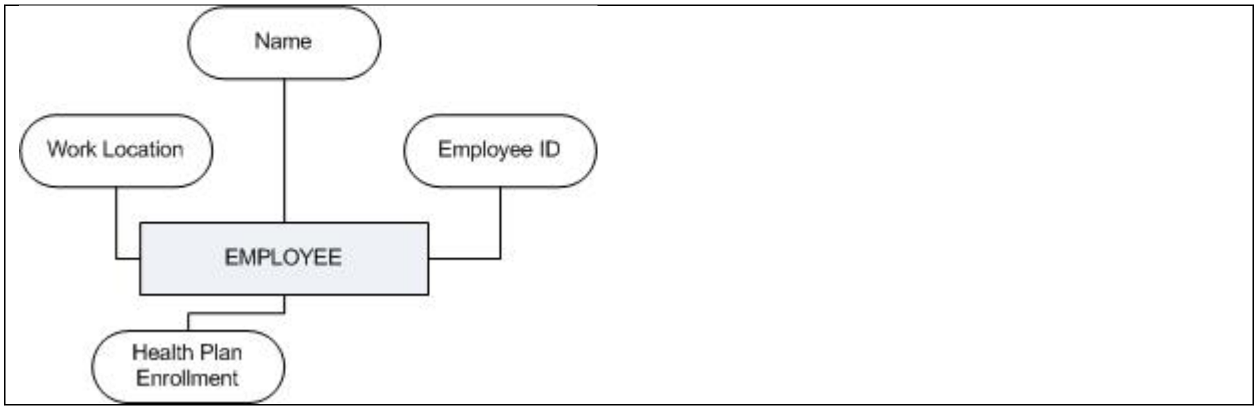
WHAT ARE ATTRIBUTES?

An **attribute** defines the information about the entity that needs to be stored. If the entity is an employee, attributes could include name, employee ID, health plan enrollment, and work location. An entity will have zero or more attributes, and each of those attributes apply only to that entity. For example, the employee ID of 123456 belongs to that employee entity alone

Attributes also have further refinements, such as domain and key. The **domain** of an entity describes the possible values of attributes. In the entity, each attribute will have only one value, which could be blank or it could be a number, text, a date, or a time. Here are examples of entity types and domains:

- Name: Jane Doe
- Employee ID: 123456
- Health Plan Enrollment: Premium Plan
- Work Location: RO, ME, Floor 2

Here's an example figure of an entity.



The **key** is the unique identifier that identifies the entity. A key is also a domain because it will have values. These values are unique to each record, and so it's a special type of domain. A key isn't always required, but it should be! In our example, a unique key value ensures that the Employee entity cannot have duplicate Social Security Numbers or Employee IDs.

2.Tuple

A tuple is simply a row contained in a table in the tablespace. A table usually contains columns and rows in which rows stand for records while columns stand for attributes. A single row of a table that has a single record for such a relation is known as a tuple. A Tuple is, therefore, a single entry in a table; it is also called a row or record. They usually represent a set of related data, in Math it is simply an ordered list of elements.

It should be noted that the relational model arranges data into at least one table of rows and columns using a unique key. The columns are referred to as attributes while the rows are called tuples or records. Furthermore, each of the rows as contained in a table has its unique key. Also, the rows in a different table can be connected by including a column for the key of the linked row

3. Normalization

Normalization is **the process of organizing data in a database**. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.

Or it can be defined as a schema design technique, by which an existing schema is modified to minimize redundancy and dependency of data.

Some facts about database normalization:

- The words normalization and normal form refer to the structure of a database.
- Normalization was developed by IBM researcher E.F. Codd In the 1970s.
- Normalization increases clarity in organizing data in Databases.

Normalization of a Database is achieved by following a set of rules called **'forms'** in creating the database.

Database Normalization rules :

The database normalization process is divided into following the normal form:

- FIRST NORMAL FORM (1NF)
- SECOND NORMAL FORM (2NF)
- THIRD NORMAL FORM (3NF)
- BOYCE-CODD NORMAL FORM (BCNF)
- FOURTH NORMAL FORM (4NF)
- FIFTH NORMAL FORM (5NF)

FIRST NORMAL FORM:

Each column is unique in 1NF.

Example:

Sample Employee table, it displays employees are working with multiple departments.

Employee	Age	Department
Melvin	32	Marketing, Sales
Edward	45	Quality Assurance
Alex	36	Human Resource

SECOND NORMAL FORM:

The entity should be considered already in 1NF, and all attributes within the entity should depend solely on the unique identifier of the entity.

