Database Design Project

CSU34041

Jack Gilbride – STUDENT Number 17340868

2019

# Description

The database that I chose to design for this project was one for a professional wrestling company. The data seen in the database in this project models that of World Wrestling Entertainment Inc, or WWE. A lot of this information would be easily attainable by fans through applications like WWE.com or the WWE Network subscription model. While most of the database holds information that could be shared outside the company through application, there is also a link to private data, e.g. staff information. The relations that I decided to model are as follows:

### STAFF

This would contain information about staff employed by WWE. It would contain their personal information such as their name, address, data of birth, salary and social security number (SSN). I have assumed that each staff member has a staff number which is unique; this is used as the primary key into the table.

### WRESTLER

Many of the staff of WWE are professional wrestlers who require their own unique attributes that would not need to be stored about regular staff. Thus for every wrestler there is a WRESTLER tuple. Not every staff is a wrestler but every wrestler is staff, so there would be a foreign key from a WRESTLER tuple which links to their STAFF tuple. The attributes include gender, height and weight, which are all important for determining what matches a wrestler may fight in. Gender is in this table because while it is important for a wrestler’s eligibility, there is no reason to store it for staff in general. There is an attribute for what location the wrestler is billed from, which may not be their actual address. This is a single attribute as often it is not as specific as an address which requires multiple attributes. The ring name for the wrestler (i.e. what they are called), which may not be their real name, is the primary key to the tuple as it will be unique for every wrestler.

### CHAMPIONSHIP

There are various championships that a wrestler may hold. These would require database entries to keep track of all of the championships in the company and who’s holding them. Each championship has a unique name which would serve as the primary key in the tuple. The tuple’s attributes are information that may be useful about the championship such as the date it was created, the plate colour and the strap colour of the belt. A championship is held by a wrestler so there will be a foreign key to the WRESTLER table, as well as one to the SHOW table, as each championship belongs to a particular show.

### SHOW

WWE produces multiple weekly shows for television. Each show has a different name, airtime (the length of an episode), channel on which it is shown, and primary colour for promotional materials. These are all attributes of each tuple, where Name is the primary key as each show name is unique. Another attribute is the date on which the show first aired as this may be valuable information to query when giving data about that show.

### EPISODE

Each show runs weekly, so would have multiple episodes. The episode title would be the primary key. Other attributes of EPISODE would be the episode title, start time and number. The end time is not included as this can be calculated from start date and the airtime of the respective show. There are foreign keys to SHOW and ARENA, noting what show the episode is part of and what arena is hosting the show respectively.

### ARENA

Each arena has a unique name (assuming that the city name may be in the arena name), so this is the primary key to the tuple. The attendance of the arena is a useful attribute, as is the location, which is actually three attributes; street, city and state. An arena may host multiple episodes while each episode is only hosted by one arena.

