Disney World Theme Park Database Project

Anastasia LaPeruta

CMPT308

Table of Contents

Executive Summary

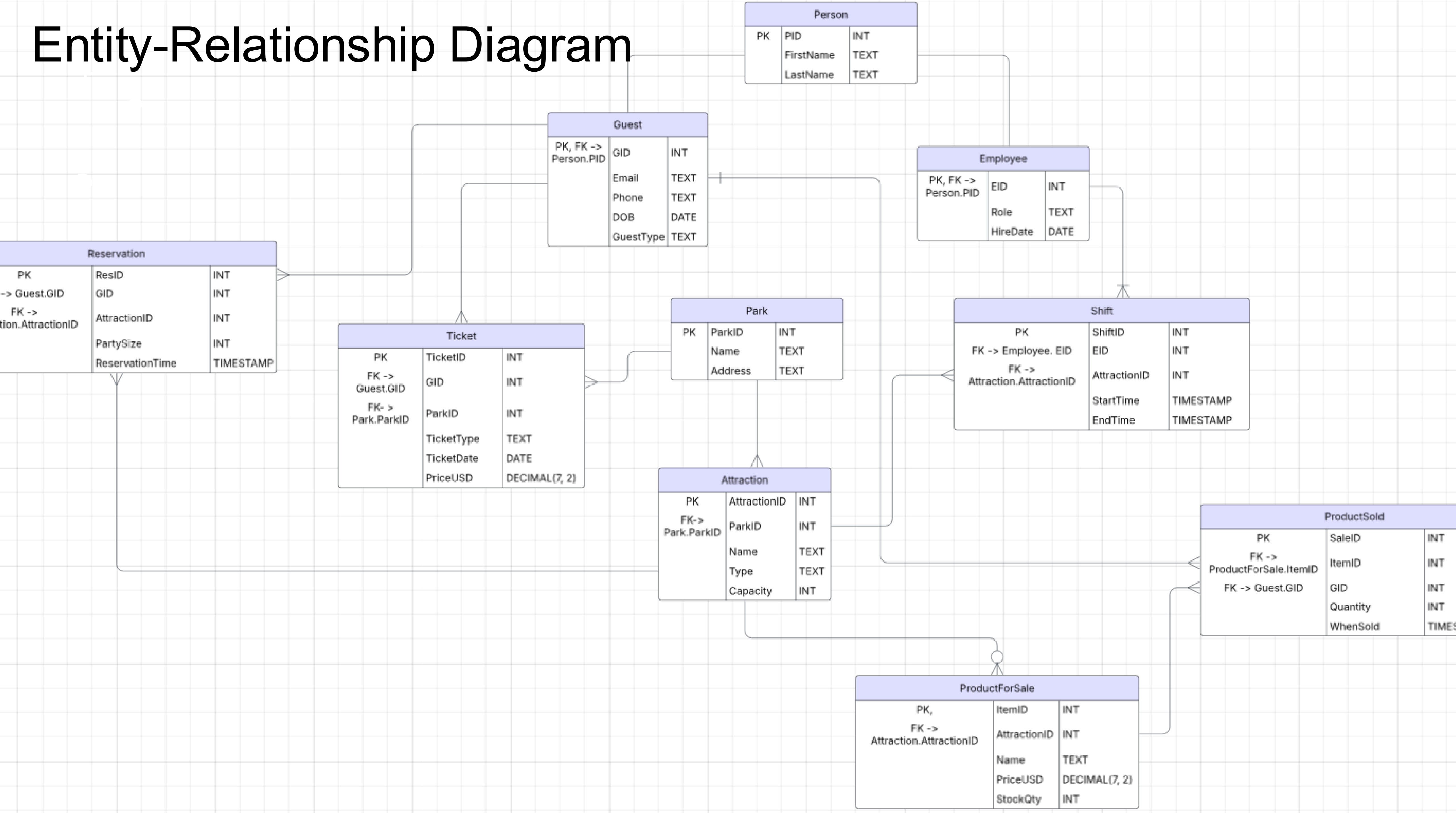
Overview:

- This project captures the operations of Disney World Theme Park. It includes data on guests, employees, parks, attractions (rides, restaurants and stores), tickets, reservations, products for sale, sales transactions, and staff shifts.

Objectives:

- Design normalized tables that represent various factors of the theme park
- Enforce business rules through the use of constraints, triggers, and stored procedures
- Provide views, reports, and access controls for different user roles
- Secure data access with roles and GRANT/REVOKE.