Example:

Sample Products table:

productID	product	Brand
1	Monitor	Apple
2	Monitor	Samsung
3	Scanner	HP
4	Head phone	JB

THIRD NORMAL FORM:

The entity should be considered already in 2NF, and no column entry should be dependent on any other entry (value) other than the key for the table.

If such an entity exists, move it outside into a new table.

3NF is achieved, considered as the database is normalized

BOYCE-CODD NORMAL FORM :

3NF and all tables in the database should be only one primary key

FOURTH NORMAL FORM:

Tables cannot have multi-valued dependencies on a Primary Key.

FIFTH NORMAL FORM:

A composite key shouldn't have any cyclic dependencies.

Well, this is a highly simplified explanation for Database Normalization. One can study this process extensively, though. After working with databases for some time, you'll automatically create Normalized databases, as it's logical and practical.

USES OF NORMALIZATION:

1. It is used to remove the duplicate data and database anomalies from the relational table.
2. Normalization helps to reduce redundancy and complexity by examining new data types used in the table.
3. It is helpful to divide the large database table into smaller tables and link them using relationship.
4. It avoids duplicate data or no repeating groups into a table.
5. It reduces the chances for anomalies to occur in a database.

4. ER DIAGRAMS

ER Diagram stands for Entity Relationship Diagram, also known as ERD is **a diagram that displays the relationship of entity sets stored in a database**. In other words, ER diagrams help to explain the logical structure of databases.

Or it can also be defined as:

ERD stands for entity relationship diagram. People also call these types of diagrams ER diagrams and Entity Relationship Models. An ERD visualizes the relationships between entities like people, things, or concepts in a database. An ERD will also often visualize the attributes of these entities.

WHY MAKE AN ERD?

An ER diagram can help businesses document existing databases and thereby troubleshoot logic or deployment problems or spot inefficiencies and help improve processes when a business wants to undertake business process re-engineering. ERDs can also be used to design and model new databases and make sure that engineers can identify any logic or design flaws before they're implemented in production.

- Document an existing database structure
- Debug, troubleshoot, and analyze
- Design a new database
- Gather design requirements
- Business process re-engineering (BPR)

When documenting a system or process, looking at the system in multiple ways increases the understanding of that system. ERD diagrams are commonly used in conjunction with a data flow diagram to display the contents of a data store. They help us to visualize how data is connected in a general way, and are particularly useful for constructing a relational database.

The History of Entity Relationship Diagrams
Peter Chen developed ERDs in the 1970s and published his proposal for entity relationship modeling in a 1976 paper titled "The Entity-Relationship Model: Toward a Unified View of Data". Peter Chen was a computer scientist who worked on improving database design. His entity relationship model was a way to visualize a database that unified other existing models into a single understanding that removed ambiguities. Prior to ERDs, there were three data models for databases: the network model, the relational model, and the entity set model. Each had their own strengths and weaknesses, but none provided a complete view of the database. With an ERD, Chen could provide a unified framework for database modeling.

Peter Chen's work was greatly influenced by scientists and engineers who came before him, specifically Charles Bachman, who worked on visualizing databases in the 1960s and his data structure diagrams became known as Bachman diagrams.

Chen's entity relationship model is in many ways the foundation for later practices like Unified Modeling Language or UML in information systems.

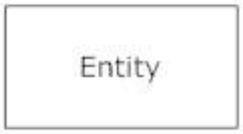
In the 1980s, another computer scientist named James Martin, worked to further refine Chen's ER model and introduced what's known today as the IE notation. IE notation uses Crow's foot to express cardinality (one to many relationship) instead of Chen's notation to epxress the same.

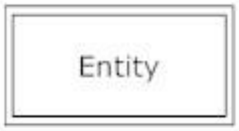
Types of ERD
ER diagrams will differ on how they express cardinality. They will also differ in how they display entities and their attributes and whether or not they show relationships or attributes as separate symbols.

