Pakistan Institute of Engineering and Applied Sciences

Art Vault

Submitted To:	**Dr. Irfan Hameed**
Submitted By:	**Ehmaan Shafqat**
Course:	**Database system**

Description:

ArtVault is a comprehensive art gallery management system designed to streamline operations by managing artworks, artists, customers, orders, events, and employees. It tracks artwork details, artist profiles, customer orders, event management, and employee assignments. The system also handles ticketing, sales, and inventory management, ensuring smooth gallery operations. With a relational database structure, ArtVault provides an efficient platform for managing and analyzing gallery data, improving operational efficiency and customer experience.

Business rules:

Following are the business rules for this project:

1. **Person Identification:**
   - Each person must have a unique CNIC for identification.
   - Every person can act as a customer, employee, or artist, but basic personal information is stored in a single Person record.
2. **Artwork Management:**
   - Each artwork must be associated with exactly **One Artist**.
   - Each artwork must have a **Unique Identifier**.
   - There is only **One Piece** of each Artwork.
   - Artworks can only have a status of **"Available"**, **"Sold"**, or **"On Loan"**.
   - An artwork’s SellDate cannot be before its StockedDate.
3. **Artist Management:**
   - Each **Artist** must have a unique identifier (ArtistID).
   - Artists can create **Multiple Artworks**, but an artwork must belong to only **One Artist.**
   - An artist cannot be deleted if they have associated artworks in the system.
4. **Customer Management:**
   - Each **Customer** must have a unique identifier (CustomerID).
   - A customer can purchase **Multiple artworks**.
5. **Order Management:**
   - Every **Order** must be linked to exactly one customer.
   - **Order** can have multiple Order lines.
   - An order line cannot reference the same artwork multiple times for the same order.
6. **Event Management:**
   - Each **Event** must have a unique identifier (EventID).
   - An exhibition can showcase **Multiple Artworks**, and an artwork can be displayed in **Multiple Events**.
   - An event can have **Multiple Employees,** and an employee can be in **Multiple Events.**
7. **Price Management:**
   - No price or ticket records in ArtworkPrice or TicketPrice may be deleted, ensuring an audit trail**.**
8. **Employee Management:**
   - Each **Employee** must have a unique identifier (EmployeeID).
   - An employee may be assigned to multiple events, but their role must be specified for each event.

Entities:

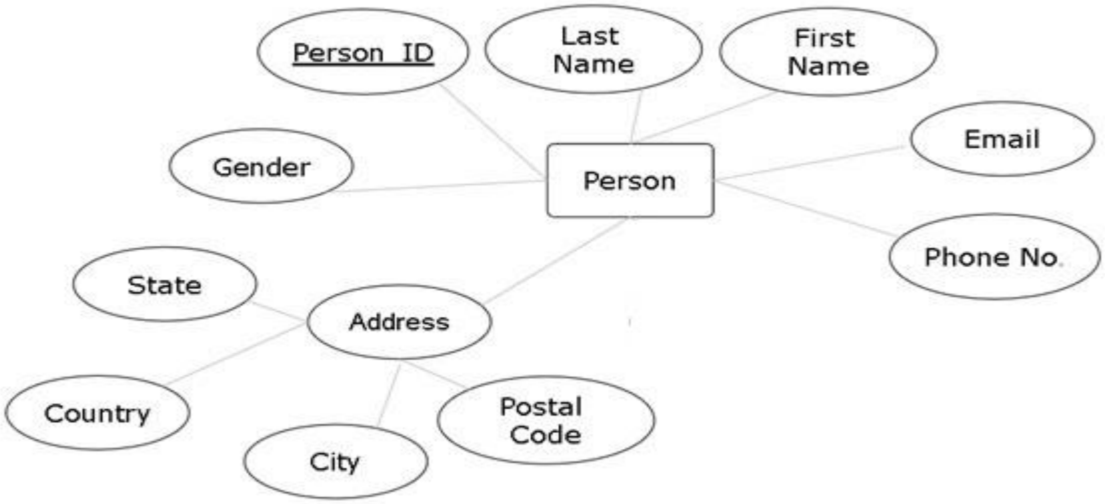
For ArtVault Database mangemet we select below entities:

1. Person:

Description: The Person entity represents the basic personal information of all individuals involved in the system, such as customers, employees, and artists. It includes common attributes such as PERSON_ID, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, and CONTACT_DETAILS. This entity serves as a foundation for other entities like Customer, Employee, and Artist, linking them to specific individuals.

Attributes:

- PERSON_ID (PK)
- FIRST_NAME
- LAST_NAME
- DATE_OF_BIRTH
- CONTACT_DETAILS

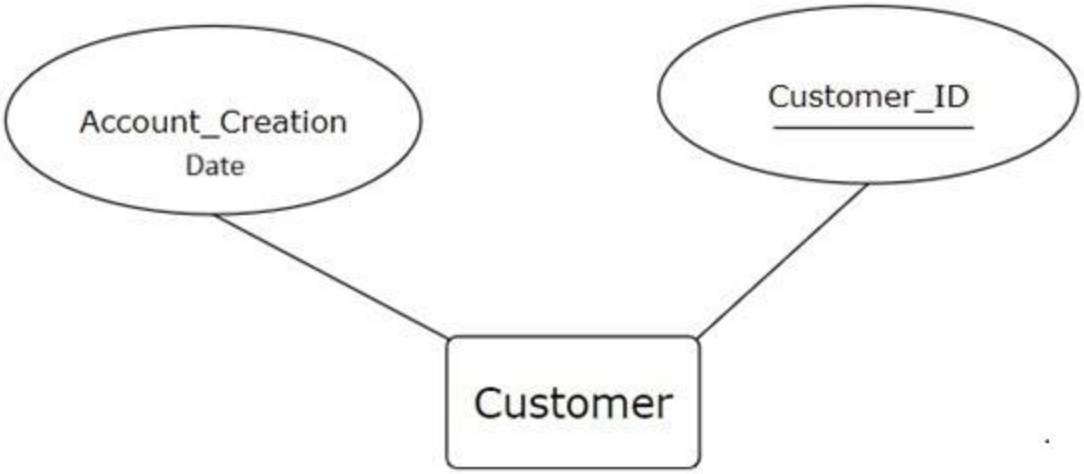


2. Customer:

Description: The Customer entity represents individuals who make purchases (orders) from the art gallery. It inherits from the Person entity, which means it shares personal information attributes like name and contact details. Additionally, the Customer entity includes specific customer attributes like ACCOUNT_CREATION_DATE and a link to PERSON_ID from the Person entity.

Attributes:

- CUSTOMER_ID (PK, FK referencing PERSON_ID)
- ACCOUNT_CREATION_DATE
- STATUS

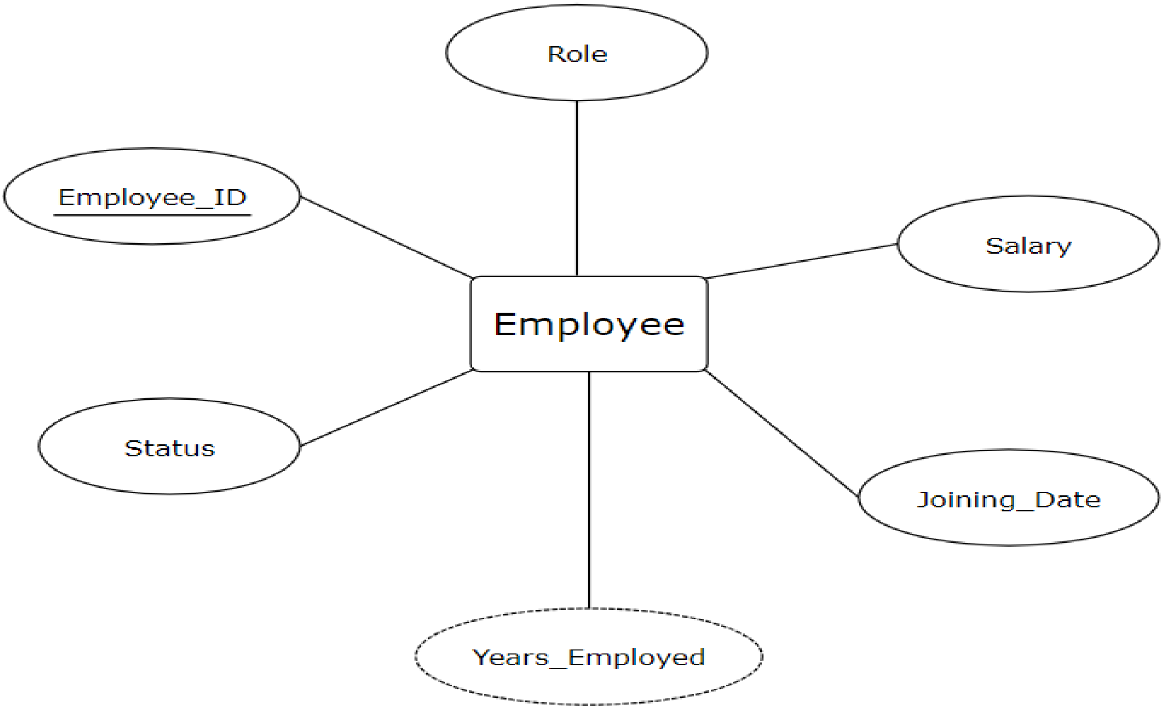


3. Employee:

Description: The Employee entity represents individuals employed by the art gallery in various roles. It inherits from the Person entity. Employees can have different roles such as Manager, Coordinator, or Assistant. The Employee entity includes attributes like ROLE, SALARY, STATUS, and JOINING_DATE, along with a link to PERSON_ID from the Person entity.

Attributes:

- EMPLOYEE_ID (PK, FK referencing PERSON_ID)
- ROLE
- SALARY
- STATUS
- JOINING_DATE

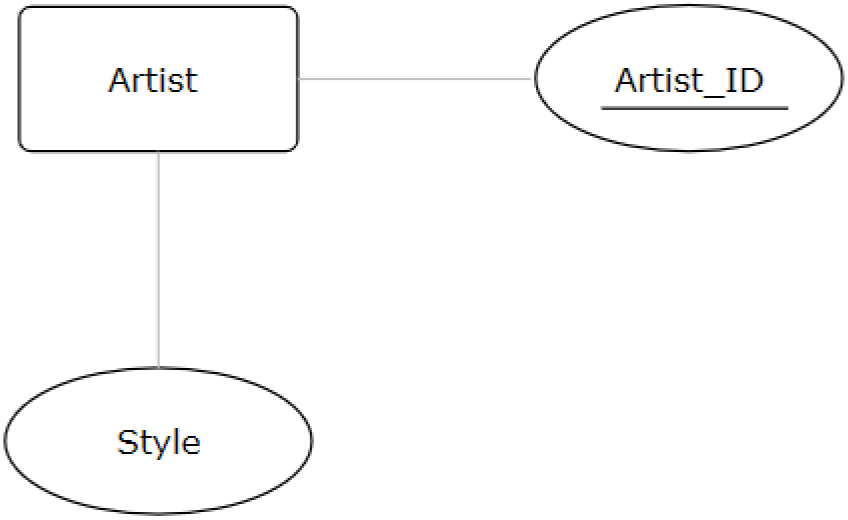


4. Artist:

Description: The Artist entity represents individuals who create artworks displayed in the gallery. Like Employee and Customer, the Artist entity inherits from the Person entity. It may include specific attributes such as BIOGRAPHY, GENRE, and DEBUT_YEAR, along with a reference to the PERSON_ID of the individual artist.

Attributes:

- ARTIST_ID (PK, FK referencing PERSON_ID)
- BIOGRAPHY
- GENRE
- DEBUT_YEAR

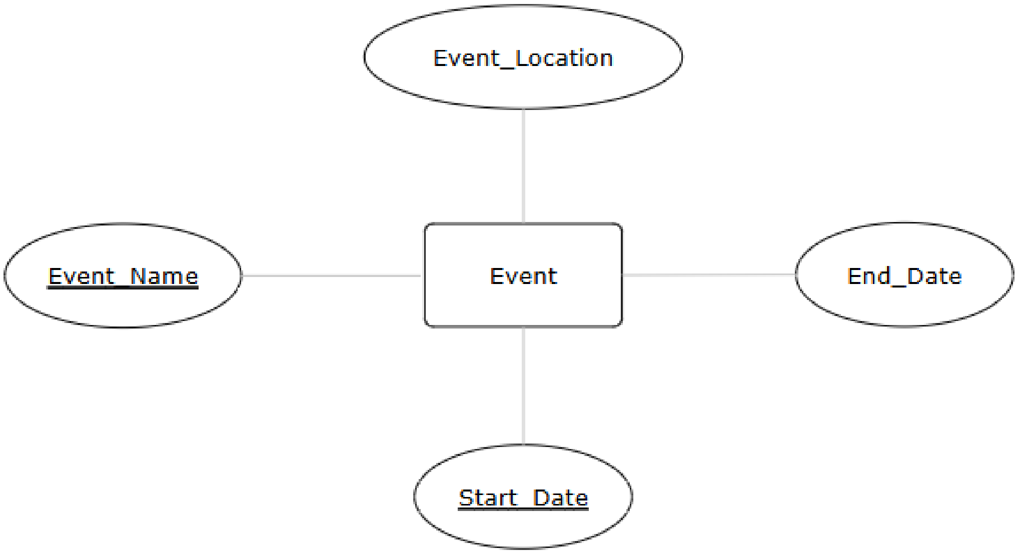


5. Event:

Description: The Event entity represents various events or exhibitions held at the gallery, such as art shows or fairs. It includes event details like EVENT_NAME, START_DATE, END_DATE, and EVENT_LOCATION. The EVENT entity may also be linked to employees who are working at these events.

Attributes:

- EVENT_NAME (PK)
- START_DATE
- END_DATE
- EVENT_LOCATION

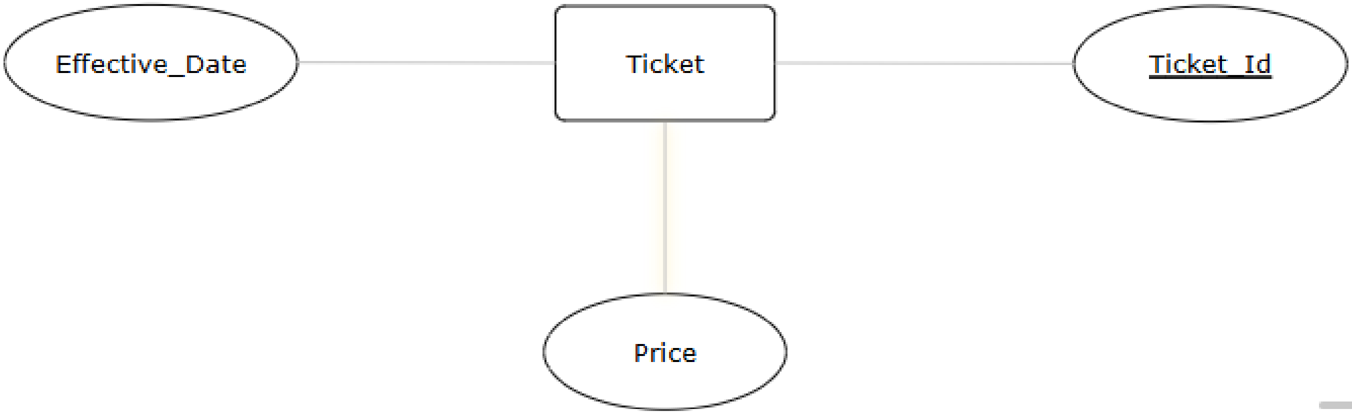


Ticket:

Description: The Ticket entity represents tickets sold for events. It includes TICKET_ID, EVENT_NAME, START_DATE, TICKET_PRICE, and EFFECTIVE_DATE, linking it to a specific event. The TICKET entity ensures the pricing is tracked for each event.

Attributes:

- TICKET_ID (PK)
- EVENT_NAME (FK referencing Event)
- START_DATE (FK referencing Event)
- TICKET_PRICE
- EFFECTIVE_DATE

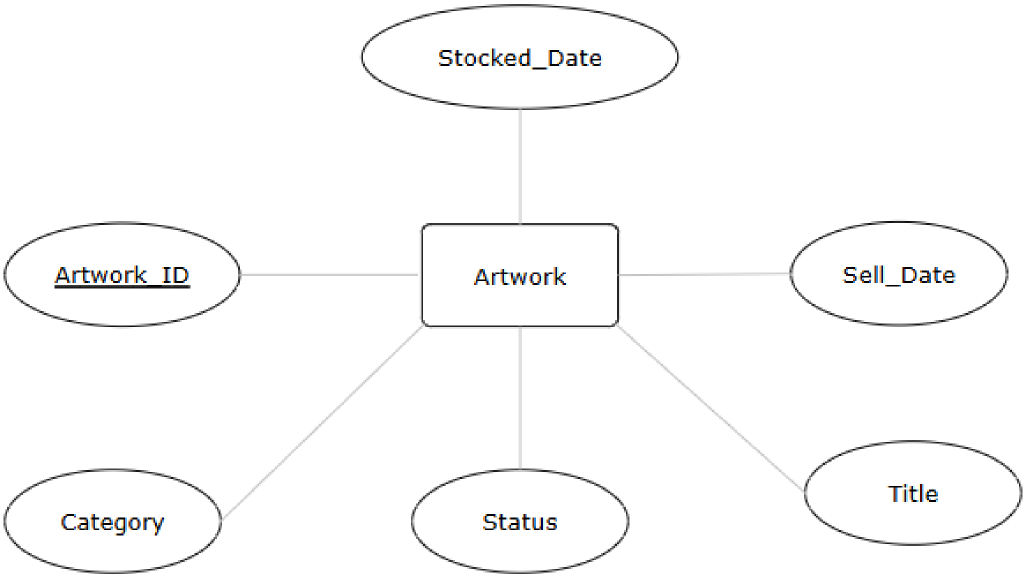


6. Artwork:

Description: The Artwork entity represents the individual pieces of art available for purchase or display in the gallery. It includes attributes such as ARTWORK_ID, TITLE, CATEGORY, ARTIST_ID, STOCKED_DATE, and STATUS. The ARTWORK entity links to the Artist entity via ARTIST_ID.