Entity-Relationship Diagram



Person		
PK	PID	INT
	FirstName	TEXT
	LastName	TEXT

Guest		
PK, FK -> Person.PID	GID	INT
	Email	TEXT
	Phone	TEXT
	DOB	DATE
	GuestType	TEXT

Employee		
PK, FK -> Person.PID	EID	INT
	Role	TEXT
	HireDate	DATE

Reservation		
PK	ResID	INT
-> Guest.GID	GID	INT
FK -> tion.AttractionID	AttractionID	INT
	PartySize	INT
	ReservationTime	TIMESTAMP

Ticket		
PK	TicketID	INT
FK -> Guest.GID	GID	INT
FK- > Park.ParkID	ParkID	INT
	TicketType	TEXT
	TicketDate	DATE
	PriceUSD	DECIMAL(7, 2)

Park		
PK	ParkID	INT
	Name	TEXT
	Address	TEXT

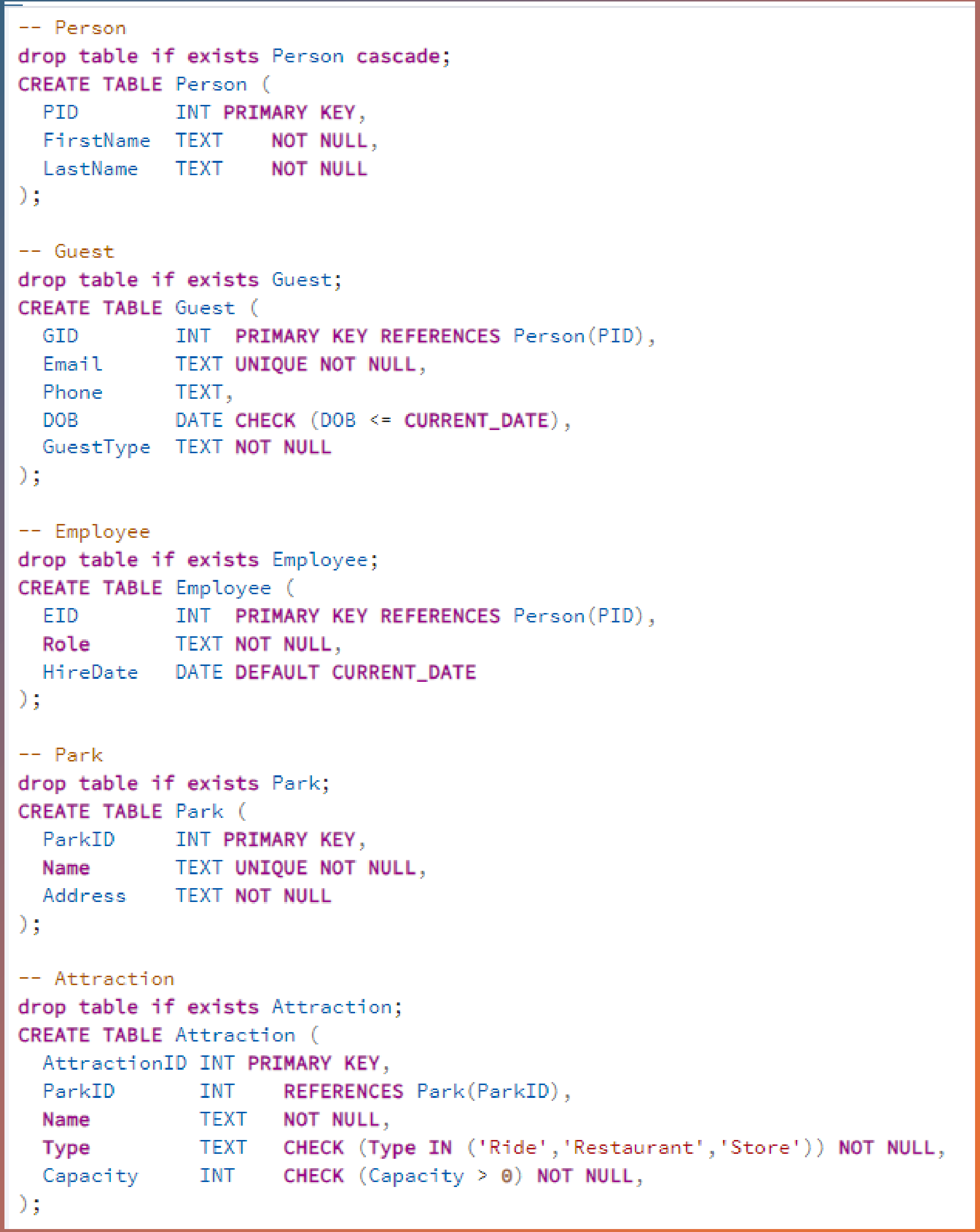
Shift		
PK	ShiftID	INT
FK -> Employee. EID	EID	INT
FK -> Attraction.AttractionID	AttractionID	INT
	StartTime	TIMESTAMP
	EndTime	TIMESTAMP

Attraction		
PK	AttractionID	INT
FK-> Park.ParkID	ParkID	INT
	Name	TEXT
	Type	TEXT
	Capacity	INT