### MATCH

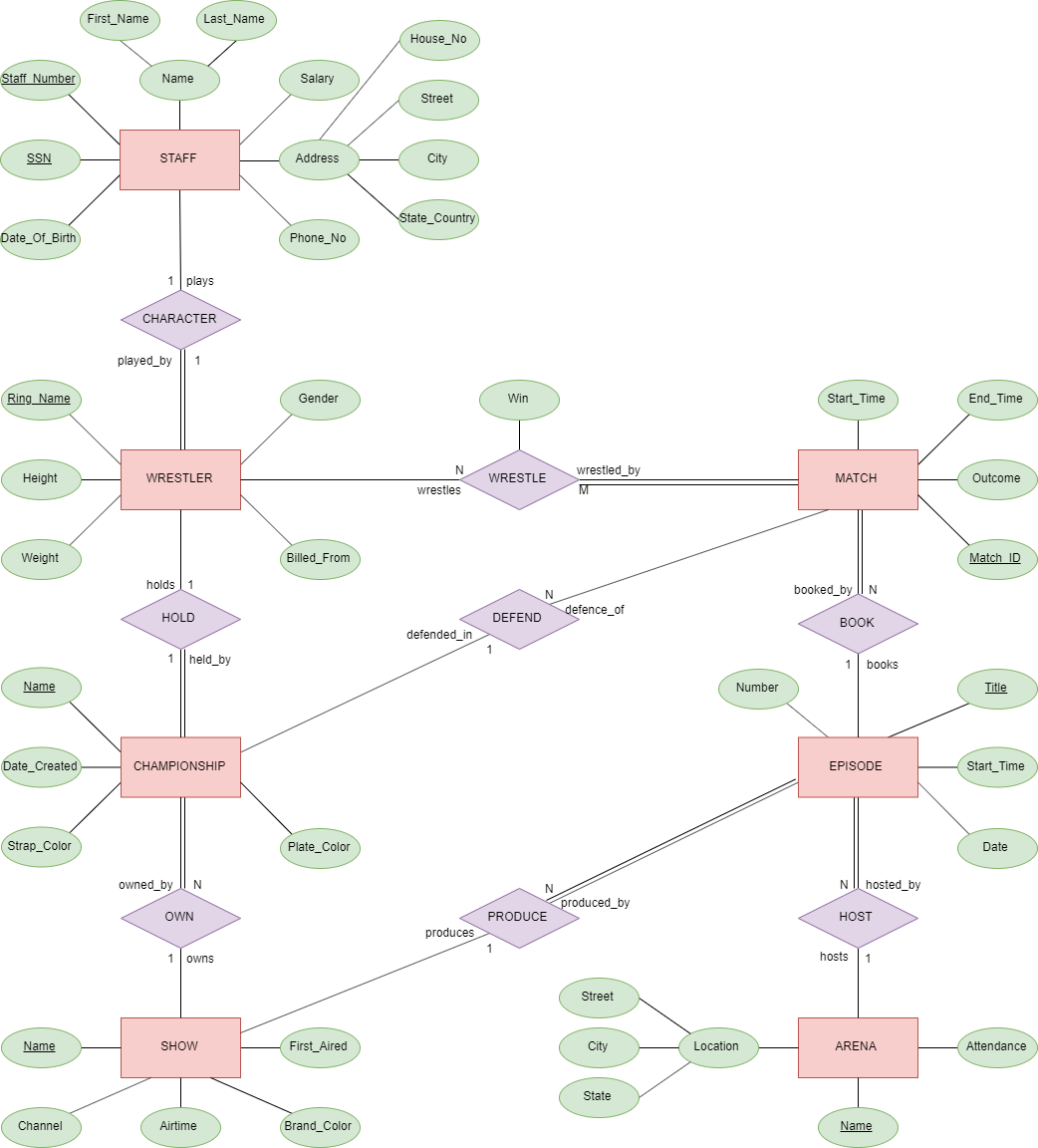
The final relation that I wanted to record was a wrestling match. I decided to assume a unique Match ID as the primary key and identifier of each tuple. The tuple would also record the start time, end time and outcome of the match (i.e. pinfall, submission finish, count-out or disqualification). During a match a championship can be defended, so a foreign key to the CHAMPIONSHIP table can go in the tuple. A match is also booked on a particular episode, so there is also a foreign key to the EPISODE table.

### WRESTLE

A wrestler can wrestle M matches and a match can be wrestled by N wrestlers. As usual when modelling M..N relationships, the relationship is modelled as its own table. The win attribute is added to each tuple in this relation to identify whether a particular wrestler won a particular match. This column does not contain redundant data; one wrestler winning does not mean that all other wrestlers lose, e.g. in a tag team match there can be two winners. Therefore it is important to note whether each wrestler was a winner of that particular match in each tuple.

# Entity Relationship Diagram

I created an entity relationship diagram using the notation introduced to us in lectures. Entity types are represented by rectangles with the relation name inside it, with their attributes surrounding them in ovals. Key attributes are represented by underlining the attribute. In this case the only key attributes were primary keys, apart from in STAFF where SSN would also be a unique value. There was no need to put uniqueness constraints on any other non-primary key attributes. Composite attributes are seen a number of times; as the Name and Address in STAFF, and the Location in ARENA.