Attributes:

- ARTWORK_ID (PK)
- TITLE
- CATEGORY
- ARTIST_ID (FK referencing Artist)
- STOCKED_DATE
- STATUS

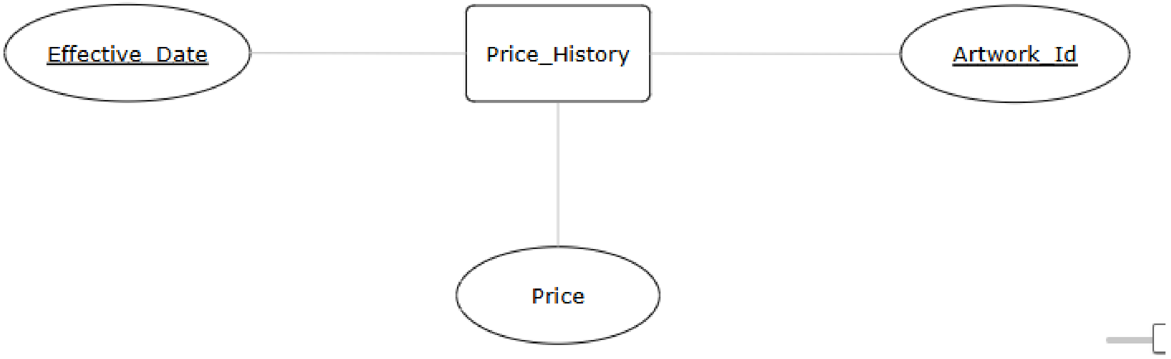


Price_History:

Description: The Price_History entity represents the price of an artwork at specific dates. This allows the gallery to track price changes over time for each artwork. It includes ARTWORK_ID, PRICE, and EFFECTIVE_DATE, forming a composite key that associates prices with particular effective dates.

Attributes:

- ARTWORK_ID (PK, FK referencing Artwork)
- PRICE
- EFFECTIVE_DATE

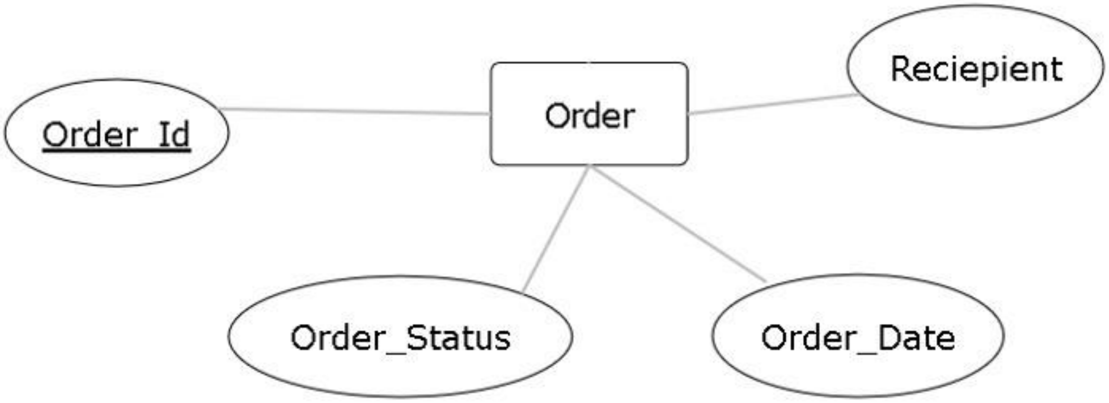


7. Order:

Description: The Order entity represents a customer's purchase request in the art gallery system. It includes order-specific details like ORDER_ID, ORDER_DATE, ORDER_STATUS, and a reference to the CUSTOMER_ID, linking it to a customer who placed the order. Orders may contain multiple artworks, represented through the OrderLine entity.

Attributes:

- ORDER_ID (PK)
- ORDER_DATE
- ORDER_STATUS
- CUSTOMER_ID (FK referencing Customer)

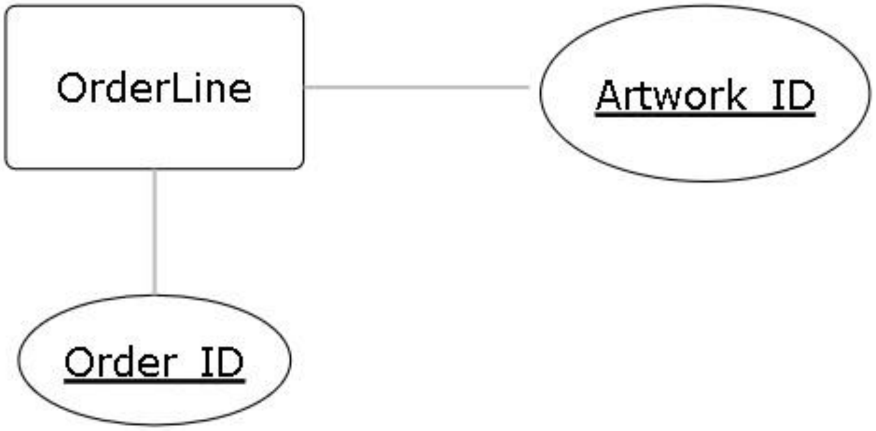


8. Order_line:

Description: The OrderLine entity represents individual items within a specific order, linking an artwork to an order. It consists of the ORDER_ID and ARTWORK_ID, forming a composite primary key. The ORDERLINE table allows a many-to-many relationship between Order and Artwork, where an order can contain multiple artworks.

Attributes:

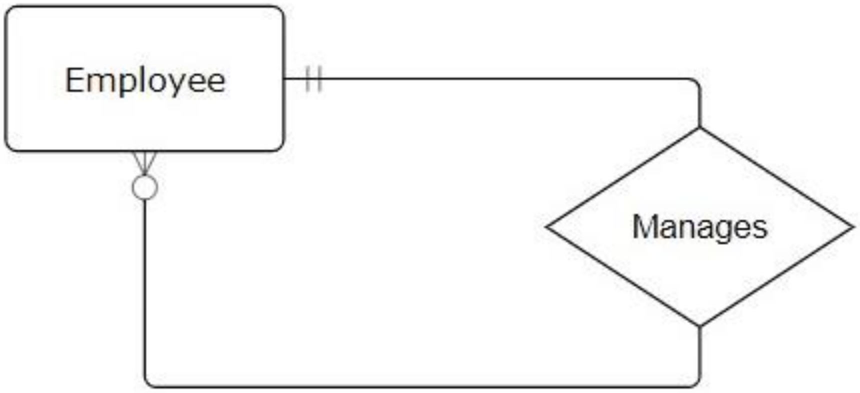
- ORDER_ID (PK, FK referencing Order)
- ARTWORK_ID (PK, FK referencing Artwork)



Relationships:

Unary Relationship:

In an employee hierarchy, an Employee entity might have a unary relationship where an employee has a "supervisor" who is also an employee. This relationship would link an employee to another employee, indicating the supervisory role.



Binary Relationship:

ORDER_T and CUSTOMER:

- **Relationship**: A customer can place multiple orders, but each order is placed by one customer.
- **Cardinality**: One-to-Many (1:N) from CUSTOMER to ORDER_T.
- **Foreign Key**: CUSTOMER_ID in ORDER_T references CUSTOMER_ID in CUSTOMER.

ORDER_T and PERSON (via CUSTOMER):

- **Relationship**: Each customer is represented by a person, and orders contain customer details.
- **Cardinality**: One-to-One (1:1) between CUSTOMER and PERSON.
- **Foreign Key**: CUSTOMER_ID in ORDER_T references PERSON_ID in PERSON (through CUSTOMER).

ARTWORK and ARTIST:

- **Relationship**: Each artwork is created by one artist, and an artist can create multiple artworks.
- **Cardinality**: One-to-Many (1:N) from ARTIST to ARTWORK.
- **Foreign Key**: ARTIST_ID in ARTWORK references ARTIST_ID in ARTIST.

ARTWORK and ARTWORK_PRICE:

- **Relationship**: An artwork can have multiple prices over time, each with a specific effective date.
- **Cardinality**: One-to-Many (1:N) from ARTWORK to ARTWORK_PRICE.
- **Foreign Key**: ARTWORK_ID in ARTWORK_PRICE references ARTWORK_ID in ARTWORK.

EVENT and EVENT_EMPLOYEE:

- **Relationship**: An event involves multiple employees, and an employee can work at multiple events.
- **Cardinality**: Many-to-Many (M:N) between EVENT and EMPLOYEE, represented through the associative entity EVENT_EMPLOYEE.
- **Foreign Keys**:
  - EVENT_NAME and START_DATE in EVENT_EMPLOYEE reference EVENT_NAME and START_DATE in EVENT.
  - EMPLOYEE_ID in EVENT_EMPLOYEE references EMPLOYEE_ID in EMPLOYEE.

EVENT and ARTWORK:

- **Relationship**: An event can feature multiple artworks, and an artwork can be part of multiple events.
- **Cardinality**: Many-to-Many (M:N) between EVENT and ARTWORK, represented through the associative entity EVENT_ARTWORK.
- **Foreign Keys**:
  - EVENT_NAME and START_DATE in EVENT_ARTWORK reference EVENT_NAME and START_DATE in EVENT.
  - ARTWORK_ID in EVENT_ARTWORK references ARTWORK_ID in ARTWORK.

TICKET and EVENT:

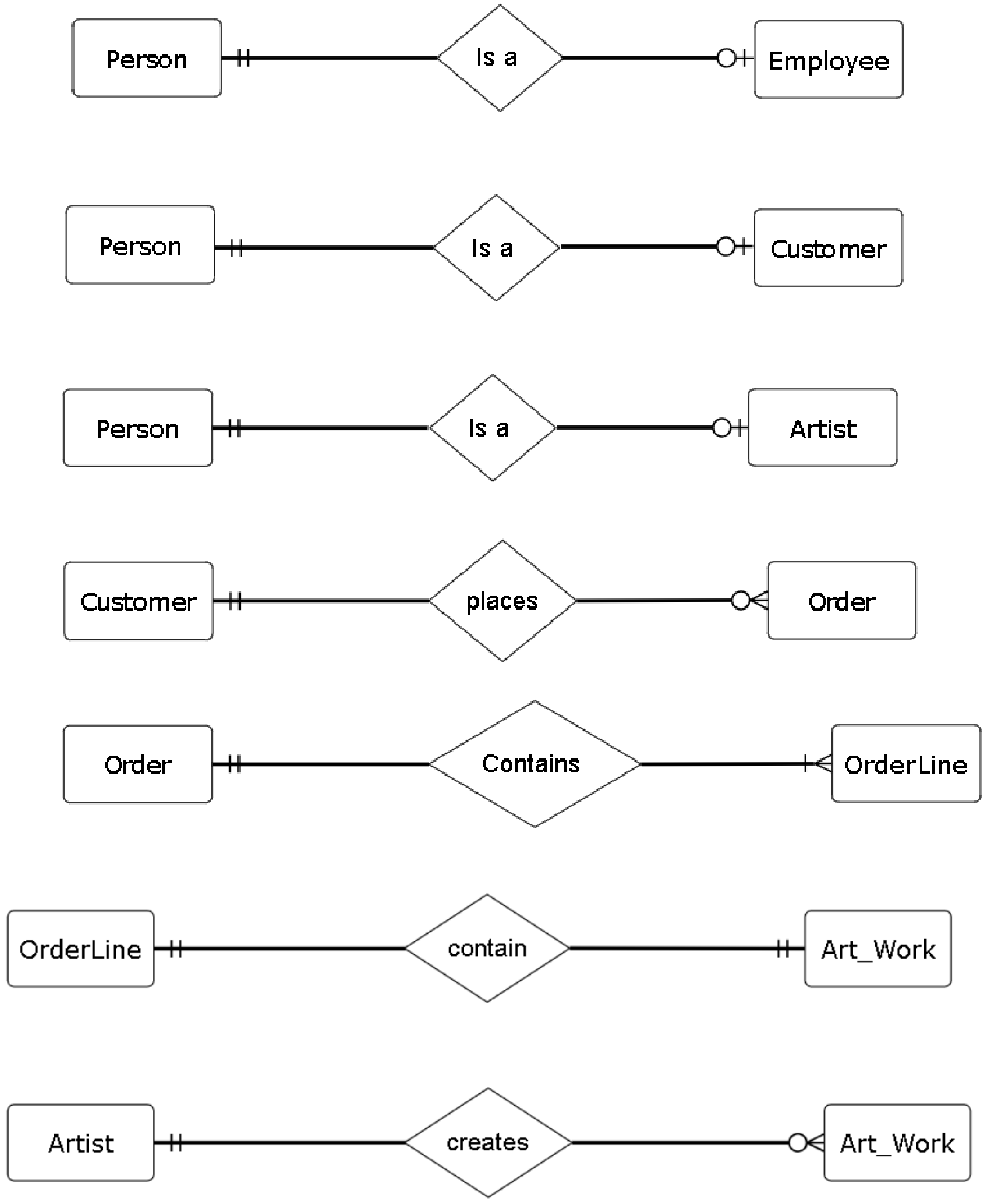
- **Relationship**: A ticket is associated with a specific event, and an event can have multiple tickets sold.
- **Cardinality**: One-to-Many (1:N) from EVENT to TICKET.
- **Foreign Key**: EVENT_NAME and START_DATE in TICKET reference EVENT_NAME and START_DATE in EVENT.

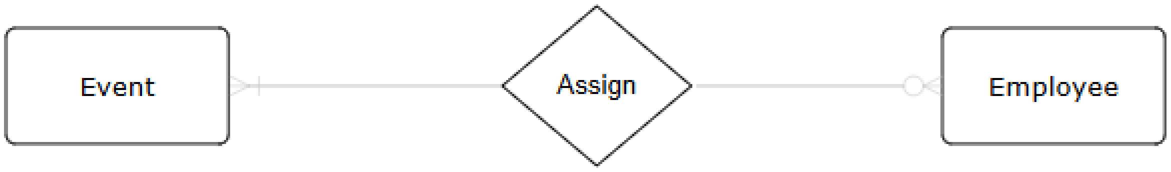
ORDER_T and ORDERLINE:

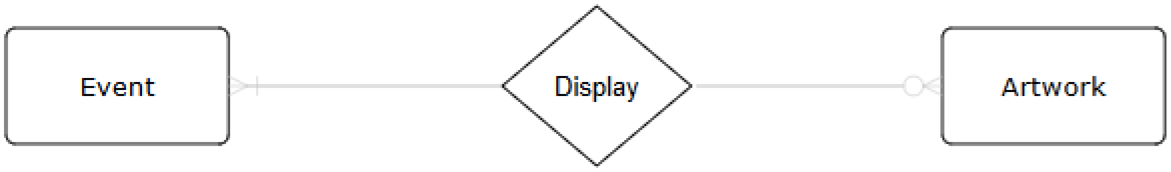
- **Relationship**: Each order can contain multiple artworks, and each artwork in an order is represented by an order line.
- **Cardinality**: One-to-Many (1:N) from ORDER_T to ORDERLINE.
- **Foreign Key**: ORDER_ID in ORDERLINE references ORDER_ID in ORDER_T.

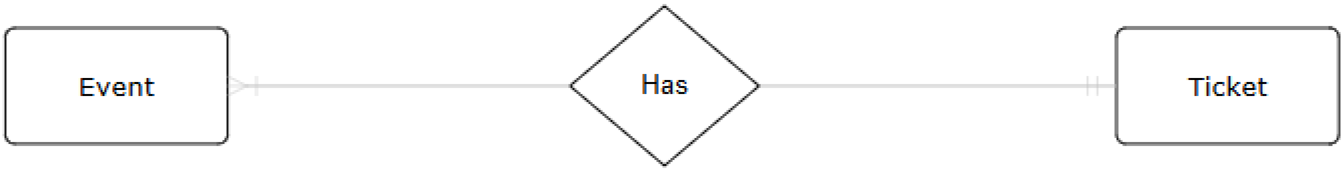
ORDERLINE and ARTWORK:

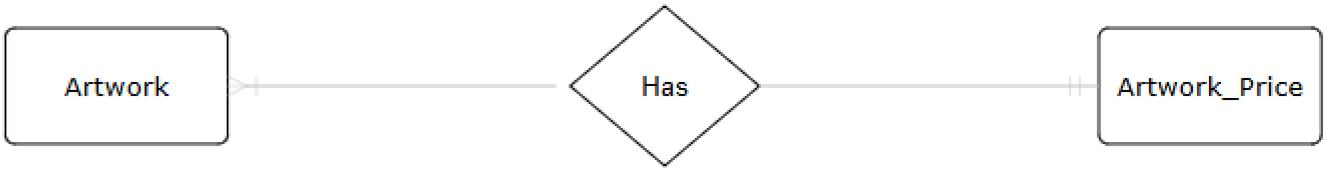
- **Relationship**: Each order line corresponds to a specific artwork, and an artwork can be included in multiple order lines.
- **Cardinality**: Many-to-One (N:1) from ORDERLINE to ARTWORK.
- **Foreign Key**: ARTWORK_ID in ORDERLINE references ARTWORK_ID in ARTWORK.