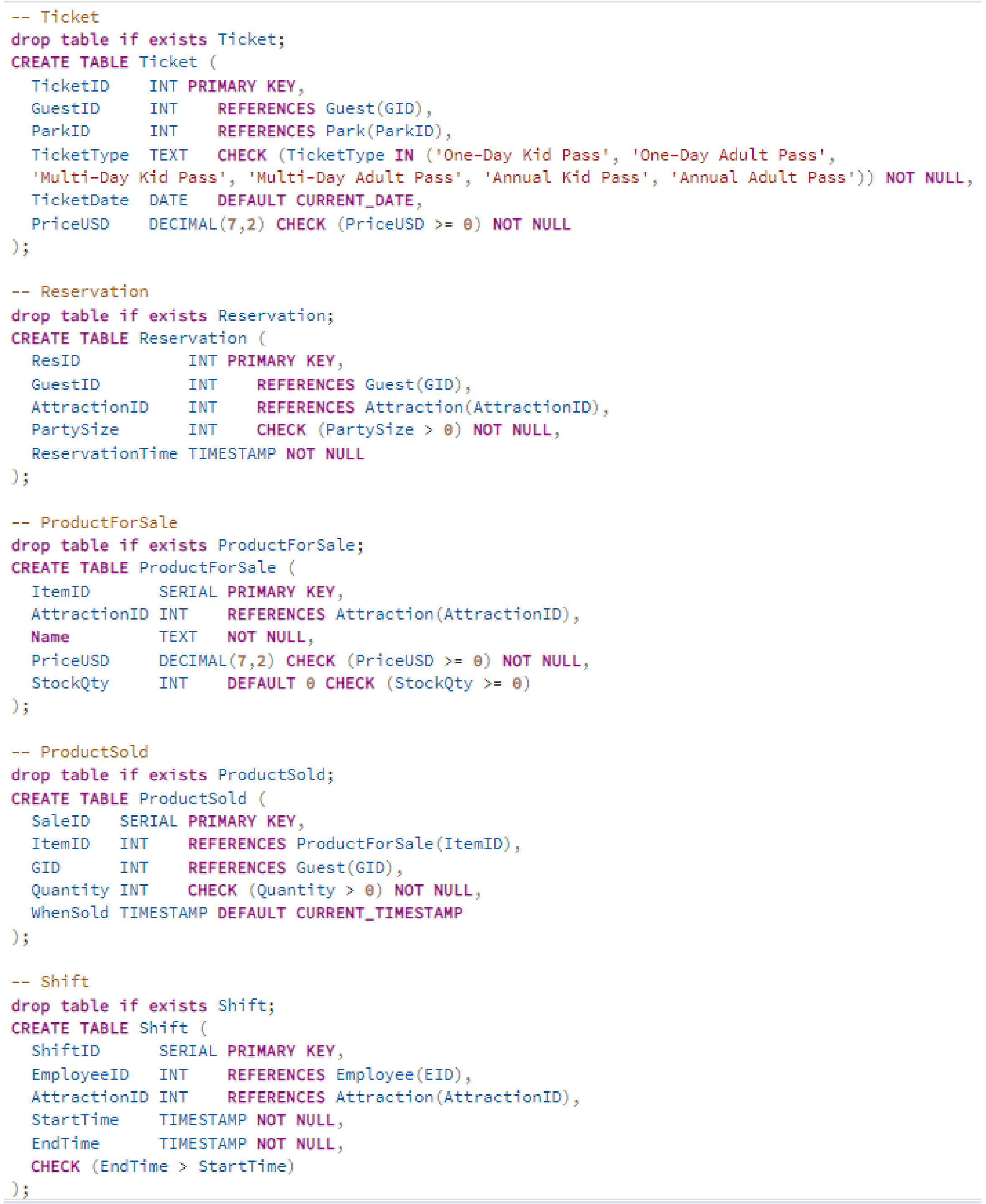
ProductSold		
PK	SaleID	INT
FK -> ProductForSale.ItemID	ItemID	INT
FK -> Guest.GID	GID	INT
	Quantity	INT
	WhenSold	TIMES

ProductForSale		
PK,	ItemID	INT
FK -> Attraction.AttractionID	AttractionID	INT
	Name	TEXT
	PriceUSD	DECIMAL(7, 2)
	StockQty	INT