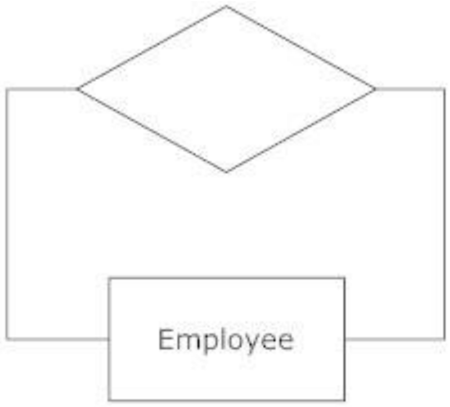
Traditional ERD
The traditional Chen ERD is like a flowchart with connected symbols for entities, relationships, and attributes. Since its first introduction by Chen in the 70s, others have proposed different cardinality notations, but the basic symbols used and how they're connected tend to look the same

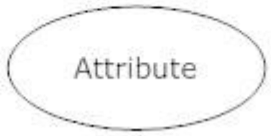
IDEF1X Notation ERD - Relational Schema
IDEF1X stands for integrated definition for data modeling. This type of ER diagram will show entities connected to each other without relationship symbols. The attributes for any entity will be listed as part of a table inside each entity shape instead of separate symbols. Some also call this type of ER diagram a Relational Schema diagram.

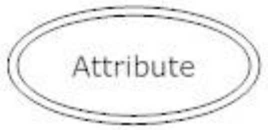
Common ERD Symbols
An ER diagram has three main components: entities, relationships, and attributes connected by lines.

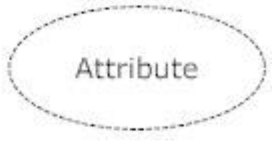
- **Entities**, which are represented by rectangles. An entity is an object or concept about which you want to store information. 

A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone. 

- **Relationships**, which are represented by diamond shapes, show how two entities share information in the database. In some cases, entities can be self-linked.


- **Attributes**, which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.


A multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values. 

A derived attribute is based on another attribute. For example, an employee's 

monthly salary is based on the employee's annual salary.

- **Connecting lines**, solid lines that connect attributes and show the relationships of entities in the diagram.

5. CARDINALITY

Cardinality is a mathematical term that refers to the number of elements in a given set. Database administrators may use cardinality to count tables and values. In a database, cardinality usually represents the relationship between the data in two different tables by highlighting how many times a specific entity occurs compared to another. For example, the database of an auto repair shop may show that a mechanic works with multiple customers every day. This means that the relationship between the mechanic entity and the customer entity is one mechanic to many customers.

WHY IS CARDINALITY IMPORTANT IN DATA BASE?

Cardinality is important because it creates links from one table or entity to another in a structured manner. This has a significant impact on the query execution plan. A query execution plan is a sequence of steps users can take to search for and access data stored in a database system. Having a well-structured query execution plan can make it easier for users to locate the data they need quickly. Cardinality can be applied to databases for a variety of reasons, but businesses typically use the cardinality model to analyze information about their customers or their inventory.

For example, an online retailer may have a database table that lists each one of its unique customers. They may also have another database table that lists all the purchases customers have made from their store. Since it's likely that each customer purchased multiple items from the store, the database administrator may represent this by using a one-to-many cardinality relationship that links each customer in the first table to all the purchases they made in the second table

Types of cardinality in databases
There are three types of cardinality that may apply to a database. They are one-to-one relationships, one-to-many relationships and many-to-many relationships. Here are definitions and examples for each type of cardinality:

One-to-one relationship
A one-to-one (1:1) relationship describes a situation where one occurrence of an entity relates to exactly one occurrence of another entity. You might see this type of cardinality in a database if you're working with one row in a specific table that relates to one row in a different table.

Example: *A school may use cardinality in their student database to show a one-to-one relationship between each student and their student ID. Since the school only assigns one ID per student, using this modeling concept can help faculty members look up a student's ID number quickly if they need to create a replacement card or use it to access a specific file.*

One-to-many relationship
A one-to-many (1:Many) relationship describes a situation where one occurrence in an entity relates to many occurrences in another entity. You might see this type of cardinality in a database if you're working with one row in a specific table that relates to multiple rows in a different table.

Example: *An online food delivery service may create a table to house all of their customer ID numbers. They may also create another table for unique order ID numbers. Each order ID number must correlate with the specific customer ID number of the person who placed the order. However, a single customer may place multiple orders with the food delivery service over time.*

This means that one customer ID number may be associated with many order ID numbers. A database developer can use cardinality to establish this as a one-to-many *relationship to make it easy for other team members to connect each customer with all of the orders they've placed.*

Many-to-many relationship
A many-to-many (M:M) relationship describes a situation where multiple occurrences in one entity relate to multiple occurrences in another entity. You might see this type of cardinality in a database if you're working with several rows in a specific table that relate to several rows in another table.