Person
Is a
Employee
Person
Is a
Customer
Person
Is a
Artist
Customer
places
Order
Order
Contains
OrderLine
OrderLine
contain
Art_Work
Artist
creates
Art_Work









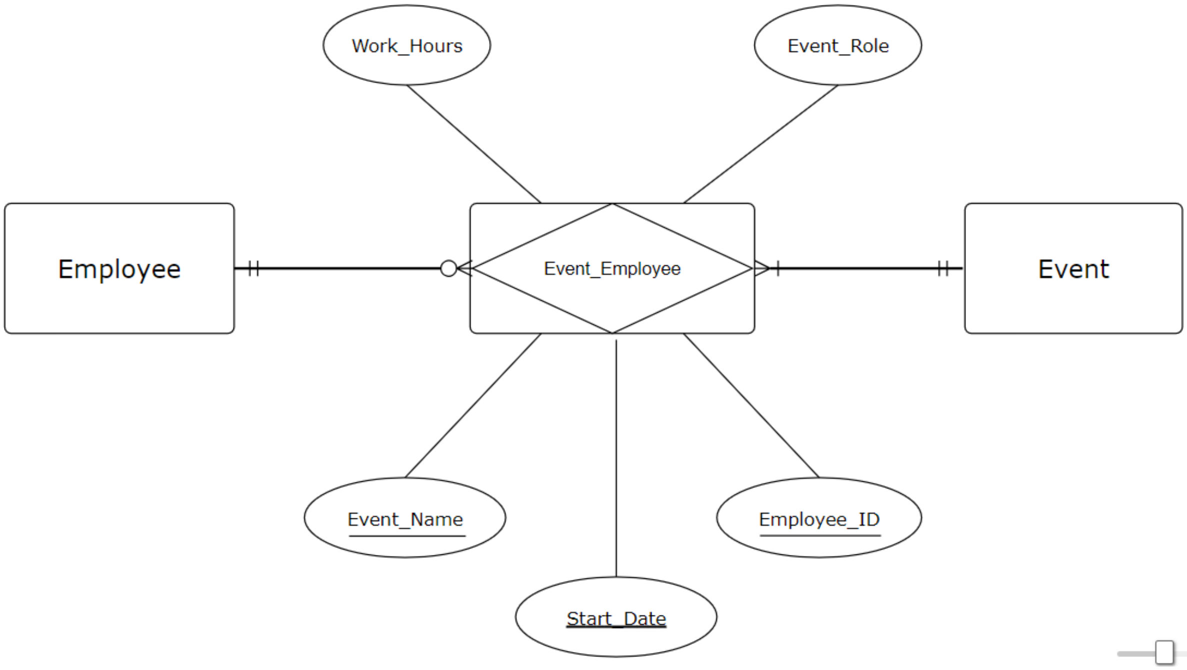
Associative Entities:

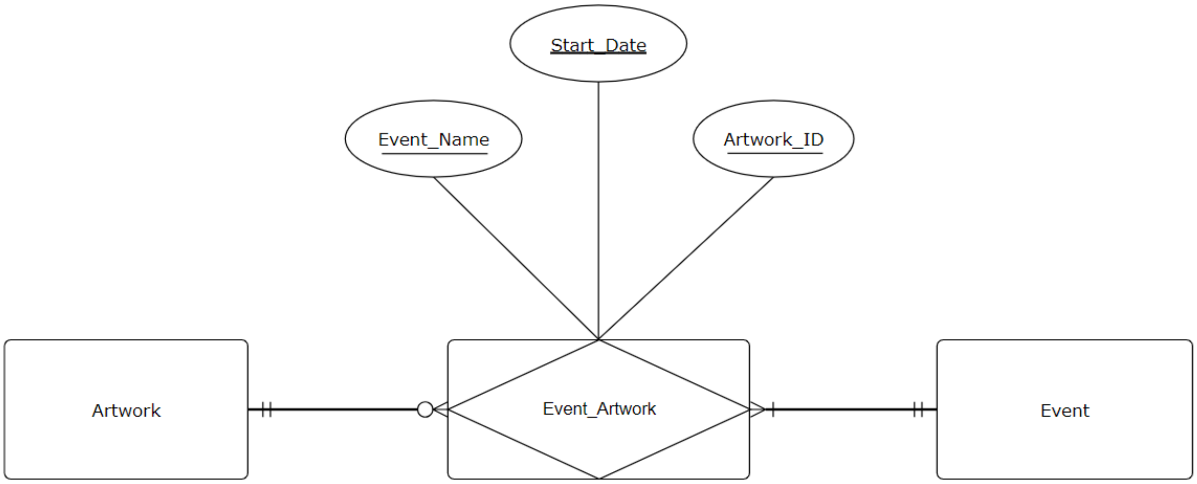
1. **EVENT_ARTWORK (Associative Entity)**:

- **Purpose**: This entity represents the many-to-many relationship between **EVENT** and **ARTWORK**. It tracks which artworks are featured in which events.
- **Attributes**:
  - EVENT_NAME, START_DATE (Foreign Keys referencing the EVENT table)
  - ARTWORK_ID (Foreign Key referencing the ARTWORK table)
- **Primary Key**: Composite of EVENT_NAME, START_DATE, and ARTWORK_ID.
- **Relationship**: An event can feature multiple artworks, and an artwork can be part of multiple events.

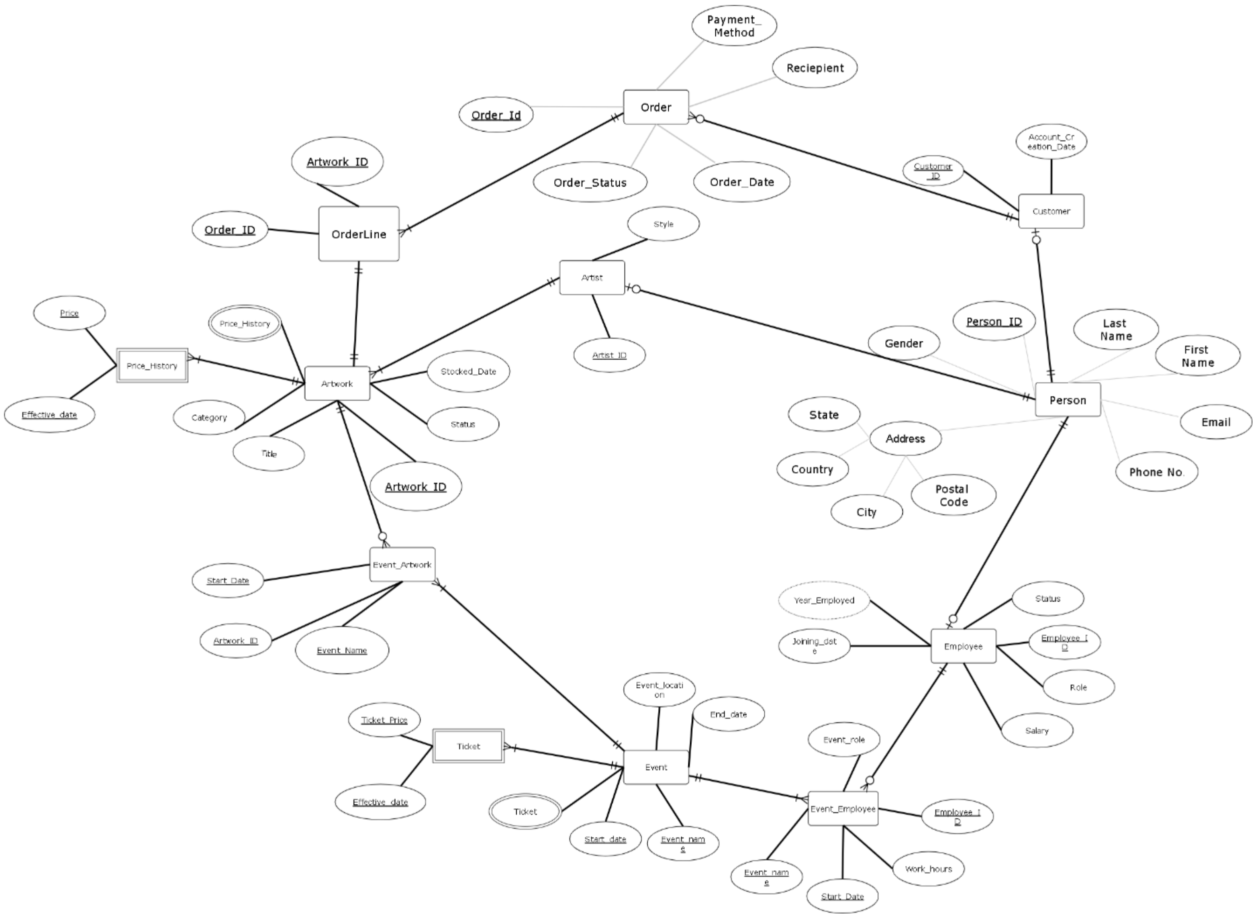
2. **EVENT_EMPLOYEE (Associative Entity)**:

- **Purpose**: This entity represents the many-to-many relationship between **EVENT** and **EMPLOYEE**. It tracks which employees are assigned to which events and their roles.
- **Attributes**:
  - EVENT_NAME, START_DATE (Foreign Keys referencing the EVENT table)
  - EMPLOYEE_ID (Foreign Key referencing the EMPLOYEE table)
  - WORK_HOURS, EVENT_ROLE
- **Primary Key**: Composite of EVENT_NAME, EMPLOYEE_ID, and START_DATE.
- **Relationship**: An event involves multiple employees, and an employee can work at multiple events in different roles.





Complete ER Diagram:



Relational Model:

Description:

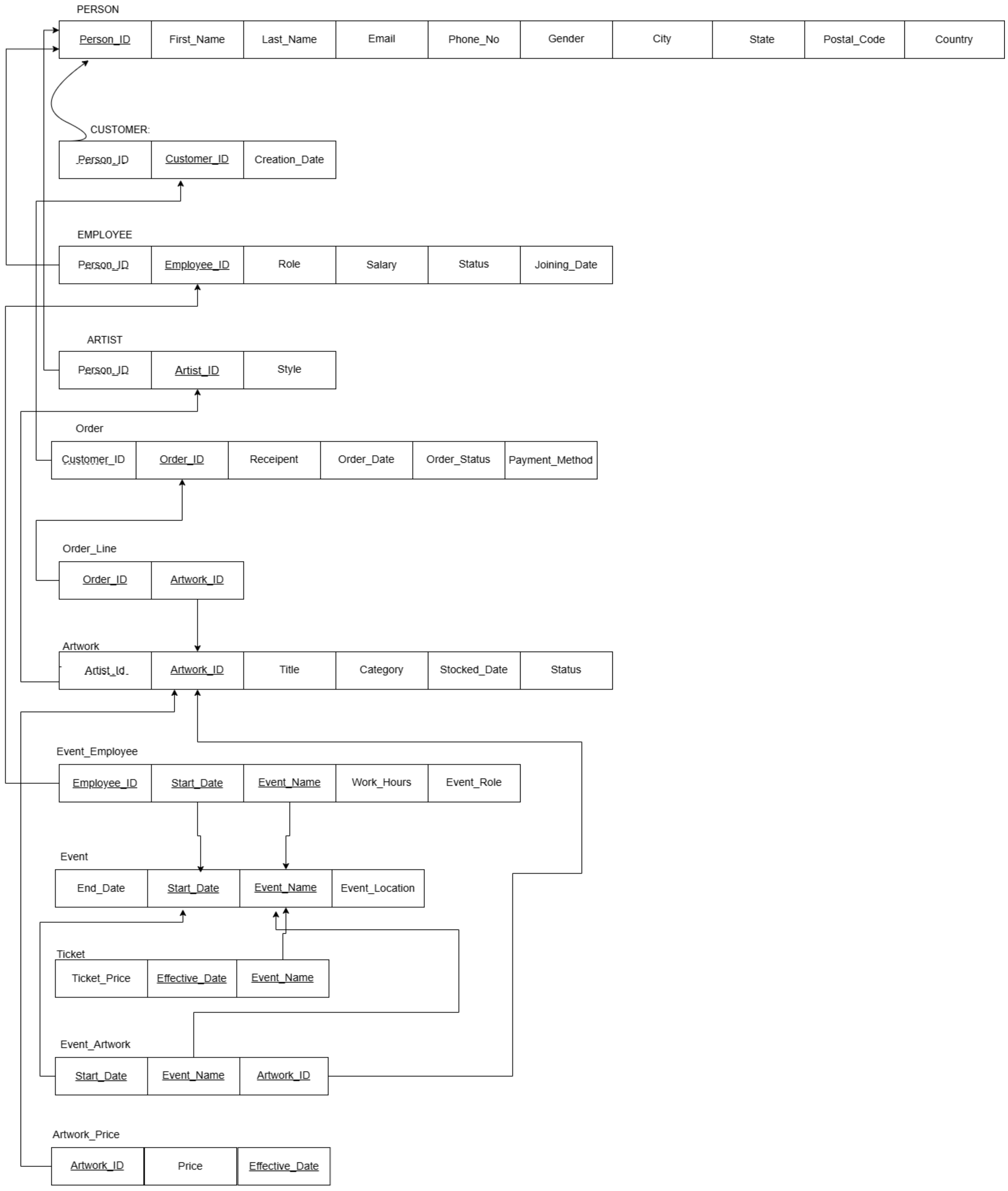
1. **PERSON**: Stores personal details of individuals.
   - **Attributes**: PERSON_ID (Primary Key), FIRST_NAME, LAST_NAME, CONTACT_INFO, etc.
2. **CUSTOMER**: Represents customers and inherits from PERSON.
   - **Attributes**: CUSTOMER_ID (Primary Key, Foreign Key referencing PERSON), ACCOUNT_CREATION_DATE.
3. **EMPLOYEE**: Represents employees and inherits from PERSON.
   - **Attributes**: EMPLOYEE_ID (Primary Key, Foreign Key referencing PERSON), ROLE, SALARY, STATUS, JOINING_DATE.
4. **ARTIST**: Represents artists, also inheriting from PERSON.
   - **Attributes**: ARTIST_ID (Primary Key, Foreign Key referencing PERSON), STYLE.
5. **ORDER_T**: Represents customer orders.
   - **Attributes**: ORDER_ID (Primary Key), CUSTOMER_ID (Foreign Key referencing CUSTOMER), RECIPIENT, ORDER_DATE, ORDER_STATUS, PAYMENT_METHOD.
6. **ORDERLINE**: Associates multiple artworks with orders.
   - **Attributes**: ORDER_ID, ARTWORK_ID (Foreign Keys referencing ORDER_T and ARTWORK respectively), Primary Key is a composite of ORDER_ID and ARTWORK_ID.
7. **ARTWORK**: Represents artwork items.
   - **Attributes**: ARTWORK_ID (Primary Key), TITLE, CATEGORY, ARTIST_ID (Foreign Key referencing ARTIST), STOCKED_DATE, STATUS.
8. **ARTWORK_PRICE**: Represents the pricing history of artworks.
   - **Attributes**: ARTWORK_ID (Foreign Key referencing ARTWORK), PRICE, EFFECTIVE_DATE, Primary Key is a composite of ARTWORK_ID and EFFECTIVE_DATE.
9. **EVENT**: Represents an event in which artworks are displayed.
   - **Attributes**: EVENT_NAME, START_DATE (Primary Key), END_DATE, EVENT_LOCATION.
10. **TICKET**: Represents tickets sold for events.
    - **Attributes**: TICKET_ID (Primary Key), EVENT_NAME, START_DATE (Foreign Keys referencing EVENT), TICKET_PRICE, EFFECTIVE_DATE.
11. **EVENT_ARTWORK**: Represents the many-to-many relationship between EVENT and ARTWORK.
    - **Attributes**: EVENT_NAME, START_DATE, ARTWORK_ID (Foreign Keys referencing EVENT and ARTWORK), Primary Key is a composite of these three attributes.
12. **EVENT_EMPLOYEE**: Represents the many-to-many relationship between EVENT and EMPLOYEE.
    - **Attributes**: EVENT_NAME, START_DATE, EMPLOYEE_ID (Foreign Keys referencing EVENT and EMPLOYEE), WORK_HOURS, EVENT_ROLE, Primary Key is a composite of EVENT_NAME, EMPLOYEE_ID, and START_DATE.

Key Relationships:

- **Person ↔ Customer, Employee, Artist**: Inheritance relationship, where PERSON serves as the base entity.
- **Order_T ↔ Customer**: A customer can place many orders, with each order linked to a specific customer.
- **Order_T ↔ Artwork**: Through ORDERLINE, an order can include multiple artworks, and an artwork can appear in multiple orders.
- **Artwork ↔ Artist**: Each artwork is associated with a specific artist.
- **Event ↔ Artwork**: Many artworks can be displayed at multiple events via EVENT_ARTWORK.

- **Event ↔ Employee**: Employees are assigned to events via EVENT_EMPLOYEE, allowing them to work at multiple events.
- **Event ↔ Ticket**: Each event has tickets, with their prices recorded.
- **Event ↔ Artwork, Event ↔ Employee**: Both of these are managed through associative entities (EVENT_ARTWORK and EVENT_EMPLOYEE), representing the many-to-many relationships.

RELATIONAL SCHEMA



PERSON

Person_ID	First_Name	Last_Name	Email	Phone_No	Gender	City	State	Postal_Code	Country

CUSTOMER:

Person_ID	Customer_ID	Creation_Date

EMPLOYEE

Person_ID	Employee_ID	Role	Salary	Status	Joining_Date

ARTIST

Person_ID	Artist_ID	Style

Order

Customer_ID	Order_ID	Receipent	Order_Date	Order_Status	Payment_Method

Order_Line

Order_ID	Artwork_ID

Artwork

Artist_Id	Artwork_ID	Title	Category	Stocked_Date	Status

Event_Employee

Employee_ID	Start_Date	Event_Name	Work_Hours	Event_Role

Event

End_Date	Start_Date	Event_Name	Event_Location

Ticket

Ticket_Price	Effective_Date	Event_Name

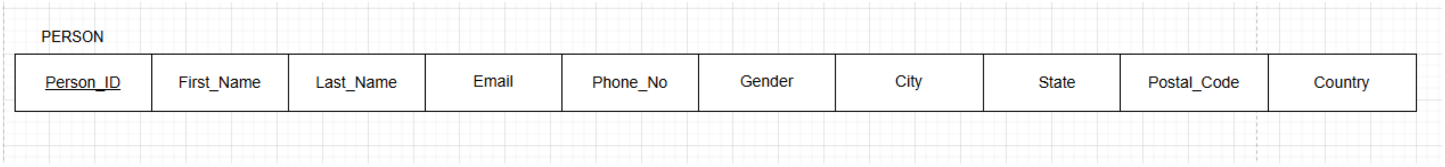
Event_Artwork

Start_Date	Event_Name	Artwork_ID

Artwork_Price

Artwork_ID	Price	Effective_Date

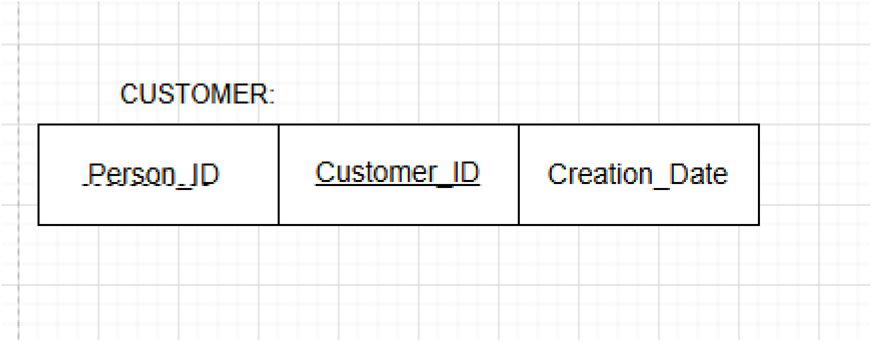
Normalized Schema:



PERSON

Person_ID	First_Name	Last_Name	Email	Phone_No	Gender	City	State	Postal_Code	Country