+ Tables:



```
-- Person
drop table if exists Person cascade;
CREATE TABLE Person (
  PID         INT PRIMARY KEY,
  FirstName   TEXT    NOT NULL,
  LastName    TEXT    NOT NULL
);

-- Guest
drop table if exists Guest;
CREATE TABLE Guest (
  GID         INT  PRIMARY KEY REFERENCES Person(PID),
  Email       TEXT UNIQUE NOT NULL,
  Phone       TEXT,
  DOB         DATE CHECK (DOB <= CURRENT_DATE),
  GuestType   TEXT NOT NULL
);

-- Employee
drop table if exists Employee;
CREATE TABLE Employee (
  EID         INT  PRIMARY KEY REFERENCES Person(PID),
  Role        TEXT NOT NULL,
  HireDate    DATE DEFAULT CURRENT_DATE
);

-- Park
drop table if exists Park;
CREATE TABLE Park (
  ParkID      INT PRIMARY KEY,
  Name        TEXT UNIQUE NOT NULL,
  Address     TEXT NOT NULL
);

-- Attraction
drop table if exists Attraction;
CREATE TABLE Attraction (
  AttractionID INT PRIMARY KEY,
  ParkID       INT    REFERENCES Park(ParkID),
  Name         TEXT   NOT NULL,
  Type         TEXT   CHECK (Type IN ('Ride','Restaurant','Store')) NOT NULL,
  Capacity     INT    CHECK (Capacity > 0) NOT NULL,
);
```



```
-- Ticket
drop table if exists Ticket;
CREATE TABLE Ticket (
  TicketID    INT PRIMARY KEY,
  GuestID     INT    REFERENCES Guest(GID),
  ParkID      INT    REFERENCES Park(ParkID),
  TicketType  TEXT   CHECK (TicketType IN ('One-Day Kid Pass', 'One-Day Adult Pass',
  'Multi-Day Kid Pass', 'Multi-Day Adult Pass', 'Annual Kid Pass', 'Annual Adult Pass')) NOT NULL,
  TicketDate  DATE   DEFAULT CURRENT_DATE,
  PriceUSD    DECIMAL(7,2) CHECK (PriceUSD >= 0) NOT NULL
);

-- Reservation
drop table if exists Reservation;
CREATE TABLE Reservation (
  ResID           INT PRIMARY KEY,
  GuestID         INT    REFERENCES Guest(GID),
  AttractionID    INT    REFERENCES Attraction(AttractionID),
  PartySize       INT    CHECK (PartySize > 0) NOT NULL,
  ReservationTime TIMESTAMP NOT NULL
);

-- ProductForSale
drop table if exists ProductForSale;
CREATE TABLE ProductForSale (
  ItemID       SERIAL PRIMARY KEY,
  AttractionID INT    REFERENCES Attraction(AttractionID),
  Name         TEXT   NOT NULL,
  PriceUSD     DECIMAL(7,2) CHECK (PriceUSD >= 0) NOT NULL,
  StockQty     INT    DEFAULT 0 CHECK (StockQty >= 0)
);

-- ProductSold
drop table if exists ProductSold;
CREATE TABLE ProductSold (
  SaleID   SERIAL PRIMARY KEY,
  ItemID   INT    REFERENCES ProductForSale(ItemID),
  GID      INT    REFERENCES Guest(GID),
  Quantity INT    CHECK (Quantity > 0) NOT NULL,
  WhenSold TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Shift
drop table if exists Shift;
CREATE TABLE Shift (
  ShiftID      SERIAL PRIMARY KEY,
  EmployeeID   INT    REFERENCES Employee(EID),
  AttractionID INT    REFERENCES Attraction(AttractionID),
  StartTime    TIMESTAMP NOT NULL,
  EndTime      TIMESTAMP NOT NULL,
  CHECK (EndTime > StartTime)
);
```

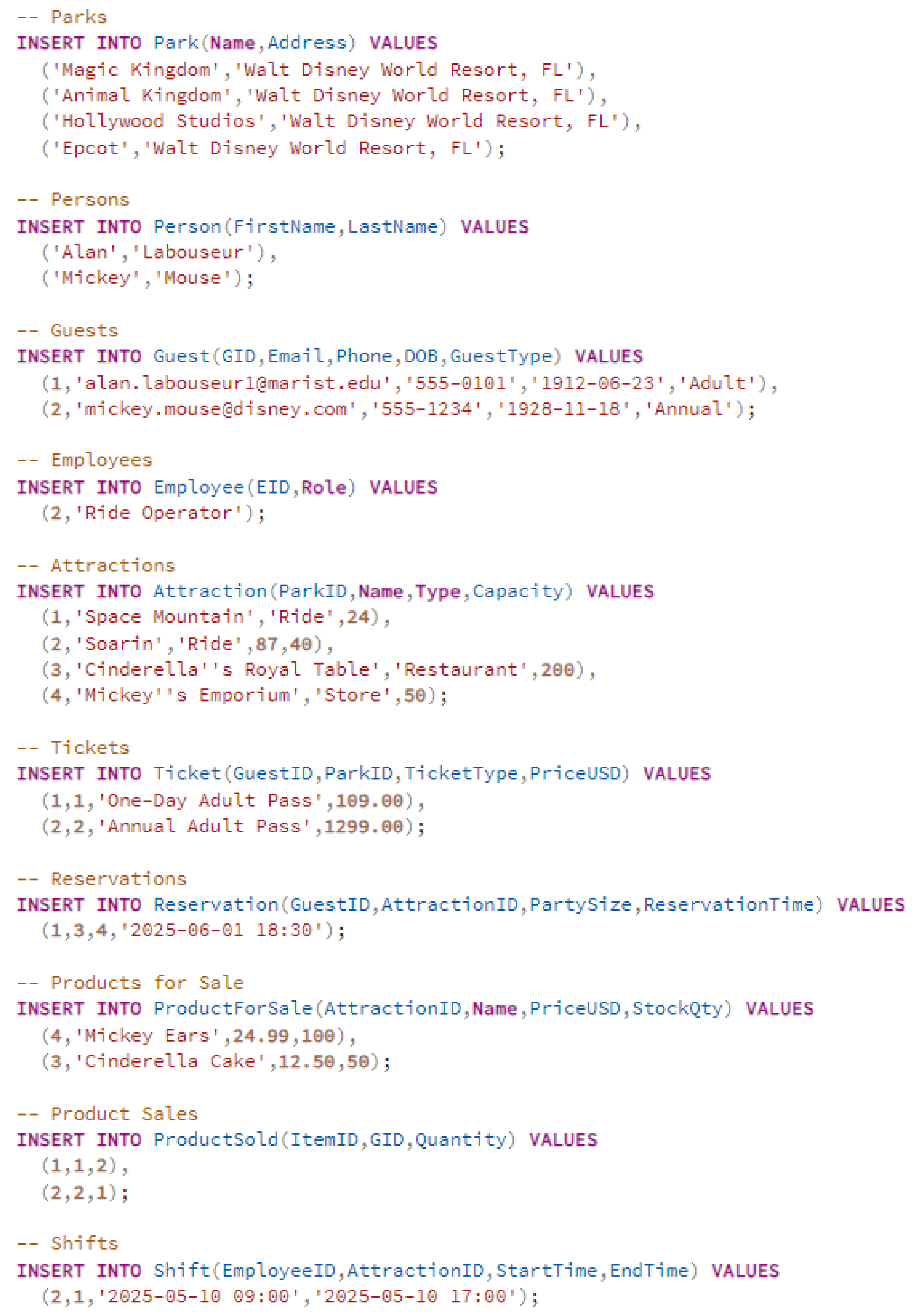
Tables:

Functional Dependencies

- **Person**: PID → FirstName, LastName
- **Guest**: GID → Email, Phone, DOB, GuestType
- **Employee**: EID → Role, HireDate
- **Park**: ParkID → Name, Address
- **Attraction**: AttractionID → ParkID, Name, Type, Capacity
- **Ticket**: TicketID → GuestID, ParkID, TicketType, TicketDate, PriceUSD
- **Reservation**: ResID → GuestID, AttractionID, PartySize, ReservationTime
- **ProductForSale**: ItemID → AttractionID, Name, PriceUSD, StockQty
- **ProductSold**: SaleID → ItemID, GID, Quantity, WhenSold
- **Shift**: ShiftID → EmployeeID, AttractionID, StartTime, EndTime

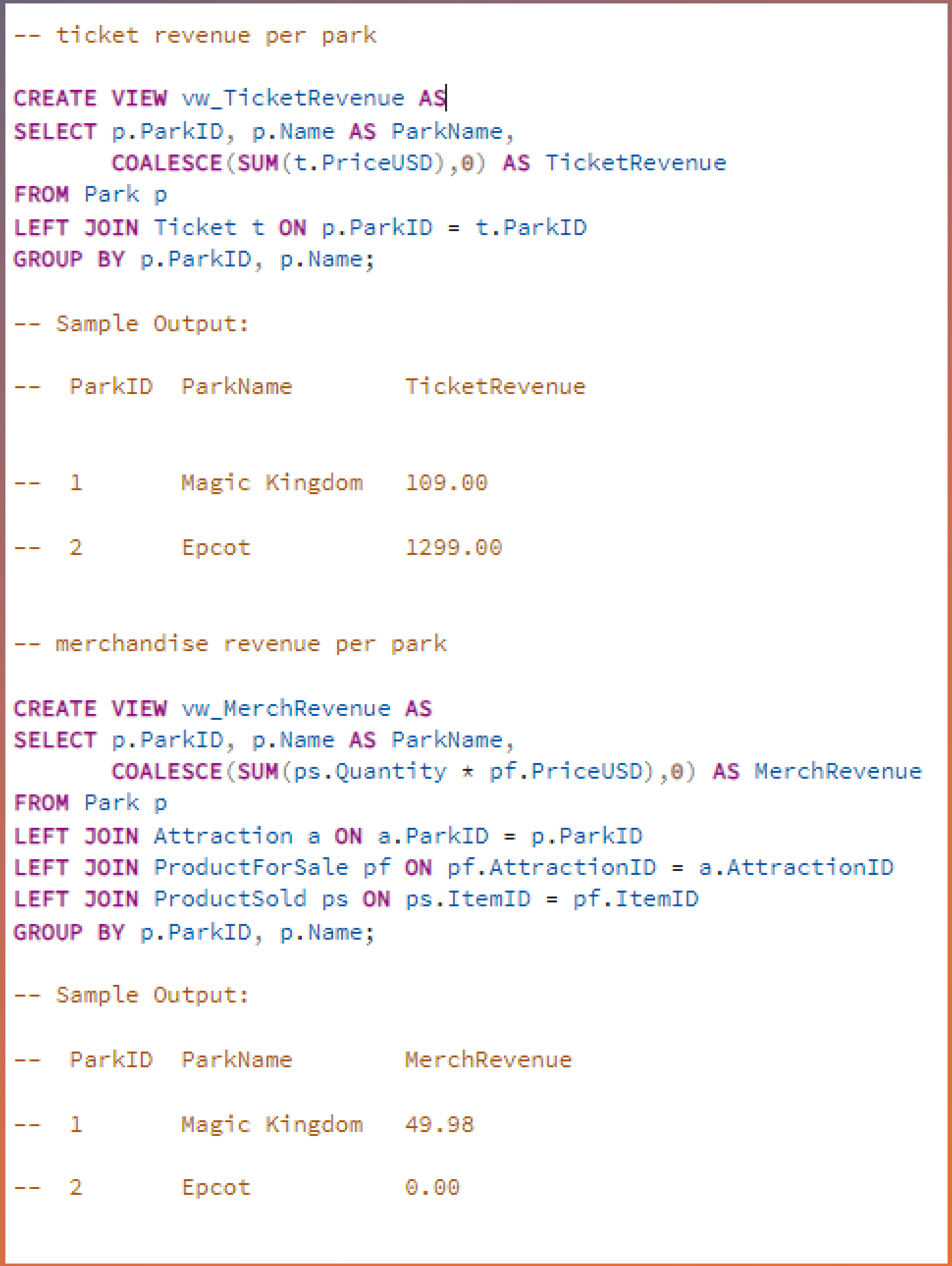
Tables:

Sample/ Test Data



```sql
-- Parks
INSERT INTO Park(Name,Address) VALUES
  ('Magic Kingdom','Walt Disney World Resort, FL'),
  ('Animal Kingdom','Walt Disney World Resort, FL'),
  ('Hollywood Studios','Walt Disney World Resort, FL'),
  ('Epcot','Walt Disney World Resort, FL');

-- Persons
INSERT INTO Person(FirstName,LastName) VALUES
  ('Alan','Labouseur'),
  ('Mickey','Mouse');

-- Guests
INSERT INTO Guest(GID,Email,Phone,DOB,GuestType) VALUES
  (1,'alan.labouseur1@marist.edu','555-0101','1912-06-23','Adult'),
  (2,'mickey.mouse@disney.com','555-1234','1928-11-18','Annual');

-- Employees
INSERT INTO Employee(EID,Role) VALUES
  (2,'Ride Operator');

-- Attractions
INSERT INTO Attraction(ParkID,Name,Type,Capacity) VALUES
  (1,'Space Mountain','Ride',24),
  (2,'Soarin','Ride',87,40),
  (3,'Cinderella''s Royal Table','Restaurant',200),
  (4,'Mickey''s Emporium','Store',50);

-- Tickets
INSERT INTO Ticket(GuestID,ParkID,TicketType,PriceUSD) VALUES
  (1,1,'One-Day Adult Pass',109.00),
  (2,2,'Annual Adult Pass',1299.00);

-- Reservations
INSERT INTO Reservation(GuestID,AttractionID,PartySize,ReservationTime) VALUES
  (1,3,4,'2025-06-01 18:30');

-- Products for Sale
INSERT INTO ProductForSale(AttractionID,Name,PriceUSD,StockQty) VALUES
  (4,'Mickey Ears',24.99,100),
  (3,'Cinderella Cake',12.50,50);

-- Product Sales
INSERT INTO ProductSold(ItemID,GID,Quantity) VALUES
  (1,1,2),
  (2,2,1);

-- Shifts
INSERT INTO Shift(EmployeeID,AttractionID,StartTime,EndTime) VALUES
  (2,1,'2025-05-10 09:00','2025-05-10 17:00');
```

View Definitions and Sample Output

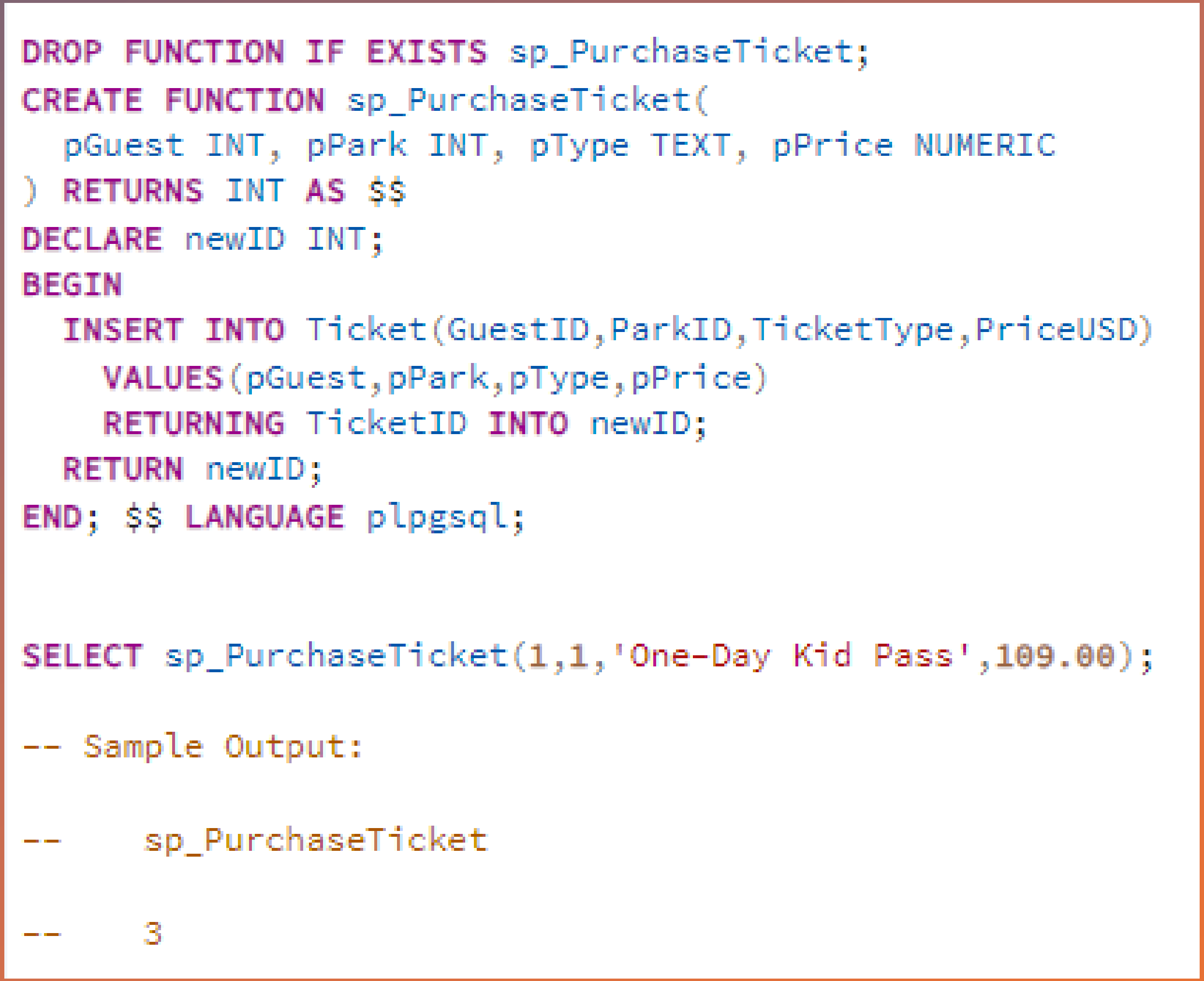


```
-- ticket revenue per park

CREATE VIEW vw_TicketRevenue AS
SELECT p.ParkID, p.Name AS ParkName,
       COALESCE(SUM(t.PriceUSD),0) AS TicketRevenue
FROM Park p
LEFT JOIN Ticket t ON p.ParkID = t.ParkID
GROUP BY p.ParkID, p.Name;

-- Sample Output:

--  ParkID  ParkName        TicketRevenue


--  1       Magic Kingdom   109.00

--  2       Epcot           1299.00


-- merchandise revenue per park

CREATE VIEW vw_MerchRevenue AS
SELECT p.ParkID, p.Name AS ParkName,
       COALESCE(SUM(ps.Quantity * pf.PriceUSD),0) AS MerchRevenue
FROM Park p
LEFT JOIN Attraction a ON a.ParkID = p.ParkID
LEFT JOIN ProductForSale pf ON pf.AttractionID = a.AttractionID
LEFT JOIN ProductSold ps ON ps.ItemID = pf.ItemID
GROUP BY p.ParkID, p.Name;

-- Sample Output:

--  ParkID  ParkName        MerchRevenue

--  1       Magic Kingdom   49.98

--  2       Epcot           0.00
```