Example: *A book retailer may use the many-to-many relationship model to manage their online database. They might include a list of book titles in one table and a list of author names in another table. Many of the authors may have multiple books and some of the authors may have even co-written books together.*

This means that a single author may have a relationship with more than one book and a single book may have a relationship with more than one author. By using the many-to-many relationship model, the book retailer can quickly assess which authors and books belong together.

DIFFERENCE BETWEEN CADINALITY AND MODALITY:

While cardinality and modality are both modeling concepts used in database design to analyze entities and their relationship with each other, there are some key differences between these two methods. Cardinality measures the maximum number of associations between two different table rows or columns. However, modality represents whether a relationship between two or more entities exists at all. In other words, modality focuses on the minimum number of associations, whether a relationship is mandatory and if the relationship is nullable.

DIFFERENCES BETWEEN HIGH AND LOW CARDINALITY?

High cardinality describes a data set that has a large number of unique values or entities. This represents a significant level of diversity and very little repetition. For example, a data set that lists the name of each unique customer would be considered high cardinality because the names are likely to vary.

Low cardinality refers to a data set that has a large quantity of the same values or entities. In a low cardinality data set, many of the same entities repeat themselves and there is less variety. For example, a data set that lists the category of each product for a small retail store may have low cardinality because there are only a few categories that are likely to repeat

6. WHAT IS DBA?

Database administrators (DBAs) use specialized software to **store and organize data**. The role may include capacity planning, installation, configuration, database design, migration, performance monitoring, security, troubleshooting, as well as backup and data recovery.

DBA stands for data base administrator they hold integral role in companies big and small. But the growth and capability of automation software has left some people worrying that DBAs are on their way out. They fear that automation will supplant the traditional role of the DBA.

TYPES OF DBA?

- System DBA
- Development DBA
- Application DBS

SYSTEM DBA's:

Systems DBAs focus on the physical aspects of **database management**. For example, installing and maintaining database management systems, backing up databases, optimising performance, and disaster recovery.

DEVELOPMENT DBA:

Development DBAs handle the development aspects of database administration. For example, data definition language (DDL) generation and structured query language (SQL) writing

APPLICATION DBA:

Application DBAs manage any third-party applications and software that interact with your database. (For example, automation software.) Their tasks might include installing new software applications, ensuring system integration, and managing upgrades.

It's not uncommon for one person to handle the responsibilities of all three types of DBA.

WHAT ARE DBAS ACTUALLY NEEDED FOR ?

Database administrators are critical members of the IT team. They're **overseers for your computer systems**. And, of course, they specialize in managing and maintaining an organization's databases.

DBAs are particularly integral to any business that relies heavily on their information systems. For example, banks, insurance companies, and hospitals.

In other words, database administrators oversee the creation, maintenance and security of your databases. They do this with a working knowledge of (and experience with) database management products and software.

So, what is a database administrator? They ensure the integrity, security and accessibility of a company's data.

SOME TYPICAL DBA TASKS

Database administration is a skilled, varied, and busy job. So, as a sample, day to day tasks include:

- Ensuring the security of company data
- Backing up and restoring data
- Identifying database user needs
- Creating and managing company databases
- Keeping databases running correctly and efficiently

- Making and testing modifications to database structures
- Updating old or legacy databases into new ones
- Managing cross-system data merges
- Ongoing database maintainence

A data administrator’s best friend

You might have heard a tired, stale argument that automation will **wipe out DBA roles**. In fact, the opposite is true. Rather than eliminating the field, automation software is augmenting database administration.

For example, running automation software helps DBAs with the routine maintenance of company databases. It helps keep the database up-to-date while your DBA works on it. (Rather than in it.)

Automation software (like ThinkAutomation) can help with a variety of repetitive tasks. For example, it can automatically back up new records and delete old records after a set amount of time has passed. This time would be determined by your DBA.

A DBA can also use automation to ensure the consistency of data throughout databases. Or to run automated reports. Or for queries and lookups, for tracking and measuring, to **extract, transform, and load**.

And that’s only scratching the surface. There are many more routine tasks weighing down your database administrators that automation software could handle.

But if automation software does all this, what is a database administrator for?

A helping hand, not a replacement

So, what is a database **administrator with automation**? They’re someone that can focus on optimising your company databases and information systems on every level.

Your DBA is also needed to control the automation. They manage the **rules it follows** and the processes it completes.

A DBA is also important for bigger, more complex tasks and decisions that automation isn’t capable of making