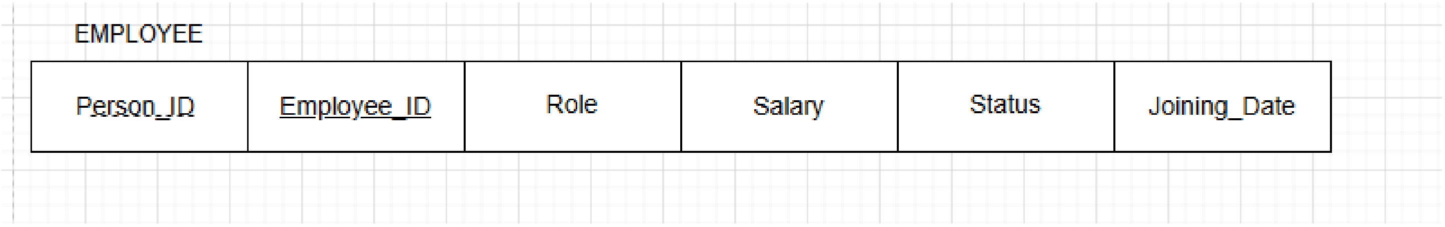
- All attributes of Person are atomic
- There is no partial and transitive dependency
- So person is in 3rd Normal Form



CUSTOMER:

Person_ID	Customer_ID	Creation_Date

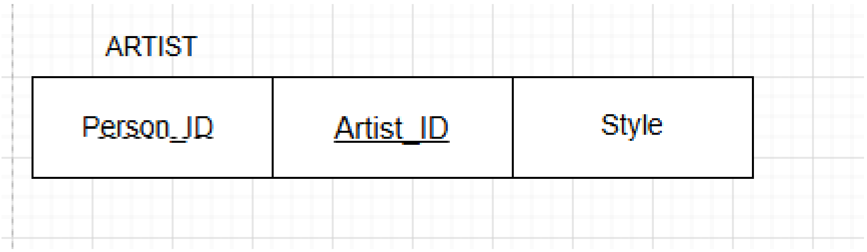
- All attributes of Customer are atomic
- There is no partial and transitive dependency
- So Customer is in 3rd Normal Form



EMPLOYEE

Person_ID	Employee_ID	Role	Salary	Status	Joining_Date

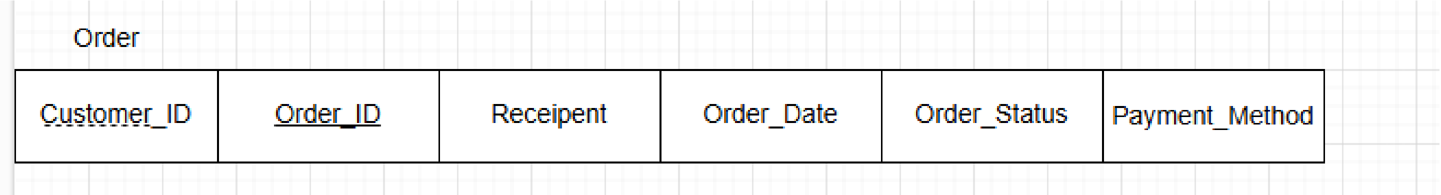
- All attributes of Employee are atomic
- There is no partial and transitive dependency
- So Employee is in 3rd Normal Form



ARTIST

Person_ID	Artist_ID	Style

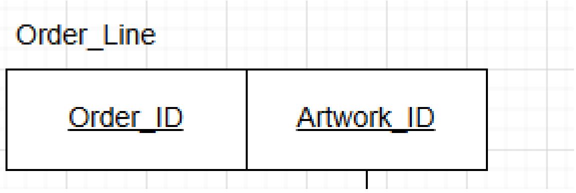
- All attributes of Artist are atomic
- There is no partial and transitive dependency
- So Artist is in 3rd Normal Form



Order

Customer_ID	Order_ID	Receipent	Order_Date	Order_Status	Payment_Method

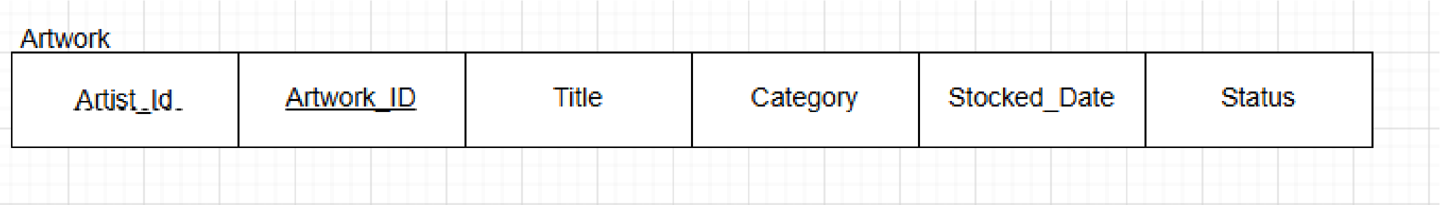
- All attributes of Order are atomic
- There is no partial and transitive dependency
- So Order is in 3rd Normal Form



Order_Line

Order_ID	Artwork_ID

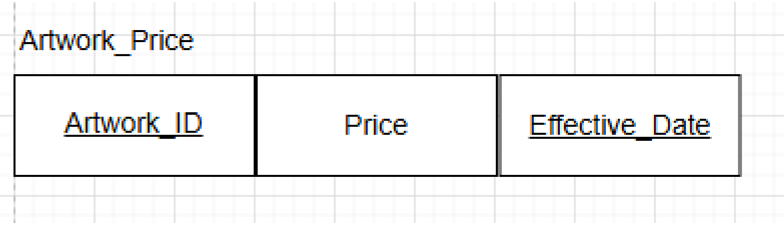
- All attributes of Order_Line are atomic
- There is no partial and transitive dependency
- So Order_Line is in 3rd Normal Form



Artwork

Artist_Id	Artwork_ID	Title	Category	Stocked_Date	Status

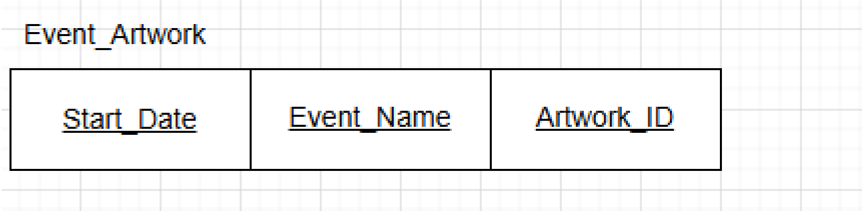
- All attributes of Artwork are atomic
- There is no partial and transitive dependency
- So Artwork is in 3rd Normal Form



Artwork_Price

Artwork_ID	Price	Effective_Date

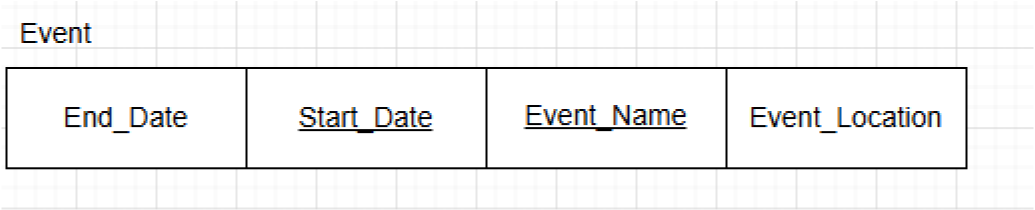
- All attributes of Artwork_Price are atomic
- There is no partial and transitive dependency
- So Artwork_Price is in 3rd Normal Form



Event_Artwork

Start_Date	Event_Name	Artwork_ID

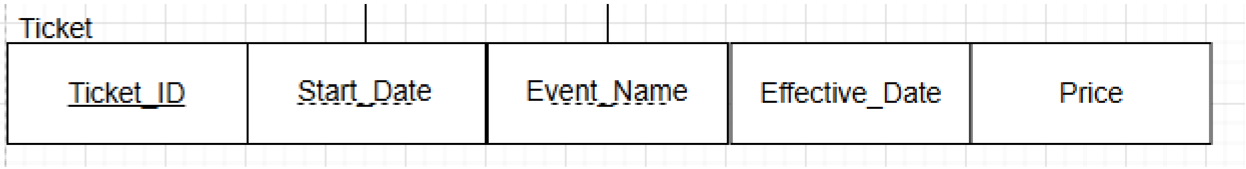
- All attributes of Event_Artwork are atomic
- There is no partial and transitive dependency
- So Event_Artwork is in 3rd Normal Form



Event

End_Date	Start_Date	Event_Name	Event_Location

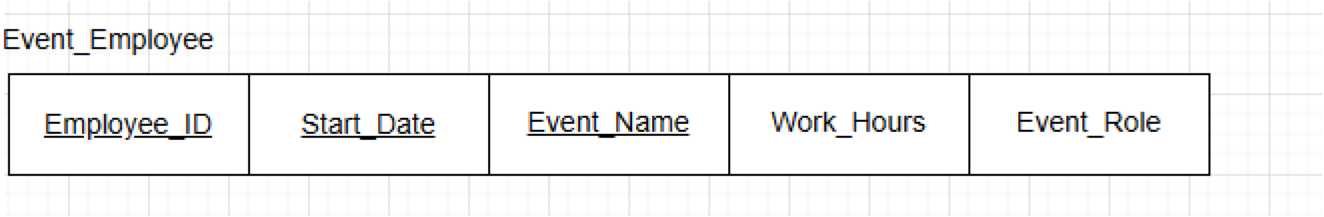
- All attributes of Event are atomic
- There is no partial and transitive dependency
- So Event is in 3rd Normal Form



Ticket

Ticket_ID	Start_Date	Event_Name	Effective_Date	Price

- All attributes of Ticket are atomic
- There is no partial and transitive dependency
- So Ticket is in 3rd Normal Form



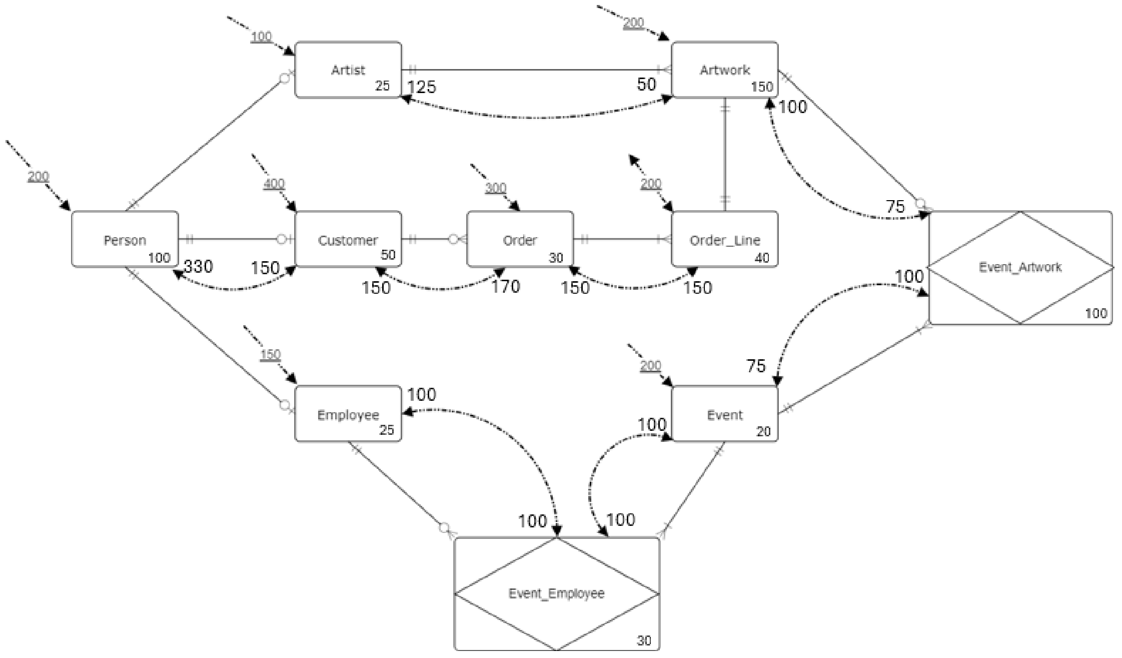
Event_Employee

Employee_ID	Start_Date	Event_Name	Work_Hours	Event_Role

- All attributes of Event_Employee are atomic
- There is no partial and transitive dependency
- So Event_Employee is in 3rd Normal Form

Composite Usage Model:

Access Frequency (Per Hour):



Code:

Creating table:

Person:

The PERSON table serves as a foundational entity in the database to store general information about individuals. This table can act as a **supertype** for other entities such as customers, employees, or artists.

Key Features:

1. **Primary Key**:
   - PERSON_ID uniquely identifies each person in the table and ensures no duplicate records exist.
   - A CHECK constraint enforces a specific format: NNNNN-NNNNNNN-N, where N represents digits, aligning with standard identification formats.
2. **Attributes**:
   - FIRST_NAME and LAST_NAME: Store the individual's name.
   - EMAIL: Optional field for storing the person's email address.
   - PHONE_NO: Mandatory contact number for the person.
   - GENDER: Captures the gender of the individual using a single character (M, F, etc.).
   - CITY, STATE, POSTAL_CODE, and COUNTRY: These attributes collectively store the address details of the person.
3. **Constraints**:

- person_pk: Ensures PERSON_ID is a unique and non-null value.
- person_id_format_check: Validates the PERSON_ID format using a **regular expression** to maintain data consistency.

Use Case:

The PERSON table can be extended through foreign key relationships in specialized entities like CUSTOMER, EMPLOYEE, or ARTIST to avoid redundancy and maintain a clean data structure. This makes the table a versatile base for managing individuals' information.

```
CREATE TABLE PERSON(
  PERSON_ID       VARCHAR2(15)  NOT NULL,
  FIRST_NAME      VARCHAR2(20)  NOT NULL,
  LAST_NAME       VARCHAR2(20),
  EMAIL           VARCHAR2(40),
  PHONE_NO        VARCHAR2(20)  NOT NULL,
  GENDER          CHAR(1)       NOT NULL,
  CITY            VARCHAR2(85)  NOT NULL,
  STATE           VARCHAR2(2)   NOT NULL,
  POSTAL_CODE     VARCHAR2(10)  NOT NULL,
  COUNTRY         VARCHAR2(50)  NOT NULL,


  CONSTRAINT person_pk PRIMARY KEY (PERSON_ID),


  CONSTRAINT person_id_format_check
    CHECK (REGEXP_LIKE(PERSON_ID, '^[0-9]{5}-[0-9]{7}-[0-9]$'))
);
```

Customer:

The CUSTOMER table is a **subtype** of the PERSON table and is used to store additional information specific to customers.

Key Features:

1. **Primary Key**:
   - CUSTOMER_ID uniquely identifies each customer and ensures no duplicates exist.
   - CUSTOMER_ID also acts as a foreign key, referencing the PERSON table's PERSON_ID, establishing an **inheritance relationship**.

2. **Attributes**:
   - ACCOUNT_CREATION_DATE: Stores the date when the customer's account was created. This helps track customer registration timelines.
3. **Constraints**:
   - customer_pk: Ensures CUSTOMER_ID is unique and non-null.
   - fk_customer_person: Establishes a foreign key relationship with the PERSON table, ensuring that every CUSTOMER_ID exists in the PERSON table. This maintains data integrity and avoids orphan records.

Use Case:

The CUSTOMER table adds a customer-specific layer on top of general person data, enabling efficient tracking of customers without duplicating common attributes like name, contact details, and address.

```
CREATE TABLE CUSTOMER(

 CUSTOMER_ID            VARCHAR2(15)  NOT NULL,

 ACCOUNT_CREATION_DATE    DATE,



 CONSTRAINT customer_pk PRIMARY KEY (CUSTOMER_ID),



 CONSTRAINT fk_customer_person FOREIGN KEY (CUSTOMER_ID)

      REFERENCES PERSON(PERSON_ID)

);
```

Employee:

The EMPLOYEE table is a **subtype** of the PERSON table and is designed to store information specific to employees within the system.

Key Features:

1. **Primary Key**:
   - EMPLOYEE_ID uniquely identifies each employee and ensures no duplicates exist.
   - EMPLOYEE_ID also acts as a **foreign key**, referencing the PERSON table's PERSON_ID, establishing an inheritance relationship between PERSON and EMPLOYEE.
2. **Attributes**:
   - ROLE: Describes the employee's position or job role within the organization (e.g., Manager, Support Staff).
   - SALARY: Records the employee's salary in numeric format with up to two decimal places.
   - STATUS: Specifies the current status of the employee, such as "Active" or "Inactive."
   - JOINING_DATE: Captures the date the employee joined the organization.
3. **Constraints**:
   - employee_pk: Ensures that EMPLOYEE_ID is unique and cannot be null.
   - fk_employee_person: Establishes a **foreign key relationship** with the PERSON table, ensuring all employees are registered as persons first, maintaining data consistency.

Use Case:

The EMPLOYEE table allows the system to manage employee-specific details like job roles, salaries, and statuses while reusing general attributes (e.g., names, contact information) from the PERSON table, promoting a normalized database design.

```
CREATE TABLE EMPLOYEE(
  EMPLOYEE_ID      VARCHAR2(15)  NOT NULL,
  ROLE             VARCHAR2(50)  NOT NULL,
  SALARY           NUMBER(8,2)   NOT NULL,
  STATUS           VARCHAR2(20)  NOT NULL,
  JOINING_DATE     DATE          NOT NULL,


  CONSTRAINT employee_pk PRIMARY KEY (EMPLOYEE_ID),
  CONSTRAINT fk_employee_person FOREIGN KEY (EMPLOYEE_ID)
       REFERENCES PERSON(PERSON_ID)
);
```

Artist:

The ARTIST table is designed to store specific information about artists within the Art Vault Management System, linking each artist to their unique attributes and style.

Key Features:

1. **Primary Key:**
   - **ARTIST_ID (VARCHAR2)**: Uniquely identifies each artist in the system. This ensures that no two artists have the same identifier and that each artist is distinguishable.
2. **Attributes:**
   - **STYLE (VARCHAR2)**: Describes the artistic style of the artist, such as Abstract, Renaissance, or Modern. This attribute provides context about the artist's preferred artistic approach or genre.
3. **Constraints:**
   - **artist_pk**: Ensures the uniqueness and non-nullability of the ARTIST_ID, maintaining the integrity of the table.
   - **fk_artist_person**: Establishes a foreign key relationship between ARTIST_ID and PERSON_ID in the PERSON table. This ensures that every artist is also represented as a person in the system, linking the artist's data with their general personal information (e.g., name, contact details).

Use Case:

The ARTIST table is used to manage and store specific details about each artist within the system. By associating an artist with a particular style, the system can organize and categorize artists according to their artistic approaches. The foreign key relationship with the PERSON table ensures that general information about the artist is stored centrally and avoids redundancy.

```
CREATE TABLE ARTIST(

 ARTIST_ID       VARCHAR2(15)  NOT NULL,

 STYLE          VARCHAR2(100) NOT NULL,


 CONSTRAINT artist_pk PRIMARY KEY (ARTIST_ID),


 CONSTRAINT fk_artist_person FOREIGN KEY (ARTIST_ID)

    REFERENCES PERSON(PERSON_ID)

);
```

Order:

The ORDER_T table is designed to store information about customer orders within the Art Vault Management System. It captures the details of each order placed by a customer, along with the relevant status and payment information.