Reports and their Queries with Sample Output

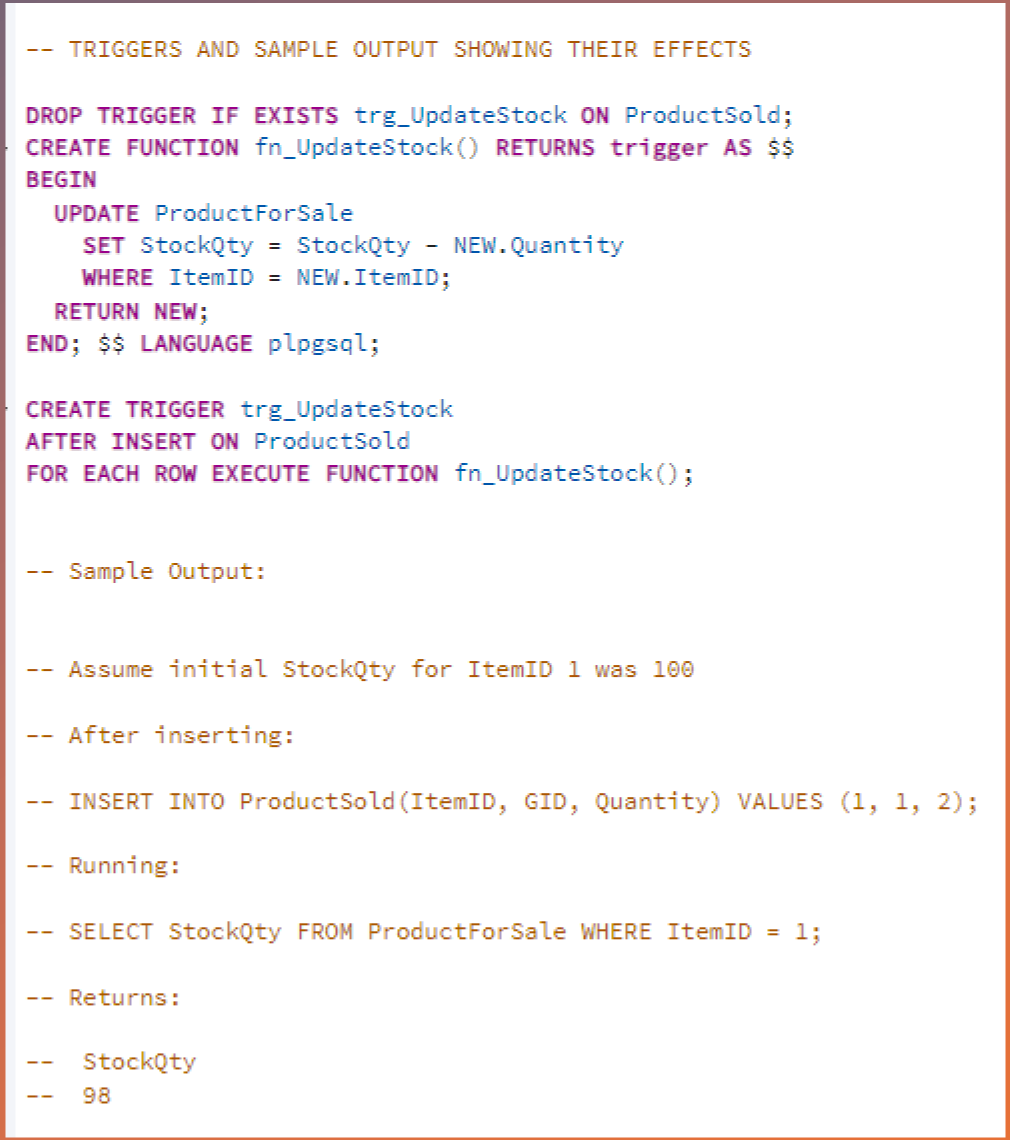
```sql
-- daily revenue (tickets + merchandise)

SELECT DATE(t.TicketDate) AS Day,
       SUM(t.PriceUSD) + COALESCE(SUM(ps.Quantity * pf.PriceUSD),0) AS TotalRevenue
FROM Ticket t
LEFT JOIN ProductSold ps ON DATE(ps.WhenSold) = DATE(t.TicketDate)
LEFT JOIN ProductForSale pf ON pf.ItemID = ps.ItemID
GROUP BY Day;


-- Sample Output:

--   Day            TotalRevenue

--   2025-05-10     158.98



-- top attractions by reservations count

SELECT a.Name, COUNT(r.ResID) AS NumReservations
FROM Attraction a
LEFT JOIN Reservation r ON r.AttractionID = a.AttractionID
GROUP BY a.Name
ORDER BY NumReservations DESC;


-- Sample Output:

--   Name                        NumReservations

--   Cinderella's Royal Table    1

--   Space Mountain              0

--   Soarin                      0

--   Mickey's Emporium           0
```