Attributes of entity types that refer to other entity types were represented as relationships. The notation for a relationship between two entity types is a diamond between them. Most of the relationships have 1:N cardinality constraints, but there are also 1:1 constraints (STAFF:WRESTLER and WRESTLER:CHAMPIONSHIP) and an N:M constraint (MATCH:WRESTLER). These constraints are in line with the real-world semantics of the data. There are also participation constraints represented by a double line, e.g. a WRESTLER has to be played by a member of STAFF, and a MATCH has to be wrestled by N WRESTLERs. An example of a relationship with an attribute can also be seen in the WRESTLE relationship, with the Win attribute discussed previously.

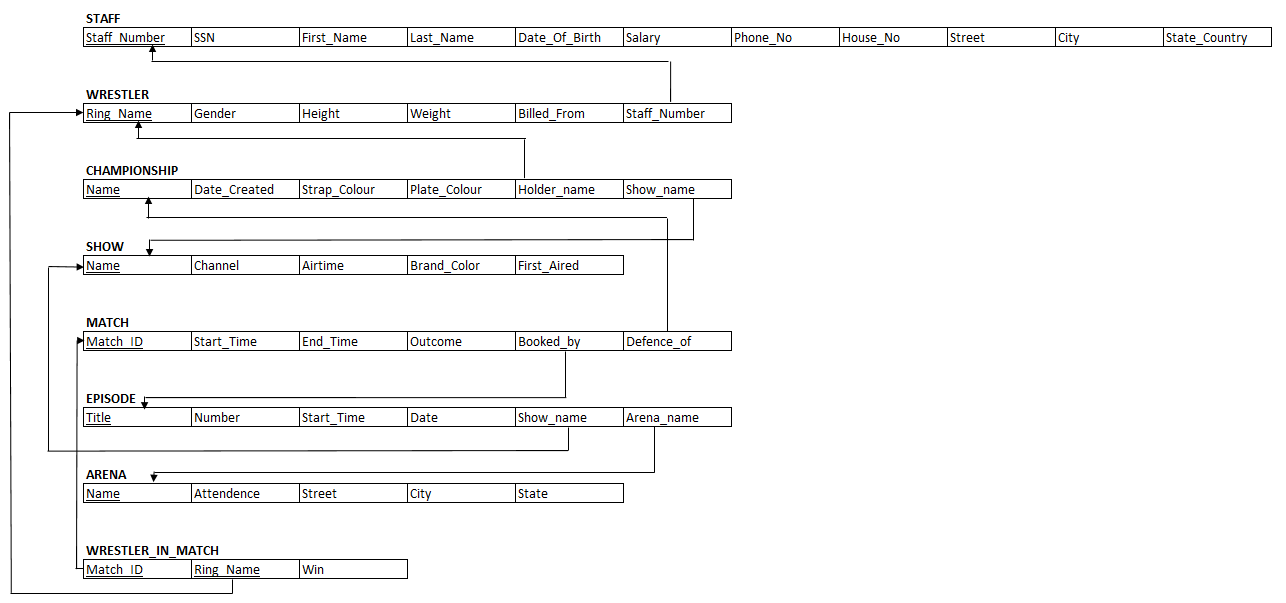
# Relational Schema

The relational schema was translated from the entity relationship model as shown in lectures. Each entity type became a table, and each attribute became an attribute of that table. For each entity type, one of the key attributes was chosen as the primary key. In most cases there was only one key attribute to choose from, except in STAFF where there was both Staff\_Number and SSN. After applying real-world semantics the Staff\_Number seemed like the more worthy primary key, as it would be more often used to identify a staff member within the business, rather than SSN which would be more related to the person’s government identity.

There was only one 1:1 relationship; STAFF:WRESTLER. This could be mapped using the foreign-key approach or the merged relation approach. However, as there was not total participation on both end of this relationship, merging the tuples would lead to many NULL columns where a staff member was not a wrestler. As a result the foreign key approach was used. As there was total participation on the WRESTLER side, the foreign key was put in WRESTLER which pointed to STAFF.