Key Features:

1. **Primary Key:**
    - **ORDER_ID (VARCHAR2)**: Uniquely identifies each order in the system. This ensures that each order can be distinctly referenced and no duplicates exist.
2. **Attributes:**
    - **CUSTOMER_ID (VARCHAR2)**: Refers to the customer who placed the order. This is a foreign key that links to the CUSTOMER table, ensuring each order is associated with a valid customer.
    - **RECIPIENT (VARCHAR2)**: Specifies the recipient's name for the order, which may differ from the customer's name if the order is a gift or being delivered to someone else.
    - **ORDER_DATE (DATE)**: Records the date when the order was placed. This helps track the timing and manage order history.
    - **ORDER_STATUS (VARCHAR2)**: Describes the current status of the order, such as "Pending," "Shipped," "Delivered," or "Cancelled." This helps in managing the order lifecycle.
    - **PAYMENT_METHOD (VARCHAR2)**: Indicates the method of payment used for the order (e.g., Credit Card, PayPal, Bank Transfer). This helps track how the transaction was processed.
3. **Constraints:**
    - **order_pk**: Ensures the uniqueness and non-nullability of the ORDER_ID, maintaining the integrity of the table and making each order identifiable.
    - **fk_order_customer**: Establishes a foreign key relationship with the CUSTOMER table, ensuring that every order is associated with a valid customer.

Use Case:

The ORDER_T table is used to manage customer orders, ensuring that each order is linked to a customer and contains relevant details such as recipient information, order status, and payment method. The foreign key relationship with the CUSTOMER table ensures data integrity by enforcing that each order is associated with an existing customer in the system. This structure helps in tracking and managing orders efficiently throughout the order lifecycle.

```
CREATE TABLE ORDER_T(
 ORDER_ID         VARCHAR2(15)  NOT NULL,
 CUSTOMER_ID      VARCHAR2(15)  NOT NULL,
 RECIEPIENT       VARCHAR2(50)  NOT NULL,
 ORDER_DATE       DATE          NOT NULL,
 ORDER_STATUS     VARCHAR2(20)  NOT NULL,
 PAYMENT_METHOD   VARCHAR2(20)  NOT NULL,

 CONSTRAINT order_pk PRIMARY KEY (ORDER_ID),

 CONSTRAINT fk_order_customer FOREIGN KEY (CUSTOMER_ID)
   REFERENCES CUSTOMER(CUSTOMER_ID)
);
```

Artwork:

The ARTWORK table is designed to store detailed information about each artwork available in the Art Vault Management System. It captures essential attributes of the artwork, including its identity, category, status, and associated artist.

Key Features:

1. **Primary Key:**
   - **ARTWORK_ID (VARCHAR2)**: Uniquely identifies each artwork in the system. It ensures that each artwork can be referenced without ambiguity.
2. **Attributes:**
   - **TITLE (VARCHAR2)**: Specifies the title of the artwork, providing a descriptive name for the piece.
   - **CATEGORY (VARCHAR2)**: Defines the category or type of the artwork (e.g., Painting, Sculpture, Photography), helping to organize and classify the collection.
   - **ARTIST_ID (VARCHAR2)**: Refers to the artist who created the artwork. This is a foreign key that links to the ARTIST table, ensuring that each artwork is associated with a valid artist.
   - **STOCKED_DATE (DATE)**: Captures the date when the artwork was added to the collection. This helps track the timeline of the artwork's inclusion in the gallery.
   - **STATUS (VARCHAR2)**: Describes the current status of the artwork, such as "Available," "Sold," "Reserved," or "Exhibited." This assists in managing inventory and artwork lifecycle.
3. **Constraints:**

  - **artwork_pk**: Ensures the uniqueness and non-nullability of the ARTWORK_ID, making it a reliable identifier for each piece of artwork.
  - **fk_artwork_artist**: Establishes a foreign key relationship with the ARTIST table, ensuring that every artwork is associated with an existing artist, promoting data consistency.

Use Case:

The ARTWORK table helps in managing details about artworks in the collection, linking them to artists and maintaining key information such as titles, categories, status, and the date they were added. The foreign key relationship with the ARTIST table ensures each artwork has a valid artist, and the table's attributes enable efficient tracking, categorization, and status management of the artworks in the system.

```
CREATE TABLE ARTWORK (
   ARTWORK_ID     VARCHAR2(15)    NOT NULL,
   TITLE          VARCHAR2(100)   NOT NULL,
   CATEGORY       VARCHAR2(50)    NOT NULL,
   ARTIST_ID      VARCHAR2(15)    NOT NULL,
   STOCKED_DATE   DATE            NOT NULL,
   STATUS         VARCHAR2(20)    NOT NULL,

   CONSTRAINT artwork_pk PRIMARY KEY (ARTWORK_ID),

   CONSTRAINT fk_artwork_artist FOREIGN KEY (ARTIST_ID) REFERENCES ARTIST (ARTIST_ID)
);
```

OrderLine:

The ORDERLINE table is designed to capture the relationship between orders and the specific artworks included in those orders. It supports the many-to-many relationship between orders and artworks, allowing multiple artworks to be part of a single order, and an artwork to appear in multiple orders.

Key Features:

1. **Primary Key:**
   - **ORDER_ID (VARCHAR2)**: Part of the composite primary key that links the order line to a specific order in the ORDER_T table.
   - **ARTWORK_ID (VARCHAR2)**: Part of the composite primary key that links the order line to a specific artwork in the ARTWORK table.
   - The combination of ORDER_ID and ARTWORK_ID uniquely identifies each entry in the ORDERLINE table, ensuring no duplicate records for the same artwork within an order.
2. **Foreign Keys:**

   - **fk_orderline_artwork**: Establishes a foreign key relationship with the ARTWORK table, ensuring that every artwork in the order line corresponds to an existing artwork in the system.
   - **fk_orderline_order**: Establishes a foreign key relationship with the ORDER_T table, ensuring that every order line is linked to a valid order in the system.
3. **Constraints:**
   - **orderline_pk**: Ensures that the combination of ORDER_ID and ARTWORK_ID is unique within the table, preventing duplicate entries of the same artwork in an order.

Use Case:

The ORDERLINE table enables the tracking of which artworks are included in each order. By linking the ORDER_T and ARTWORK tables through foreign keys, this table supports the many-to-many relationship between orders and artworks. It ensures data integrity by enforcing that each artwork in an order is valid, and each order line is associated with a legitimate order. This structure allows efficient management and tracking of ordered artworks.

```
CREATE TABLE ORDERLINE (
   ORDER_ID       VARCHAR2(15)    NOT NULL,
   ARTWORK_ID     VARCHAR2(15)    NOT NULL,

   CONSTRAINT fk_orderline_artwork FOREIGN KEY (ARTWORK_ID) REFERENCES ARTWORK(ARTWORK_ID),

   CONSTRAINT fk_orderline_order FOREIGN KEY (ORDER_ID) REFERENCES ORDER_T(ORDER_ID),

   CONSTRAINT orderline_pk PRIMARY KEY (ORDER_ID, ARTWORK_ID)
);
```

Artwork_Price:

The Artwork_Price table is designed to track the price history of artworks, capturing changes in the price over time. It allows for the management of dynamic pricing, with each price change associated with an effective date to maintain historical pricing information.

Key Features:

1. **Primary Key:**
   - **ARTWORK_ID (VARCHAR2)**: Identifies the artwork whose price is being recorded. Combined with the EFFECTIVE_DATE, this forms a composite primary key.
   - **EFFECTIVE_DATE (DATE)**: Marks the date when the price became effective. The combination of ARTWORK_ID and EFFECTIVE_DATE uniquely identifies each price record

for an artwork, ensuring that multiple price entries for the same artwork can exist with different effective dates.

2. **Foreign Key:**
    - **fk_artwork**: Establishes a foreign key relationship with the ARTWORK table, ensuring that each price record is associated with a valid artwork.
3. **Constraints:**
    - **artwork_price_pk**: Ensures the uniqueness of price records by enforcing the composite primary key, where the same artwork can have multiple price records, but each record must have a distinct effective date.

Use Case:

The Artwork_Price table allows the system to track price changes for artworks over time. Each price is associated with a specific effective date, making it possible to maintain a historical record of artwork prices. This structure supports features like price updates, querying historical prices, and managing price changes for artworks based on time-specific conditions.

```
CREATE TABLE Artwork_Price (

   ARTWORK_ID         VARCHAR2(15)    NOT NULL,

   PRICE              DECIMAL(10, 2)  NOT NULL,

   EFFECTIVE_DATE     DATE            NOT NULL,



   CONSTRAINT fk_artwork FOREIGN KEY (ARTWORK_ID) REFERENCES Artwork(ARTWORK_ID),



   CONSTRAINT artwork_price_pk PRIMARY KEY (ARTWORK_ID, EFFECTIVE_DATE)

);
```

Event:

The **EVENT** table is designed to store details about various events hosted in the system. It serves as a central record for each event, capturing information such as the event's name, start and end dates, and location.

Key Features:

1. **Primary Key**:
    - EVENT_NAME and START_DATE together form the composite primary key, ensuring the uniqueness of each event by both name and start date.
2. **Attributes**:
    - EVENT_NAME: A string that uniquely identifies the name of the event (e.g., "Spring Art Expo").
    - START_DATE: The date the event begins.
    - END_DATE: The date the event concludes.
    - EVENT_LOCATION: The location where the event is hosted (e.g., "New York Gallery").
3. **Constraints**:

    - event_pk: Ensures that the combination of EVENT_NAME and START_DATE is unique, preventing duplicate events with the same name and start date.

<u>Use Case</u>:

The **EVENT** table is used to store and manage details about different events, such as art exhibits, fairs, and shows, with specific start and end dates and locations.

```
CREATE TABLE EVENT (

  EVENT_NAME      VARCHAR2(50)   NOT NULL,

  START_DATE      DATE           NOT NULL,

  END_DATE        DATE           NOT NULL,

  EVENT_LOCATION  VARCHAR2(100)  NOT NULL,



  CONSTRAINT event_pk PRIMARY KEY (EVENT_NAME, START_DATE)

);
```

<u>Ticket:</u>

The **TICKET** table is designed to store ticket-related information for the events in the system. Each ticket is uniquely identified by its TICKET_ID and contains details about the event, ticket price, and the effective date when the ticket price is applicable.

<u>Key Features</u>:

1. **Primary Key**:
    - TICKET_ID uniquely identifies each ticket and ensures there are no duplicate tickets in the system.
2. **Attributes**:
    - TICKET_ID: A unique identifier for the ticket (e.g., "TICKET-001").
    - EVENT_NAME: The name of the event for which the ticket is issued (links to the EVENT table).
    - START_DATE: The start date of the event for which the ticket is issued (links to the EVENT table).
    - TICKET_PRICE: The price of the ticket (e.g., 50.00).
    - EFFECTIVE_DATE: The date when the ticket price becomes effective.
3. **Constraints**:
    - ticket_pk: Ensures that the TICKET_ID is unique.
    - fk_ticket_event: Establishes a foreign key relationship with the EVENT table. The combination of EVENT_NAME and START_DATE in the **TICKET** table must correspond to an existing event in the **EVENT** table, ensuring referential integrity between the two tables.

<u>Use Case</u>: The **TICKET** table allows the system to track and manage ticket prices for events, including handling price changes over time and associating each ticket with a specific event and its details.

```
CREATE TABLE TICKET (
  TICKET_ID       VARCHAR2(15)  NOT NULL,
  EVENT_NAME      VARCHAR2(50)  NOT NULL,
  START_DATE      DATE          NOT NULL,
  TICKET_PRICE    NUMBER(5)     NOT NULL,
  EFFECTIVE_DATE  DATE          NOT NULL,


  CONSTRAINT ticket_pk PRIMARY KEY (TICKET_ID),


  CONSTRAINT fk_ticket_event FOREIGN KEY (EVENT_NAME, START_DATE)
      REFERENCES EVENT (EVENT_NAME, START_DATE)   -- Composite foreign key on EVENT_NAME
and START_DATE
);
```

Event_Employee:

The **EVENT_EMPLOYEE** table represents the many-to-many relationship between events and employees. It stores details about the employees assigned to specific events, including their roles and working hours.

Key Features:

1. **Primary Key**:
   - EVENT_NAME, EMPLOYEE_ID, and START_DATE together form a composite primary key, ensuring uniqueness for each combination of event, employee, and date.
2. **Attributes**:
   - EVENT_NAME: The name of the event to which the employee is assigned (links to the **EVENT** table).
   - START_DATE: The start date of the event (links to the **EVENT** table).
   - EMPLOYEE_ID: The unique identifier for the employee (links to the **EMPLOYEE** table).
   - WORK_HOURS: The number of hours the employee works at the event.
   - EVENT_ROLE: The role or position the employee holds during the event (e.g., "Manager," "Coordinator").
3. **Constraints**:
   - event_employee_pk: Ensures that the combination of EVENT_NAME, EMPLOYEE_ID, and START_DATE is unique, preventing duplicate assignments.
   - fk_ticket_event1: Establishes a foreign key relationship with the **EVENT** table, ensuring that each event assignment corresponds to an existing event.
   - fk_ticket_event2: Establishes a foreign key relationship with the **EMPLOYEE** table, ensuring that the employee assigned to the event exists in the system.

Use Case: The **EVENT_EMPLOYEE** table is used to manage and track employees working at various events, including their roles and the hours they contribute. It ensures that each employee is linked to specific events and maintains data integrity between the **EVENT** and **EMPLOYEE** tables.

```
CREATE TABLE EVENT_EMPLOYEE (
  EVENT_NAME     VARCHAR2(50) NOT NULL,
  START_DATE     DATE         NOT NULL,
  EMPLOYEE_ID    VARCHAR2(15) NOT NULL,
  WORK_HOURS     NUMBER(2)    NOT NULL,
  EVENT_ROLE     VARCHAR2(50) NOT NULL,


  CONSTRAINT event_employee_pk PRIMARY KEY (EVENT_NAME, EMPLOYEE_ID, START_DATE),


  CONSTRAINT fk_ticket_event1 FOREIGN KEY (EVENT_NAME, START_DATE) REFERENCES
EVENT(EVENT_NAME, START_DATE),


   CONSTRAINT fk_ticket_event2 FOREIGN KEY (EMPLOYEE_ID) REFERENCES
EMPLOYEE(EMPLOYEE_ID)
);
```

Event_Artwork:

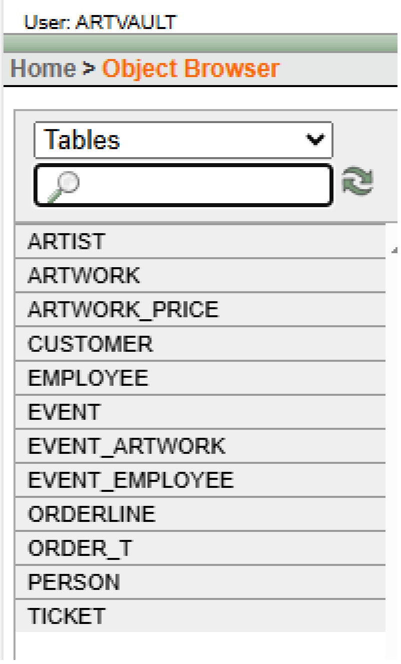
The **EVENT_ARTWORK** table represents the many-to-many relationship between events and artworks. It stores details about the artworks featured in specific events.

Key Features:

1. **Primary Key**:
   - EVENT_NAME, ARTWORK_ID, and START_DATE together form a composite primary key, ensuring uniqueness for each combination of event, artwork, and date.
2. **Attributes**:
   - EVENT_NAME: The name of the event where the artwork is featured (links to the **EVENT** table).
   - START_DATE: The start date of the event (links to the **EVENT** table).
   - ARTWORK_ID: The unique identifier of the artwork featured in the event (links to the **ARTWORK** table).
3. **Constraints**:
   - event_artwork_pk: Ensures that the combination of EVENT_NAME, ARTWORK_ID, and START_DATE is unique, preventing duplicate artwork assignments to events.
   - fk_event_artwork1: Establishes a foreign key relationship with the **EVENT** table, ensuring that each artwork assignment corresponds to an existing event.
   - fk_event_artwork2: Establishes a foreign key relationship with the **ARTWORK** table, ensuring that the artwork assigned to the event exists in the system.