Stored Procedures and Sample Output Showing their Results



```
DROP FUNCTION IF EXISTS sp_PurchaseTicket;
CREATE FUNCTION sp_PurchaseTicket(
  pGuest INT, pPark INT, pType TEXT, pPrice NUMERIC
) RETURNS INT AS $$
DECLARE newID INT;
BEGIN
  INSERT INTO Ticket(GuestID,ParkID,TicketType,PriceUSD)
    VALUES(pGuest,pPark,pType,pPrice)
    RETURNING TicketID INTO newID;
  RETURN newID;
END; $$ LANGUAGE plpgsql;


SELECT sp_PurchaseTicket(1,1,'One-Day Kid Pass',109.00);

-- Sample Output:

--    sp_PurchaseTicket

--    3
```

Triggers and Sample Output Showing Their Effects



```
-- TRIGGERS AND SAMPLE OUTPUT SHOWING THEIR EFFECTS

DROP TRIGGER IF EXISTS trg_UpdateStock ON ProductSold;
CREATE FUNCTION fn_UpdateStock() RETURNS trigger AS $$
BEGIN
  UPDATE ProductForSale
    SET StockQty = StockQty - NEW.Quantity
    WHERE ItemID = NEW.ItemID;
  RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER trg_UpdateStock
AFTER INSERT ON ProductSold
FOR EACH ROW EXECUTE FUNCTION fn_UpdateStock();


-- Sample Output:


-- Assume initial StockQty for ItemID 1 was 100

-- After inserting:

-- INSERT INTO ProductSold(ItemID, GID, Quantity) VALUES (1, 1, 2);

-- Running:

-- SELECT StockQty FROM ProductForSale WHERE ItemID = 1;

-- Returns:

--  StockQty
--  98
```

Security - Grant and Revoke for Users and Groups

```
-- Create group roles
DO $$ BEGIN
  CREATE ROLE guest_group NOLOGIN;
  CREATE ROLE manager_group NOLOGIN;
EXCEPTION WHEN duplicate_object THEN NULL;
END; $$;

-- Assign privileges to groups
GRANT SELECT ON vw_TicketRevenue, vw_MerchRevenue TO guest_group;
GRANT EXECUTE ON FUNCTION sp_PurchaseTicket(INT,INT,TEXT,NUMERIC) TO guest_group;
GRANT ALL ON Ticket, Reservation TO manager_group;

-- Create individual users
CREATE USER alice LOGIN PASSWORD 'password123';
CREATE USER bob   LOGIN PASSWORD 'password123';

-- Add users to groups
GRANT guest_group TO alice;
GRANT manager_group TO bob;

-- Revoke undesired privileges from users
REVOKE INSERT ON ProductForSale FROM guest_group;
```

Implementation Notes

All tables are structured in third normal form, making sure there is no redundancy and that there are clear foreign key relationships between entities. CHECK constraints are applied to prevent invalid data entries from taking place. Triggers automatically adjust stock quantities after each product sale, ensuring accurate inventory levels as purchases are made. Stored procedures make sure that complex transactions execute atomically and preserve data consistency.

Known Problems

- Small database: there could be many more tables if we were to get into finer-grained details
- Performance: simple views may be slow on larger data entries

Future Enhancements

- Could add even more tables to cover more aspects of Disney World theme park
- Seeing how the database handles larger sets of data and seeing if improvements in performance are possible
- Expanding on the security features and user-group management