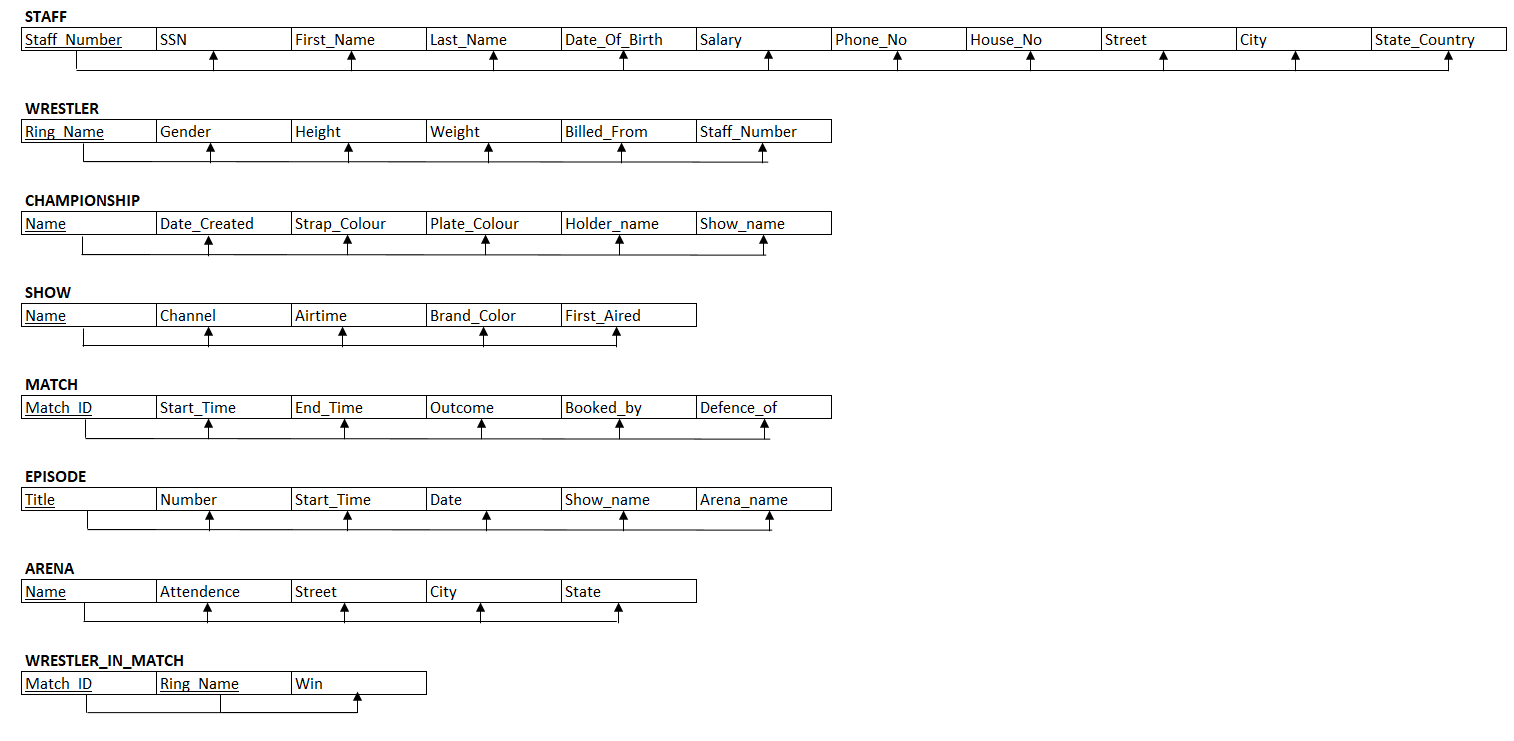
The database model is mostly comprised of 1:N relationships. These were all mapped by putting a foreign key on the N side, pointing to the primary key of the 1 side. This occurs multiple times in the model; one example being that there are N episodes of 1 show, so EPISODE has an attribute Show\_name, which is a foreign key that points to SHOW.

There is one M:N relationship between WRESTLER and MATCH. For this, a new table WRESTLER\_IN\_MATCH was created. The primary key was a composite primary key made up of the foreign keys to WRESTLER and MATCH. An attribute of the relationship, Win, was also put into the table.



# Functional Dependency Diagram and Normalization

Due to the way in which the database was designed; i.e. an entity relationship model then translated into a relational schema using the rules above, the full design is in Boyce-Codd normal form. For every relation R, all attributes in R are *dependent on the key, the whole key, and nothing but the key*. We can see this more clearly by mapping out the functional dependencies as below. Each primary key fully determines the other attributes in the tuple. This eliminates redundancy and avoids unintended data loss when a tuple is modified/deleted.