<u>Use Case</u>: The **EVENT_ARTWORK** table is used to manage and track which artworks are featured in which events. It maintains a relationship between the **EVENT** and **ARTWORK** tables, ensuring that artwork details are properly associated with specific events.

```
CREATE TABLE EVENT_ARTWORK (

  EVENT_NAME      VARCHAR2(50)      NOT NULL,

  START_DATE      DATE              NOT NULL,

  ARTWORK_ID      VARCHAR2(15)      NOT NULL,


  CONSTRAINT event_artwork_pk PRIMARY KEY (EVENT_NAME, ARTWORK_ID, START_DATE),


  CONSTRAINT fk_event_artwork1 FOREIGN KEY (EVENT_NAME, START_DATE) REFERENCES
Event(EVENT_NAME, START_DATE),


  CONSTRAINT fk_event_artwork2 FOREIGN KEY (ARTWORK_ID) REFERENCES
ARTWORK(ARTWORK_ID)

);
```



ARTIST
ARTWORK
ARTWORK_PRICE
CUSTOMER
EMPLOYEE
EVENT
EVENT_ARTWORK
EVENT_EMPLOYEE
ORDERLINE
ORDER_T
PERSON
TICKET

<u>Insertion:</u>

<u>Person:</u>

```
INSERT INTO PERSON (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NO, GENDER,
CITY, STATE, POSTAL_CODE, COUNTRY)
```

VALUES ('12345-1234567-1', 'John', 'Doe', 'johndoe@gmail.com', '1234567890', 'M', 'New York', 'NY', '10001', 'USA');

INSERT INTO PERSON VALUES ('12345-1234567-2', 'Jane', 'Smith', 'janesmith@yahoo.com', '1234567891', 'F', 'Los Angeles', 'CA', '90001', 'USA');

INSERT INTO PERSON VALUES ('12345-1234567-3', 'Mark', 'Brown', 'markbrown@mail.com', '1234567892', 'M', 'Chicago', 'IL', '60601', 'USA');

INSERT INTO PERSON VALUES ('12345-1234567-4', 'Sara', 'Johnson', 'sarajohnson@outlook.com', '1234567893', 'F', 'Houston', 'TX', '77001', 'USA');

INSERT INTO PERSON VALUES ('12345-1234567-5', 'Mike', 'Wilson', 'mikewilson@company.com', '1234567894', 'M', 'San Francisco', 'CA', '94016', 'USA');

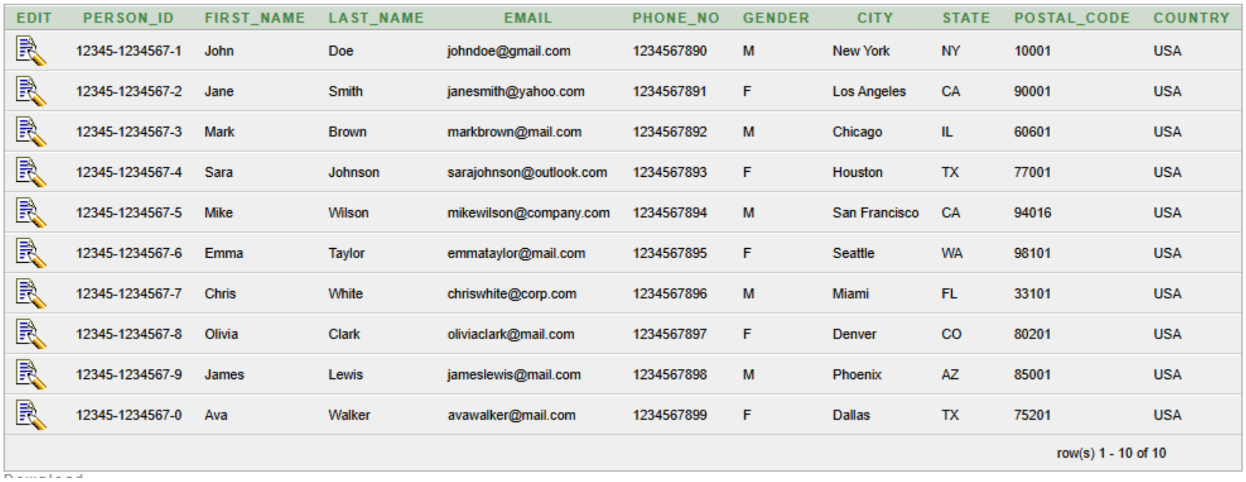
INSERT INTO PERSON VALUES ('12345-1234567-6', 'Emma', 'Taylor', 'emmataylor@mail.com', '1234567895', 'F', 'Seattle', 'WA', '98101', 'USA');

INSERT INTO PERSON VALUES ('12345-1234567-7', 'Chris', 'White', 'chriswhite@corp.com', '1234567896', 'M', 'Miami', 'FL', '33101', 'USA');

INSERT INTO PERSON VALUES ('12345-1234567-8', 'Olivia', 'Clark', 'oliviaclark@mail.com', '1234567897', 'F', 'Denver', 'CO', '80201', 'USA');

INSERT INTO PERSON VALUES ('12345-1234567-9', 'James', 'Lewis', 'jameslewis@mail.com', '1234567898', 'M', 'Phoenix', 'AZ', '85001', 'USA');

INSERT INTO PERSON VALUES ('12345-1234567-0', 'Ava', 'Walker', 'avawalker@mail.com', '1234567899', 'F', 'Dallas', 'TX', '75201', 'USA');



EDIT	PERSON_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NO	GENDER	CITY	STATE	POSTAL_CODE	COUNTRY
✎	12345-1234567-1	John	Doe	johndoe@gmail.com	1234567890	M	New York	NY	10001	USA
✎	12345-1234567-2	Jane	Smith	janesmith@yahoo.com	1234567891	F	Los Angeles	CA	90001	USA
✎	12345-1234567-3	Mark	Brown	markbrown@mail.com	1234567892	M	Chicago	IL	60601	USA
✎	12345-1234567-4	Sara	Johnson	sarajohnson@outlook.com	1234567893	F	Houston	TX	77001	USA
✎	12345-1234567-5	Mike	Wilson	mikewilson@company.com	1234567894	M	San Francisco	CA	94016	USA
✎	12345-1234567-6	Emma	Taylor	emmataylor@mail.com	1234567895	F	Seattle	WA	98101	USA
✎	12345-1234567-7	Chris	White	chriswhite@corp.com	1234567896	M	Miami	FL	33101	USA
✎	12345-1234567-8	Olivia	Clark	oliviaclark@mail.com	1234567897	F	Denver	CO	80201	USA
✎	12345-1234567-9	James	Lewis	jameslewis@mail.com	1234567898	M	Phoenix	AZ	85001	USA
✎	12345-1234567-0	Ava	Walker	avawalker@mail.com	1234567899	F	Dallas	TX	75201	USA

row(s) 1 - 10 of 10

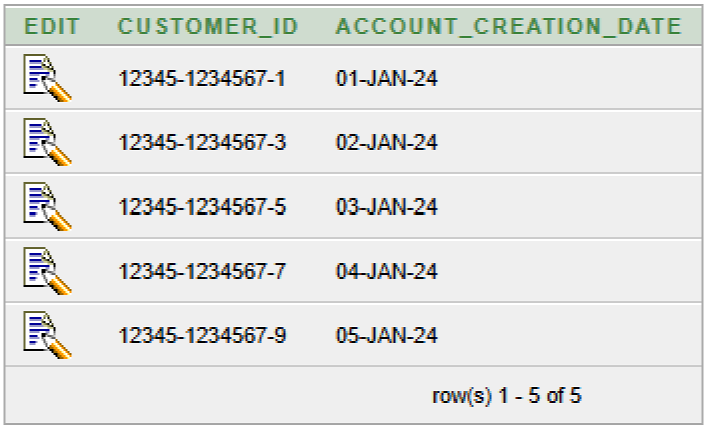
Customer:

INSERT INTO CUSTOMER (CUSTOMER_ID, ACCOUNT_CREATION_DATE)

VALUES ('12345-1234567-1', TO_DATE('2024-01-01', 'YYYY-MM-DD'));

INSERT INTO CUSTOMER VALUES ('12345-1234567-3', TO_DATE('2024-01-02', 'YYYY-MM-DD'));

```
INSERT INTO CUSTOMER VALUES ('12345-1234567-5', TO_DATE('2024-01-03', 'YYYY-MM-DD'));

INSERT INTO CUSTOMER VALUES ('12345-1234567-7', TO_DATE('2024-01-04', 'YYYY-MM-DD'));

INSERT INTO CUSTOMER VALUES ('12345-1234567-9', TO_DATE('2024-01-05', 'YYYY-MM-DD'));
```



EDIT	CUSTOMER_ID	ACCOUNT_CREATION_DATE
✎	12345-1234567-1	01-JAN-24
✎	12345-1234567-3	02-JAN-24
✎	12345-1234567-5	03-JAN-24
✎	12345-1234567-7	04-JAN-24
✎	12345-1234567-9	05-JAN-24

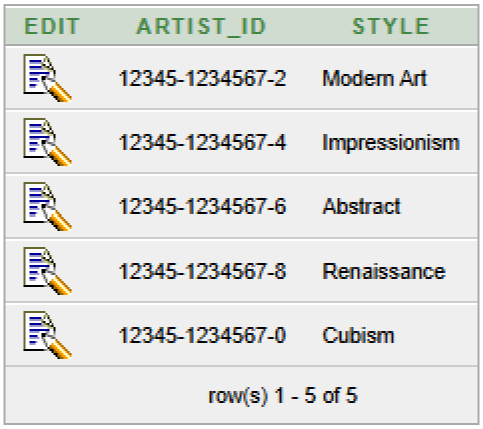
row(s) 1 - 5 of 5

Employee:

```
INSERT INTO EVENT (EVENT_NAME, EVENT_LOCATION, START_DATE, END_DATE)

VALUES ('Spring Art Expo', 'New York Gallery', TO_DATE('2024-03-01', 'YYYY-MM-DD'),
TO_DATE('2024-03-05', 'YYYY-MM-DD'));

INSERT INTO EVENT (EVENT_NAME, EVENT_LOCATION, START_DATE, END_DATE)

VALUES ('Summer Sculpture Show', 'Chicago Art Center', TO_DATE('2024-06-10', 'YYYY-MM-DD'),
TO_DATE('2024-06-15', 'YYYY-MM-DD'));

INSERT INTO EVENT (EVENT_NAME, EVENT_LOCATION, START_DATE, END_DATE)

VALUES ('Modern Art Fair', 'Los Angeles Museum', TO_DATE('2024-04-15', 'YYYY-MM-DD'),
TO_DATE('2024-04-20', 'YYYY-MM-DD'));

INSERT INTO EVENT (EVENT_NAME, EVENT_LOCATION, START_DATE, END_DATE)

VALUES ('Abstract Showcase', 'Houston Art Space', TO_DATE('2024-05-01', 'YYYY-MM-DD'),
TO_DATE('2024-05-05', 'YYYY-MM-DD'));

INSERT INTO EVENT (EVENT_NAME, EVENT_LOCATION, START_DATE, END_DATE)

VALUES ('Classic Art Week', 'San Francisco Hall', TO_DATE('2024-07-01', 'YYYY-MM-DD'),
TO_DATE('2024-07-07', 'YYYY-MM-DD'));
```

Artist:

```
INSERT INTO ARTIST (ARTIST_ID, STYLE)

VALUES ('12345-1234567-2', 'Modern Art');

INSERT INTO ARTIST VALUES ('12345-1234567-4', 'Impressionism');
INSERT INTO ARTIST VALUES ('12345-1234567-6', 'Abstract');

INSERT INTO ARTIST VALUES ('12345-1234567-8', 'Renaissance');

INSERT INTO ARTIST VALUES ('12345-1234567-0', 'Cubism');
```

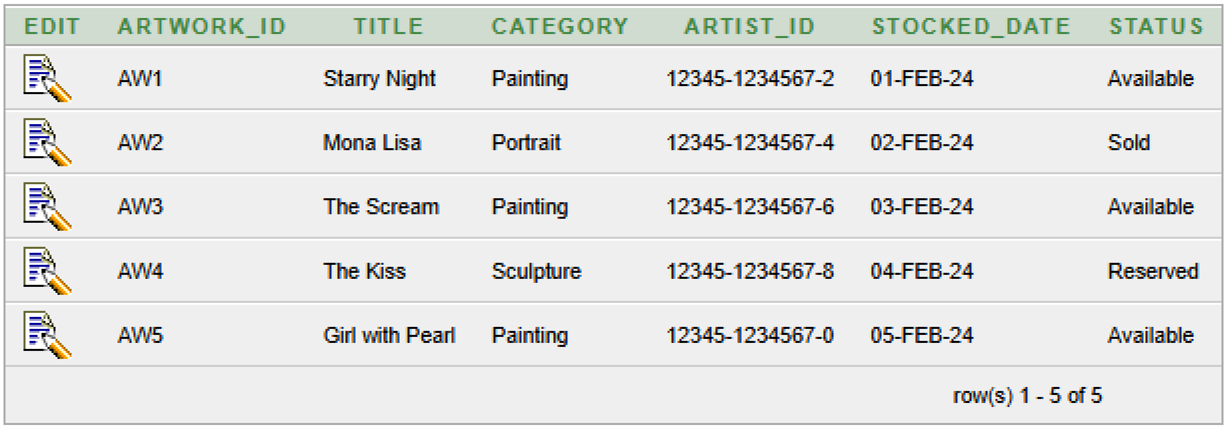


EDIT	ARTIST_ID	STYLE
✎	12345-1234567-2	Modern Art
✎	12345-1234567-4	Impressionism
✎	12345-1234567-6	Abstract
✎	12345-1234567-8	Renaissance
✎	12345-1234567-0	Cubism

row(s) 1 - 5 of 5

Artwork:

```
INSERT INTO ARTWORK (ARTWORK_ID, TITLE, CATEGORY, ARTIST_ID, STOCKED_DATE, STATUS)

VALUES ('AW1', 'Starry Night', 'Painting', '12345-1234567-2', TO_DATE('2024-02-01', 'YYYY-MM-DD'), 'Available');

INSERT INTO ARTWORK VALUES ('AW2', 'Mona Lisa', 'Portrait', '12345-1234567-4', TO_DATE('2024-02-02', 'YYYY-MM-DD'), 'Sold');

INSERT INTO ARTWORK VALUES ('AW3', 'The Scream', 'Painting', '12345-1234567-6', TO_DATE('2024-02-03', 'YYYY-MM-DD'), 'Available');

INSERT INTO ARTWORK VALUES ('AW4', 'The Kiss', 'Sculpture', '12345-1234567-8', TO_DATE('2024-02-04', 'YYYY-MM-DD'), 'Reserved');

INSERT INTO ARTWORK VALUES ('AW5', 'Girl with Pearl', 'Painting', '12345-1234567-0', TO_DATE('2024-02-05', 'YYYY-MM-DD'), 'Available');
```

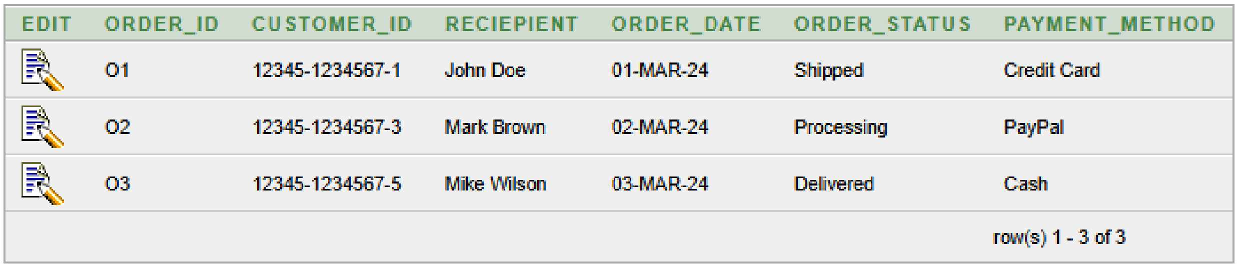


EDIT	ARTWORK_ID	TITLE	CATEGORY	ARTIST_ID	STOCKED_DATE	STATUS
✎	AW1	Starry Night	Painting	12345-1234567-2	01-FEB-24	Available
✎	AW2	Mona Lisa	Portrait	12345-1234567-4	02-FEB-24	Sold
✎	AW3	The Scream	Painting	12345-1234567-6	03-FEB-24	Available
✎	AW4	The Kiss	Sculpture	12345-1234567-8	04-FEB-24	Reserved
✎	AW5	Girl with Pearl	Painting	12345-1234567-0	05-FEB-24	Available

row(s) 1 - 5 of 5

Order:

```
INSERT INTO ORDER_T (ORDER_ID, CUSTOMER_ID, RECIEPIENT, ORDER_DATE, ORDER_STATUS, PAYMENT_METHOD)

VALUES ('O1', '12345-1234567-1', 'John Doe', TO_DATE('2024-03-01', 'YYYY-MM-DD'), 'Shipped', 'Credit Card');

INSERT INTO ORDER_T VALUES ('O2', '12345-1234567-3', 'Mark Brown', TO_DATE('2024-03-02', 'YYYY-MM-DD'), 'Processing', 'PayPal');

INSERT INTO ORDER_T VALUES ('O3', '12345-1234567-5', 'Mike Wilson', TO_DATE('2024-03-03', 'YYYY-MM-DD'), 'Delivered', 'Cash');
```

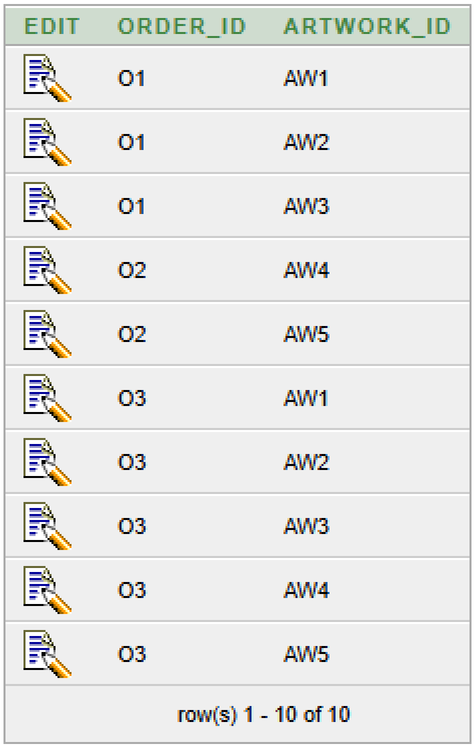


EDIT	ORDER_ID	CUSTOMER_ID	RECIEPIENT	ORDER_DATE	ORDER_STATUS	PAYMENT_METHOD
✎	O1	12345-1234567-1	John Doe	01-MAR-24	Shipped	Credit Card
✎	O2	12345-1234567-3	Mark Brown	02-MAR-24	Processing	PayPal
✎	O3	12345-1234567-5	Mike Wilson	03-MAR-24	Delivered	Cash

row(s) 1 - 3 of 3

Order Line:

```
INSERT INTO ORDERLINE (ORDER_ID, ARTWORK_ID)

VALUES ('O1', 'AW1');

INSERT INTO ORDERLINE VALUES ('O1', 'AW2');

INSERT INTO ORDERLINE VALUES ('O1', 'AW3');

INSERT INTO ORDERLINE VALUES ('O2', 'AW4');

INSERT INTO ORDERLINE VALUES ('O2', 'AW5');

INSERT INTO ORDERLINE VALUES ('O3', 'AW1');

INSERT INTO ORDERLINE VALUES ('O3', 'AW2');

INSERT INTO ORDERLINE VALUES ('O3', 'AW3');

INSERT INTO ORDERLINE VALUES ('O3', 'AW4');

INSERT INTO ORDERLINE VALUES ('O3', 'AW5');
```

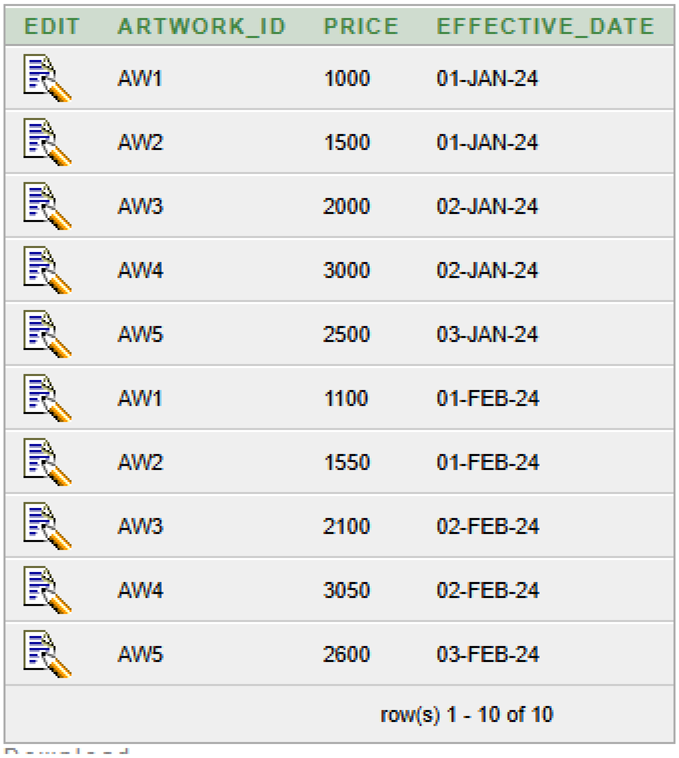


EDIT	ORDER_ID	ARTWORK_ID
✎	O1	AW1
✎	O1	AW2
✎	O1	AW3
✎	O2	AW4
✎	O2	AW5
✎	O3	AW1
✎	O3	AW2
✎	O3	AW3
✎	O3	AW4
✎	O3	AW5

row(s) 1 - 10 of 10

Artwork Price:

```
INSERT INTO ARTWORK_PRICE (ARTWORK_ID, PRICE, EFFECTIVE_DATE)
VALUES ('AW1', 1000.00, TO_DATE('2024-01-01', 'YYYY-MM-DD'));
INSERT INTO ARTWORK_PRICE VALUES ('AW2', 1500.00, TO_DATE('2024-01-01', 'YYYY-MM-DD'));
INSERT INTO ARTWORK_PRICE VALUES ('AW3', 2000.00, TO_DATE('2024-01-02', 'YYYY-MM-DD'));
INSERT INTO ARTWORK_PRICE VALUES ('AW4', 3000.00, TO_DATE('2024-01-02', 'YYYY-MM-DD'));
INSERT INTO ARTWORK_PRICE VALUES ('AW5', 2500.00, TO_DATE('2024-01-03', 'YYYY-MM-DD'));
INSERT INTO ARTWORK_PRICE VALUES ('AW1', 1100.00, TO_DATE('2024-02-01', 'YYYY-MM-DD'));
INSERT INTO ARTWORK_PRICE VALUES ('AW2', 1550.00, TO_DATE('2024-02-01', 'YYYY-MM-DD'));
INSERT INTO ARTWORK_PRICE VALUES ('AW3', 2100.00, TO_DATE('2024-02-02', 'YYYY-MM-DD'));
INSERT INTO ARTWORK_PRICE VALUES ('AW4', 3050.00, TO_DATE('2024-02-02', 'YYYY-MM-DD'));
INSERT INTO ARTWORK_PRICE VALUES ('AW5', 2600.00, TO_DATE('2024-02-03', 'YYYY-MM-DD'));
```



EDIT	ARTWORK_ID	PRICE	EFFECTIVE_DATE
✎	AW1	1000	01-JAN-24
✎	AW2	1500	01-JAN-24
✎	AW3	2000	02-JAN-24
✎	AW4	3000	02-JAN-24
✎	AW5	2500	03-JAN-24
✎	AW1	1100	01-FEB-24
✎	AW2	1550	01-FEB-24
✎	AW3	2100	02-FEB-24
✎	AW4	3050	02-FEB-24
✎	AW5	2600	03-FEB-24

row(s) 1 - 10 of 10

Event:

INSERT INTO EVENT (EVENT_NAME, START_DATE, END_DATE, EVENT_LOCATION, EMPLOYEE_ID)

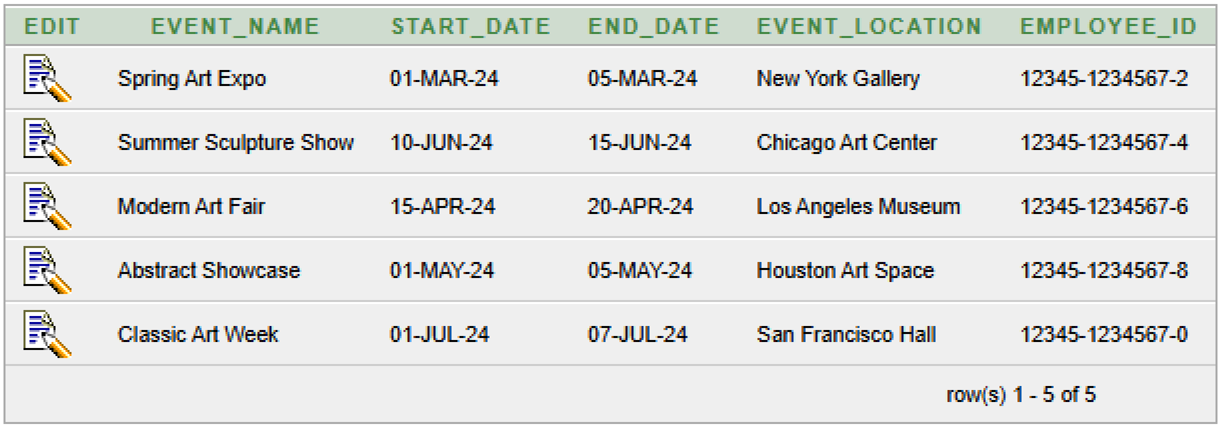
VALUES ('Spring Art Expo', TO_DATE('2024-03-01', 'YYYY-MM-DD'), TO_DATE('2024-03-05', 'YYYY-MM-DD'), 'New York Gallery', '12345-1234567-2');

INSERT INTO EVENT VALUES ('Summer Sculpture Show', TO_DATE('2024-06-10', 'YYYY-MM-DD'), TO_DATE('2024-06-15', 'YYYY-MM-DD'), 'Chicago Art Center', '12345-1234567-4');

INSERT INTO EVENT VALUES ('Modern Art Fair', TO_DATE('2024-04-15', 'YYYY-MM-DD'), TO_DATE('2024-04-20', 'YYYY-MM-DD'), 'Los Angeles Museum', '12345-1234567-6');

INSERT INTO EVENT VALUES ('Abstract Showcase', TO_DATE('2024-05-01', 'YYYY-MM-DD'), TO_DATE('2024-05-05', 'YYYY-MM-DD'), 'Houston Art Space', '12345-1234567-8');

INSERT INTO EVENT VALUES ('Classic Art Week', TO_DATE('2024-07-01', 'YYYY-MM-DD'), TO_DATE('2024-07-07', 'YYYY-MM-DD'), 'San Francisco Hall', '12345-1234567-0');



EDIT	EVENT_NAME	START_DATE	END_DATE	EVENT_LOCATION	EMPLOYEE_ID
✎	Spring Art Expo	01-MAR-24	05-MAR-24	New York Gallery	12345-1234567-2
✎	Summer Sculpture Show	10-JUN-24	15-JUN-24	Chicago Art Center	12345-1234567-4
✎	Modern Art Fair	15-APR-24	20-APR-24	Los Angeles Museum	12345-1234567-6
✎	Abstract Showcase	01-MAY-24	05-MAY-24	Houston Art Space	12345-1234567-8
✎	Classic Art Week	01-JUL-24	07-JUL-24	San Francisco Hall	12345-1234567-0

row(s) 1 - 5 of 5

Ticket:

INSERT INTO TICKET (TICKET_ID, EVENT_NAME, START_DATE, TICKET_PRICE, EFFECTIVE_DATE)

VALUES ('TICKET-001', 'Spring Art Expo', TO_DATE('2024-03-01', 'YYYY-MM-DD'), 50.00, TO_DATE('2024-03-01', 'YYYY-MM-DD'));

INSERT INTO TICKET (TICKET_ID, EVENT_NAME, START_DATE, TICKET_PRICE, EFFECTIVE_DATE)

VALUES ('TICKET-002', 'Summer Sculpture Show', TO_DATE('2024-06-10', 'YYYY-MM-DD'), 60.00, TO_DATE('2024-06-10', 'YYYY-MM-DD'));

INSERT INTO TICKET (TICKET_ID, EVENT_NAME, START_DATE, TICKET_PRICE, EFFECTIVE_DATE)

VALUES ('TICKET-003', 'Modern Art Fair', TO_DATE('2024-04-15', 'YYYY-MM-DD'), 55.00, TO_DATE('2024-04-15', 'YYYY-MM-DD'));

INSERT INTO TICKET (TICKET_ID, EVENT_NAME, START_DATE, TICKET_PRICE, EFFECTIVE_DATE)

VALUES ('TICKET-004', 'Abstract Showcase', TO_DATE('2024-05-01', 'YYYY-MM-DD'), 40.00, TO_DATE('2024-05-01', 'YYYY-MM-DD'));

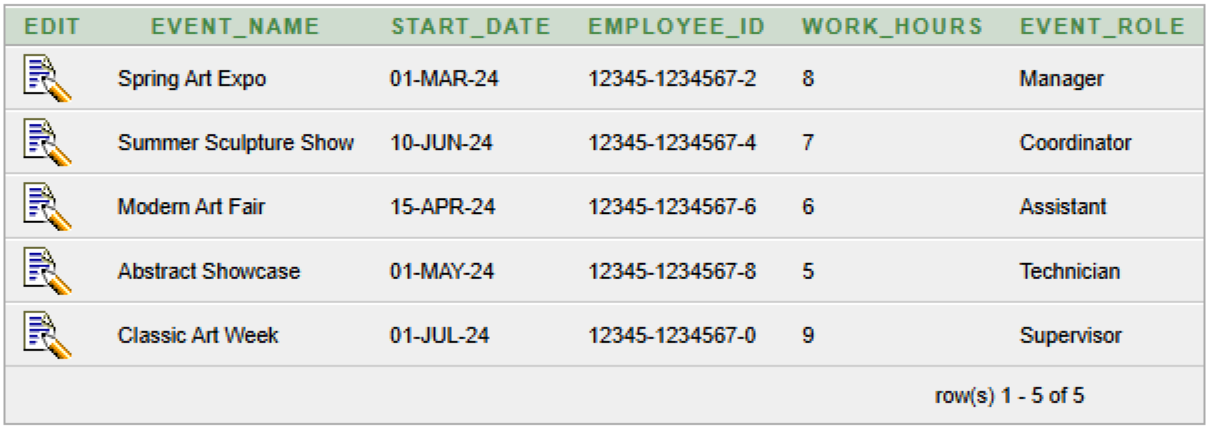
INSERT INTO TICKET (TICKET_ID, EVENT_NAME, START_DATE, TICKET_PRICE, EFFECTIVE_DATE)

VALUES ('TICKET-005', 'Classic Art Week', TO_DATE('2024-07-01', 'YYYY-MM-DD'), 70.00, TO_DATE('2024-07-01', 'YYYY-MM-DD'));

Event_Employee:

INSERT INTO EVENT_EMPLOYEE (EVENT_NAME, START_DATE, EMPLOYEE_ID, WORK_HOURS, EVENT_ROLE)

```
VALUES ('Spring Art Expo', TO_DATE('2024-03-01', 'YYYY-MM-DD'), '12345-1234567-2', 8, 'Manager');

INSERT INTO EVENT_EMPLOYEE VALUES ('Summer Sculpture Show', TO_DATE('2024-06-10', 'YYYY-MM-DD'), '12345-1234567-4', 7, 'Coordinator');

INSERT INTO EVENT_EMPLOYEE VALUES ('Modern Art Fair', TO_DATE('2024-04-15', 'YYYY-MM-DD'), '12345-1234567-6', 6, 'Assistant');

INSERT INTO EVENT_EMPLOYEE VALUES ('Abstract Showcase', TO_DATE('2024-05-01', 'YYYY-MM-DD'), '12345-1234567-8', 5, 'Technician');

INSERT INTO EVENT_EMPLOYEE VALUES ('Classic Art Week', TO_DATE('2024-07-01', 'YYYY-MM-DD'), '12345-1234567-0', 9, 'Supervisor');
```

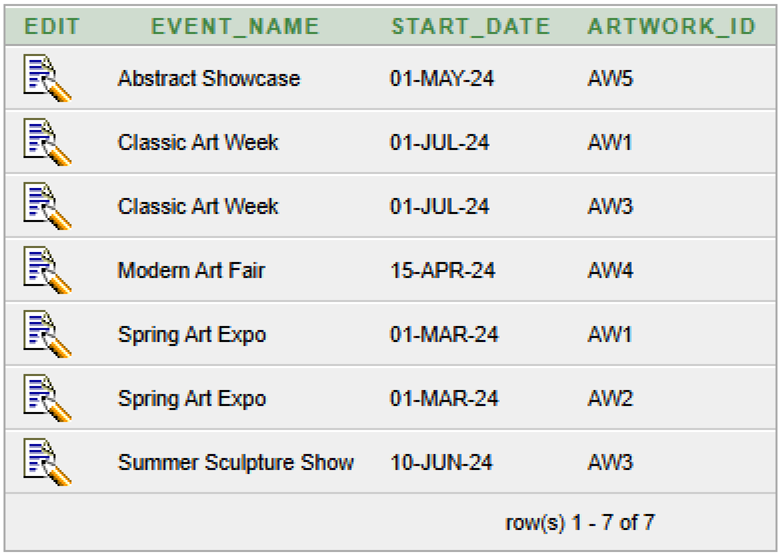


EDIT	EVENT_NAME	START_DATE	EMPLOYEE_ID	WORK_HOURS	EVENT_ROLE
✎	Spring Art Expo	01-MAR-24	12345-1234567-2	8	Manager
✎	Summer Sculpture Show	10-JUN-24	12345-1234567-4	7	Coordinator
✎	Modern Art Fair	15-APR-24	12345-1234567-6	6	Assistant
✎	Abstract Showcase	01-MAY-24	12345-1234567-8	5	Technician
✎	Classic Art Week	01-JUL-24	12345-1234567-0	9	Supervisor

row(s) 1 - 5 of 5

Event_Artwork:

```
INSERT INTO EVENT_ARTWORK (EVENT_NAME, START_DATE, ARTWORK_ID)

VALUES ('Spring Art Expo', TO_DATE('2024-03-01', 'YYYY-MM-DD'), 'AW1');

INSERT INTO EVENT_ARTWORK VALUES ('Spring Art Expo', TO_DATE('2024-03-01', 'YYYY-MM-DD'), 'AW2');

INSERT INTO EVENT_ARTWORK VALUES ('Summer Sculpture Show', TO_DATE('2024-06-10', 'YYYY-MM-DD'), 'AW3');

INSERT INTO EVENT_ARTWORK VALUES ('Modern Art Fair', TO_DATE('2024-04-15', 'YYYY-MM-DD'), 'AW4');

INSERT INTO EVENT_ARTWORK VALUES ('Abstract Showcase', TO_DATE('2024-05-01', 'YYYY-MM-DD'), 'AW5');

INSERT INTO EVENT_ARTWORK VALUES ('Classic Art Week', TO_DATE('2024-07-01', 'YYYY-MM-DD'), 'AW1');

INSERT INTO EVENT_ARTWORK VALUES ('Classic Art Week', TO_DATE('2024-07-01', 'YYYY-MM-DD'), 'AW3');
```



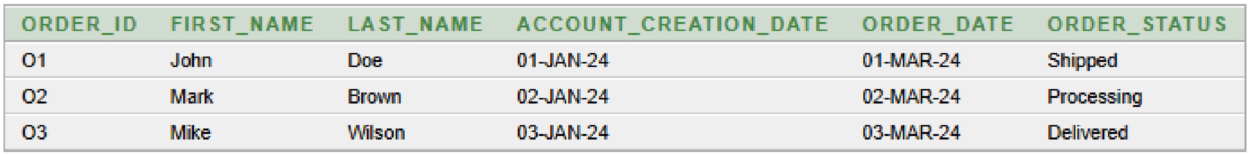
EDIT	EVENT_NAME	START_DATE	ARTWORK_ID
✎	Abstract Showcase	01-MAY-24	AW5
✎	Classic Art Week	01-JUL-24	AW1
✎	Classic Art Week	01-JUL-24	AW3
✎	Modern Art Fair	15-APR-24	AW4
✎	Spring Art Expo	01-MAR-24	AW1
✎	Spring Art Expo	01-MAR-24	AW2
✎	Summer Sculpture Show	10-JUN-24	AW3

row(s) 1 - 7 of 7

SQL Queries:

Retrieve all orders with customer details

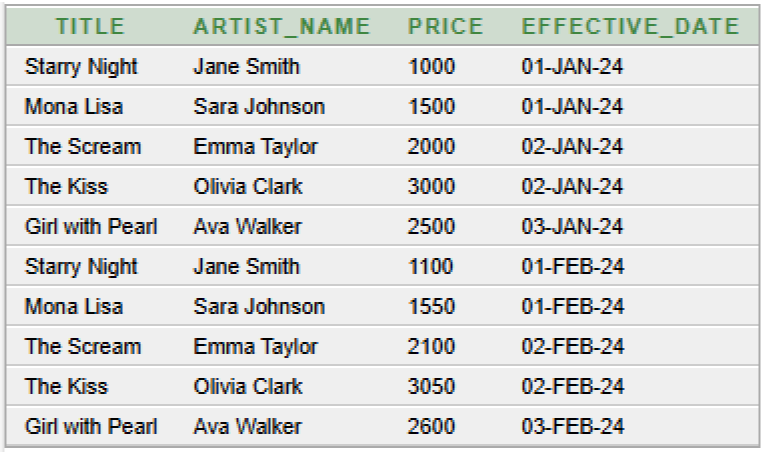
```
SELECT O.ORDER_ID, P.FIRST_NAME, P.LAST_NAME, C.ACCOUNT_CREATION_DATE, O.ORDER_DATE, O.ORDER_STATUS

FROM ORDER_T O

JOIN CUSTOMER C ON O.CUSTOMER_ID = C.CUSTOMER_ID

JOIN PERSON P ON C.CUSTOMER_ID = P.PERSON_ID;
```



ORDER_ID	FIRST_NAME	LAST_NAME	ACCOUNT_CREATION_DATE	ORDER_DATE	ORDER_STATUS
O1	John	Doe	01-JAN-24	01-MAR-24	Shipped
O2	Mark	Brown	02-JAN-24	02-MAR-24	Processing
O3	Mike	Wilson	03-JAN-24	03-MAR-24	Delivered

List all artwork titles, artists' names, and prices

```
SELECT AW.TITLE, P.FIRST_NAME || ' ' || P.LAST_NAME AS ARTIST_NAME, AP.PRICE, AP.EFFECTIVE_DATE

FROM ARTWORK AW

JOIN ARTIST A ON AW.ARTIST_ID = A.ARTIST_ID

JOIN PERSON P ON A.ARTIST_ID = P.PERSON_ID

JOIN ARTWORK_PRICE AP ON AW.ARTWORK_ID = AP.ARTWORK_ID;
```