# Semantic Constraints

### Table Constraints

Some of the table constraints were chosen based on real world semantics. A good example of this is the STAFF relation. *Staff\_Number* is a primary key, so it will be unique and not null. *SSN* is also unique and not null, as in the database every staff member should have one for administration purposes, and duplicate SSNs would mean that there is some administration error. The *First\_Name*, *Last\_Name* and *Date\_Of\_Birth* fields should not be null as these are fundamental, basic pieces of information about the staff member, so there is no excuse for them to be null. Another constraint we see is a check that *Salary* is greater than zero, as it makes no excuse to have a salary of zero or less. If there is no salary for a staff member for any reason, null is allowed. The constraints on *Staff\_Number*, *SSN*, *First\_Name*, *Last\_Name* and *Salary* are stated in this CREATE TABLE command:

CREATE TABLE STAFF (

    Staff\_Number INTEGER NOT NULL,

    SSN INTEGER NOT NULL UNIQUE,

    First\_Name VARCHAR(50) NOT NULL,

    Last\_Name VARCHAR(50) NOT NULL,

    Date\_Of\_Birth DATE NOT NULL,

    Salary INTEGER,

    Phone\_No CHAR(10),

    House\_No VARCHAR(20),

    Street VARCHAR(20),

    City VARCHAR(20),

    State\_Country VARCHAR(20),

    PRIMARY KEY (Staff\_Number),

    CHECK (Salary > 0)

);

The participation constraints from the entity relationship model carried over to the creation of the tables also. For example, in the diagram we can see that a wrestler cannot exist without a corresponding staff member. So in the WRESTLER table, the foreign key to STAFF is NOT NULL. If the primary key of STAFF is updated (a staff member gets a new staff number), the update to cascade to the foreign key from WRESTLER, so this is specified when WRESTLER is created. A staff member should not be deleted if a wrestler instance still refers to them, so there is no ON DELETE statement. A design decision was made to use this approach (not null foreign keys which update on cascade) whenever mapping from participation constraints in the entity relationship model.

CREATE TABLE WRESTLER (

    …

    Staff\_Number INTEGER NOT NULL,

    …

    FOREIGN KEY (Staff\_Number) REFERENCES STAFF (Staff\_Number) ON UPDATE

CASCADE,

    …

);

The approach to constraints seen above (primary keys, real-life semantics and mapping from participation constraints) was taken when applying table constraints to any of the tables.

### Triggers

Due to design decisions, most of the constraints implemented in the database were table constraints. It seemed clearer to have the constraints built into the table definition rather than a separate trigger on modification of a table. This is the reason that there are many constraints built into the CREATE statements above. However, another interesting use for triggers that was used was placing default values. I found this best to separate from the CREATE TABLE statements as ideally when a tuple is created, all known values will be specified. However, if either the creation date of a championship or the air date of a championship are left NULL, it would be reasonable to say that today’s date would be an expected value. So on both CHAMPIONSHIP and EPISODE, there are triggers which check if the date is NULL; if so, set the date to today’s date.

CREATE TRIGGER Championship\_Date

AFTER INSERT ON CHAMPIONSHIP

WHEN (new.Date\_Created IS NULL)

DECLARE

    Championship\_Name VARCHAR(50);

BEGIN

    Name := NEW.Championship\_Name;

    UPDATE CHAMPIONSHIP

    SET Date\_Created = CURRENT\_DATE

    WHERE Name = Championship\_Name;

END;

.

RUN;

CREATE TRIGGER Episode\_Date

AFTER INSERT ON EPISODE

WHEN (new.Date IS NULL)

DECLARE

    Episode\_Title VARCHAR(255);

BEGIN

    Episode\_Title := NEW.Title;

    UPDATE EPISODE

    SET Date = CURRENT\_DATE

    WHERE Title = Episode\_Title;

END;

.

RUN;

Separating the more important constraints from dealing with default values breaks the code up and makes it easier to understand. Variables are also used in these triggers to make the code more readable; in this case they represent the name of the new championship or title of new episode after insert respectively.