TITLE	ARTIST_NAME	PRICE	EFFECTIVE_DATE
Starry Night	Jane Smith	1000	01-JAN-24
Mona Lisa	Sara Johnson	1500	01-JAN-24
The Scream	Emma Taylor	2000	02-JAN-24
The Kiss	Olivia Clark	3000	02-JAN-24
Girl with Pearl	Ava Walker	2500	03-JAN-24
Starry Night	Jane Smith	1100	01-FEB-24
Mona Lisa	Sara Johnson	1550	01-FEB-24
The Scream	Emma Taylor	2100	02-FEB-24
The Kiss	Olivia Clark	3050	02-FEB-24
Girl with Pearl	Ava Walker	2600	03-FEB-24

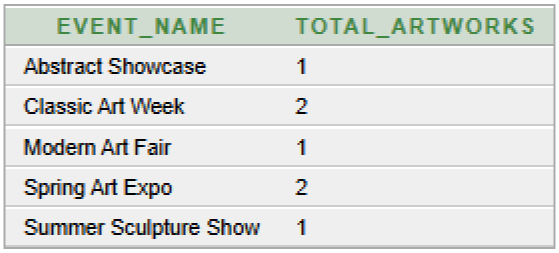
Get events and the total number of artworks associated with each event

SELECT E.EVENT_NAME, COUNT(EA.ARTWORK_ID) AS TOTAL_ARTWORKS

FROM EVENT E

LEFT JOIN EVENT_ARTWORK EA ON E.EVENT_NAME = EA.EVENT_NAME AND E.START_DATE = EA.START_DATE

GROUP BY E.EVENT_NAME;



EVENT_NAME	TOTAL_ARTWORKS
Abstract Showcase	1
Classic Art Week	2
Modern Art Fair	1
Spring Art Expo	2
Summer Sculpture Show	1

Retrieve the names of employees and their roles working in events

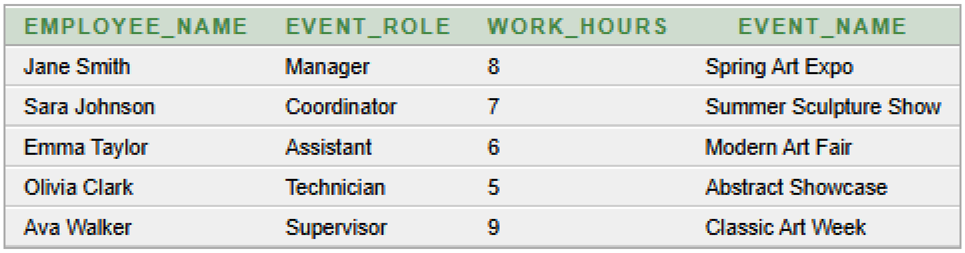
SELECT P.FIRST_NAME || ' ' || P.LAST_NAME AS EMPLOYEE_NAME, EE.EVENT_ROLE, EE.WORK_HOURS, E.EVENT_NAME

FROM EVENT_EMPLOYEE EE

JOIN EMPLOYEE EMP ON EE.EMPLOYEE_ID = EMP.EMPLOYEE_ID

JOIN PERSON P ON EMP.EMPLOYEE_ID = P.PERSON_ID

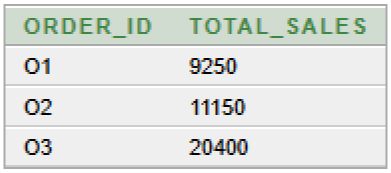
JOIN EVENT E ON EE.EVENT_NAME = E.EVENT_NAME AND EE.START_DATE = E.START_DATE;



EMPLOYEE_NAME	EVENT_ROLE	WORK_HOURS	EVENT_NAME
Jane Smith	Manager	8	Spring Art Expo
Sara Johnson	Coordinator	7	Summer Sculpture Show
Emma Taylor	Assistant	6	Modern Art Fair
Olivia Clark	Technician	5	Abstract Showcase
Ava Walker	Supervisor	9	Classic Art Week

Find the total sales amount for each order

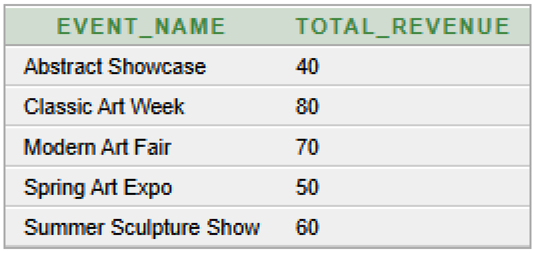
```
SELECT O.ORDER_ID, SUM(AP.PRICE) AS TOTAL_SALES

FROM ORDER_T O

JOIN ORDERLINE OL ON O.ORDER_ID = OL.ORDER_ID

JOIN ARTWORK_PRICE AP ON OL.ARTWORK_ID = AP.ARTWORK_ID

WHERE AP.EFFECTIVE_DATE <= O.ORDER_DATE

GROUP BY O.ORDER_ID

ORDER BY O.ORDER_ID;
```



ORDER_ID	TOTAL_SALES
O1	9250
O2	11150
O3	20400

List events along with their total ticket revenue

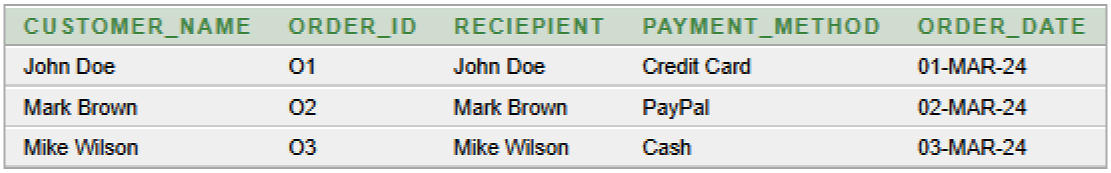
```
SELECT T.EVENT_NAME, SUM(T.TICKET_PRICE) AS TOTAL_REVENUE

FROM TICKET T

GROUP BY T.EVENT_NAME;
```



EVENT_NAME	TOTAL_REVENUE
Abstract Showcase	40
Classic Art Week	80
Modern Art Fair	70
Spring Art Expo	50
Summer Sculpture Show	60

Get customer orders with recipient and payment method details

```
SELECT P.FIRST_NAME || ' ' || P.LAST_NAME AS CUSTOMER_NAME, O.ORDER_ID, O.RECIEPIENT,
O.PAYMENT_METHOD, O.ORDER_DATE

FROM ORDER_T O

JOIN CUSTOMER C ON O.CUSTOMER_ID = C.CUSTOMER_ID

JOIN PERSON P ON C.CUSTOMER_ID = P.PERSON_ID;
```

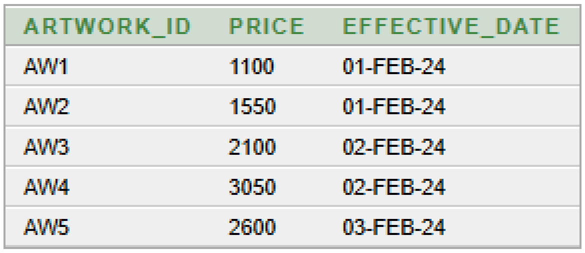


CUSTOMER_NAME	ORDER_ID	RECIEPIENT	PAYMENT_METHOD	ORDER_DATE
John Doe	O1	John Doe	Credit Card	01-MAR-24
Mark Brown	O2	Mark Brown	PayPal	02-MAR-24
Mike Wilson	O3	Mike Wilson	Cash	03-MAR-24

Retrieve the most recent price for each artwork

```
SELECT ARTWORK_ID, PRICE, EFFECTIVE_DATE
```

```
FROM ARTWORK_PRICE AP1
WHERE EFFECTIVE_DATE = (
    SELECT MAX(EFFECTIVE_DATE)
    FROM ARTWORK_PRICE AP2
    WHERE AP1.ARTWORK_ID = AP2.ARTWORK_ID
);
```



ARTWORK_ID	PRICE	EFFECTIVE_DATE
AW1	1100	01-FEB-24
AW2	1550	01-FEB-24
AW3	2100	02-FEB-24
AW4	3050	02-FEB-24
AW5	2600	03-FEB-24

Triggers:

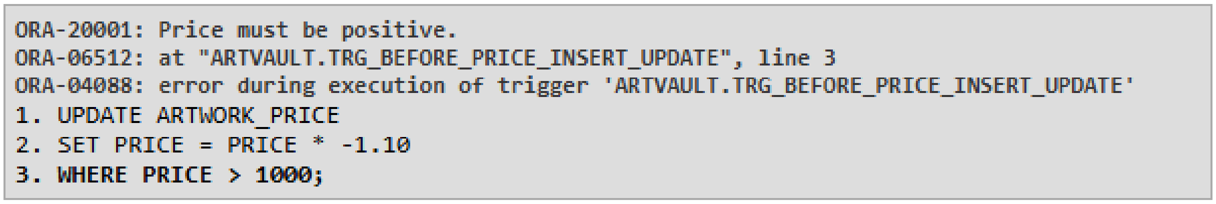
Insert and Update trigger o Artwork_Price:

Purpose: This trigger is invoked before any insert or update operation on the ARTWORK_PRICE table.
Logic: It checks if the PRICE being inserted or updated is less than or equal to 0. If so, it raises an error with the message "Price must be positive," preventing the operation from proceeding

```
CREATE OR REPLACE TRIGGER trg_before_price_insert_update
BEFORE INSERT OR UPDATE ON ARTWORK_PRICE
FOR EACH ROW
BEGIN
  IF :NEW.PRICE <= 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Price must be positive.');
  END IF;
END;
/
```

IMPLEMENTATION OF TRIGGER:

```
UPDATE ARTWORK_PRICE
SET PRICE = PRICE * -1.10
WHERE PRICE > 1000;
```



```
ORA-20001: Price must be positive.
ORA-06512: at "ARTVAULT.TRG_BEFORE_PRICE_INSERT_UPDATE", line 3
ORA-04088: error during execution of trigger 'ARTVAULT.TRG_BEFORE_PRICE_INSERT_UPDATE'
1. UPDATE ARTWORK_PRICE
2. SET PRICE = PRICE * -1.10
3. WHERE PRICE > 1000;
```

Update Artwork Status After an Order

Purpose: This trigger is invoked after an insert operation on the ORDERLINE table.
Logic: When a new order line is added (indicating an artwork has been ordered), it updates the STATUS of the corresponding artwork in the ARTWORK table to "Sold."

```
CREATE OR REPLACE TRIGGER trg_update_artwork_status

AFTER INSERT ON ORDERLINE

FOR EACH ROW

BEGIN

  UPDATE ARTWORK

  SET STATUS = 'Sold'

  WHERE ARTWORK_ID = :NEW.ARTWORK_ID;

END;

/
```

Trigger to Ensure Artwork Stock is Updated:

- **Purpose**: When an artwork is ordered, its stock quantity should be updated.
- **Logic**: After inserting an order line, the trigger will decrease the stock of the corresponding artwork.

```
CREATE OR REPLACE TRIGGER trg_update_artwork_stock
AFTER INSERT ON ORDERLINE
FOR EACH ROW
BEGIN
  UPDATE ARTWORK
  SET STOCKED_QUANTITY = STOCKED_QUANTITY - 1
  WHERE ARTWORK_ID = :NEW.ARTWORK_ID;
END;
/
```

Trigger for Preventing Negative Stock:

- **Purpose**: Ensures that stock levels cannot go below zero.
- **Logic**: Before updating or inserting a new order line, the trigger checks if there is enough stock for the artwork.

```
CREATE OR REPLACE TRIGGER trg_check_stock_before_order
BEFORE INSERT ON ORDERLINE
```

```
FOR EACH ROW
BEGIN
  DECLARE
    stock_count NUMBER;
  BEGIN
    SELECT STOCKED_QUANTITY INTO stock_count
    FROM ARTWORK
    WHERE ARTWORK_ID = :NEW.ARTWORK_ID;

    IF stock_count <= 0 THEN
      RAISE_APPLICATION_ERROR(-20002, 'Not enough stock available.');
    END IF;
  END;
END;
/
```

Trigger for Order Status Change:

- **Purpose**: Update the order status to "Completed" when the payment is processed.
- **Logic**: After the payment method is updated or an order is marked as paid, the trigger changes the order status to "Completed."

```
CREATE OR REPLACE TRIGGER trg_update_order_status
AFTER UPDATE OF PAYMENT_METHOD ON ORDER_T
FOR EACH ROW
BEGIN
  IF :NEW.PAYMENT_METHOD IS NOT NULL THEN
    UPDATE ORDER_T
    SET ORDER_STATUS = 'Completed'
    WHERE ORDER_ID = :NEW.ORDER_ID;
  END IF;
END;
/
```

Trigger for Preventing Price Updates on Sold Artwork:

- **Purpose**: Prevents price updates on artwork that has already been marked as "Sold."
- **Logic**: Before updating the ARTWORK_PRICE, the trigger checks if the artwork's status is "Sold" and raises an error if so.

sql
Copy code

```
CREATE OR REPLACE TRIGGER trg_prevent_price_update_sold_artwork
BEFORE UPDATE ON ARTWORK_PRICE
FOR EACH ROW
BEGIN
  DECLARE
    artwork_status VARCHAR2(20);
  BEGIN
    SELECT STATUS INTO artwork_status
    FROM ARTWORK
    WHERE ARTWORK_ID = :NEW.ARTWORK_ID;
    IF artwork_status = 'Sold' THEN
      RAISE_APPLICATION_ERROR(-20003, 'Cannot update price of sold artwork.');
    END IF;
  END;
END;
/
```

Roles:

ADMIN:

Description: The ADMIN role has full system access, allowing for complete control over the database schema. It can manage all tables, perform CRUD operations, and create or modify database objects like tables, views, procedures, and triggers. This role is essential for maintaining system integrity and security.

```
CREATE ROLE ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.ARTWORK TO ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.CUSTOMER TO ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.ORDER_T TO ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.ORDERLINE TO ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.ARTWORK_PRICE TO ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.EVENT TO ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.TICKET TO ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.EVENT_EMPLOYEE TO ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.EVENT_ARTWORK TO ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON YOUR_SCHEMA.EMPLOYEE TO ADMIN;

GRANT CREATE TABLE TO ADMIN;

GRANT CREATE VIEW TO ADMIN;

GRANT CREATE PROCEDURE TO ADMIN;

GRANT CREATE TRIGGER TO ADMIN;


CREATE USER EMAN IDENTIFIED BY EMAN;

GRANT ADMIN TO EMAN;
```

SALE MANAGER:

Description: The SALES_MANAGER role is responsible for managing orders, updating customer details, and viewing artwork pricing. This role allows users to create, update, and view orders and artwork prices. It also allows updating customer information but restricts access to deleting or creating customer records.

```
CREATE ROLE SALES_MANAGER;

GRANT SELECT, INSERT, UPDATE ON ORDER_T TO SALES_MANAGER;

GRANT SELECT, UPDATE ON CUSTOMER TO SALES_MANAGER;

GRANT SELECT, INSERT, UPDATE ON ARTWORK_PRICE TO SALES_MANAGER;

GRANT SELECT ON ARTWORK TO SALES_MANAGER;


CREATE USER SALES_USER IDENTIFIED BY SALES123;

GRANT SALES_MANAGER TO SALES_USER;

GRANT CREATE SESSION TO SALES_USER;
```

INVENTORY MANAGER:

Description: The INVENTORY_MANAGER role is responsible for managing artwork availability and pricing. This role allows users to view and update the stock and pricing of artwork but restricts access to other tables, such as orders. Users can create and modify artwork prices but not manage customers or orders.

```
CREATE ROLE INVENTORY_MANAGER;


GRANT SELECT, UPDATE ON ARTWORK TO INVENTORY_MANAGER;

GRANT SELECT, INSERT, UPDATE ON ARTWORK_PRICE TO INVENTORY_MANAGER;


CREATE USER INVENTORY_USER IDENTIFIED BY INVENTORY123;

GRANT INVENTORY_MANAGER TO INVENTORY_USER;

GRANT CREATE SESSION TO INVENTORY_USER;
```

CUSTOMER SUPPORT:

Description: The CUSTOMER_SUPPORT role assists customers with orders and inquiries. It allows users to view orders, customers, and artwork information, and update order statuses. This role helps customer support staff assist customers but does not allow managing artwork prices or customers directly.

Assist customers with orders and inquiries.

```
CREATE ROLE CUSTOMER_SUPPORT;
```

```
GRANT SELECT ON ORDER_T TO CUSTOMER_SUPPORT;
GRANT SELECT ON CUSTOMER TO CUSTOMER_SUPPORT;
GRANT SELECT ON ARTWORK TO CUSTOMER_SUPPORT;
GRANT UPDATE ON ORDER_T TO CUSTOMER_SUPPORT;


CREATE USER SUPPORT_USER IDENTIFIED BY SUPPORT123;
GRANT CUSTOMER_SUPPORT TO SUPPORT_USER;
GRANT CREATE SESSION TO SUPPORT_USER;
```

Database size estimate:

For 5 records in each table, the total database size is approximately **6.5 KB** (data only).

1. PERSON:

- **Row size**: 1049 bytes.
- **Table size**: 1049×5 = 5245 bytes.

2. CUSTOMER:

- **Row size**: 67 bytes.
- **Table size**: 67×5= 335 bytes.

3. EMPLOYEE:

- **Row size**: 356 bytes.
- **Table size**: 356×5= 1780 bytes.

4. ARTIST:

- **Row size**: = 460 bytes.
- **Table size**: 460×5 = 575 bytes.

5. ORDER_T:

- **Row size**: 487 bytes.
- **Table size**: 487×5 = 2435 bytes.

6. ARTWORK:

- **Row size**: 807 bytes.
- **Table size**: 807×5 = 4035 bytes.

7. ORDERLINE:

- **Row size**: 120 bytes.

- **Table size**: 120×5 = 600 bytes.

8. Artwork_Price:

- **Row size**: 78 bytes.
- **Table size**: 78×5= 390 bytes.

9. EVENT:

- **Row size**: 674 bytes.
- **Table size**: 674×5 = 3370 bytes.

10. TICKET:

- **Row size**: 219 bytes.
- **Table size**: 219×5=1095 bytes.

11. EVENT_EMPLOYEE:

- **Row size**: 469 bytes.
- **Table size**: 469×5 = 2345 bytes.

12. EVENT_ARTWORK:

- **Row size**: 267 bytes.
- **Table size**: 267×5= 1335 bytes.

Total Database Size:

Summing up all the table sizes: **23540 bytes** (≈**23.5 KB**).