# Creating a View

Views are very useful mechanisms for SQL statements which may be repeated often, allowing users to query the view rather than repeating the more complicated statement again. They may also be used for security purposes to restrict the data that a user is shown. In this project I created three views on the database.

The first view is called PAYROLL, and can be interpreted as payroll information about a staff member, but without their personal information. The view contains three columns from STAFF, which are the staff number, SSN and salary. This view allows payroll information to be queried for a staff member without identifying their name or other identifying information like their address.

CREATE VIEW PAYROLL (

    Staff\_No, Staff\_SSN, Staff\_Salary)

    AS SELECT Staff\_Number, SSN, Salary

    FROM STAFF;

The second view allows the personal information of a wrestler to be achieved by their ringname. This may be useful in the company if somebody wants to find out a wrestler’s contact information via their ring name rather than their real name or staff number. As many wrestlers are known by their ring names, this is a query that would be relatively common. This view does not contain any of the “payroll” columns above as these could be seen as more sensitive information and not information that should be queried by ring name.

CREATE VIEW WRESTLER\_INFORMATION (

    Wrestler\_Ring\_Name, Wrestler\_Gender, Wrestler\_First\_Name,

    Wrestler\_Last\_Name, Wrestler\_Date\_Of\_Birth, Wrestler\_Phone\_No,

    Wrestler\_House\_No, Wrestler\_Street, Wrestler\_City,

    Wrestler\_State\_Country)

    AS SELECT WRESTLER.Ring\_Name, WRESTLER.Gender, STAFF.First\_Name,

STAFF.Last\_Name, STAFF.Date\_Of\_Birth, STAFF.Phone\_No, STAFF.House\_No,

STAFF.Street, STAFF.City, STAFF.State\_Country

    FROM WRESTLER, STAFF

    WHERE WRESTLER.Ring\_Name = STAFF.Ring\_Name;

Important information in a professional wrestling company is the history of title (championship) wins since the company started. The TITLE\_WINS view shows the winner for every match where a championship was on the line. This brings back the wrestler’s ring name, the championship name and the match ID, to allow the database to do further querying. The result is a history of all title wins in matches in WWE history.

CREATE VIEW TITLE\_WINS (

    TITLE\_WIN\_WRESTLER, TITLE\_WIN\_CHAMPIONSHIP, TITLE\_WIN\_MATCH)

    AS SELECT WRESTLER\_IN\_MATCH.Ring\_Name, MATCH.Defence\_Of,

WRESTLER\_IN\_MATCH.Match\_ID

    FROM WRESTLER\_IN\_MATCH, MATCH

    WHERE WRESTLER\_IN\_MATCH.Match\_ID = Match\_ID

AND MATCH.Defence\_Of IS NOT NULL AND WRESTLER\_IN\_MATCH.Win = 1;

# Appendix

### Table Creation

CREATE TABLE STAFF (

    Staff\_Number INTEGER NOT NULL,

    SSN INTEGER NOT NULL UNIQUE,

    First\_Name VARCHAR(50) NOT NULL,

    Last\_Name VARCHAR(50) NOT NULL,

    Date\_Of\_Birth DATE NOT NULL,

    Salary INTEGER,

    Phone\_No CHAR(10),

    House\_No VARCHAR(20),

    Street VARCHAR(20),

    City VARCHAR(20),

    State\_Country VARCHAR(20),

    PRIMARY KEY (Staff\_Number),

    CHECK (Salary > 0));

CREATE TABLE WRESTLER (

    Ring\_Name VARCHAR(50) NOT NULL,

    Gender CHAR(1),

    Height INTEGER,

    Weight INTEGER,

    Billed\_From VARCHAR(50),

    Staff\_Number INTEGER NOT NULL,

    PRIMARY KEY (Ring\_Name),

    FOREIGN KEY (Staff\_Number) REFERENCES STAFF (Staff\_Number)

ON UPDATE CASCADE,

    CONSTRAINT CHK\_Wrestler CHECK (Height > 0 AND Weight > 0));

CREATE TABLE SHOW (

    Name VARCHAR(20) NOT NULL,

    Channel VARCHAR(20),

    Airtime INTEGER,

    Brand\_Color INTEGER,

    First\_Aired DATE,

    PRIMARY KEY (Name));

CREATE TABLE CHAMPIONSHIP (

    Name VARCHAR(50) NOT NULL,

    Date\_Created DATE,

    Strap\_Colour INTEGER,

    Plate\_Colour INTEGER,

    Holder\_Name VARCHAR(50) NOT NULL,

    Show\_Name VARCHAR(20) NOT NULL,

    PRIMARY KEY (Name),

    FOREIGN KEY (Holder\_Name) REFERENCES WRESTLER (Ring\_Name)

ON UPDATE CASCADE,

    FOREIGN KEY (Show\_Name) REFERENCES SHOW (Name) ON UPDATE CASCADE

);

CREATE TABLE ARENA (

    Name VARCHAR(255) NOT NULL,

    Attendence INTEGER,

    Street VARCHAR(20),

    City VARCHAR(20),

    State VARCHAR(20),

    PRIMARY KEY (Name),

    CHECK (Attendence > 0));

CREATE TABLE EPISODE(

    Title VARCHAR(255) NOT NULL,

    Number INTEGER,

    Start\_Time TIME(0),

    End\_Time TIME(0),

    Date DATE,

    Show\_Name VARCHAR(20) NOT NULL,

    Arena\_Name VARCHAR(255) NOT NULL,

    PRIMARY KEY (Title),

    FOREIGN KEY (Show\_Name) REFERENCES SHOW (Name) ON UPDATE CASCADE,

    FOREIGN KEY (Arena\_Name) REFERENCES ARENA (Name) ON UPDATE CASCADE,

    CHECK (Number >= 0));

CREATE TABLE MATCH (

    Match\_ID INTEGER NOT NULL,

    Start\_Time TIME(0),

    End\_Time TIME(0),

    Outcome VARCHAR(20),

    Booked\_By VARCHAR(255) NOT NULL,

    Defence\_Of VARCHAR(50),

    PRIMARY KEY (Match\_ID),

    FOREIGN KEY (Booked\_By) REFERENCES EPISODE (Title) ON UPDATE CASCADE,

    FOREIGN KEY (Defence\_Of) REFERENCES CHAMPIONSHIP (Name)

ON UPDATE CASCADE);

CREATE TABLE WRESTLER\_IN\_MATCH (

    Match\_ID INTEGER NOT NULL,

    Ring\_Name VARCHAR(50) NOT NULL,

    Win BIT(1),

    PRIMARY KEY (Match\_ID, Ring\_Name),

    FOREIGN KEY (Match\_ID) REFERENCES MATCH (Match\_ID) ON UPDATE CASCADE,

    FOREIGN KEY (Ring\_Name) REFERENCES WRESTLER (Ring\_Name)

ON UPDATE CASCADE);

### Triggers

CREATE TRIGGER Championship\_Date

AFTER INSERT ON CHAMPIONSHIP

WHEN (new.Date\_Created IS NULL)

DECLARE

    Championship\_Name VARCHAR(50);

BEGIN

    Name := NEW.Championship\_Name;

    UPDATE CHAMPIONSHIP

    SET Date\_Created = CURRENT\_DATE

    WHERE Name = Championship\_Name;

END;

.

RUN;

CREATE TRIGGER Episode\_Date

AFTER INSERT ON EPISODE

WHEN (new.Date IS NULL)

DECLARE

    Episode\_Title VARCHAR(255);

BEGIN

    Episode\_Title := NEW.Title;

    UPDATE EPISODE

    SET Date = CURRENT\_DATE

    WHERE Title = Episode\_Title;

END;

.

RUN;

### Views

CREATE VIEW PAYROLL (

    Staff\_No, Staff\_SSN, Staff\_Salary)

    AS SELECT Staff\_Number, SSN, Salary

    FROM STAFF;

CREATE VIEW WRESTLER\_INFORMATION (

    Wrestler\_Ring\_Name, Wrestler\_Gender, Wrestler\_First\_Name,

    Wrestler\_Last\_Name, Wrestler\_Date\_Of\_Birth, Wrestler\_Phone\_No,

    Wrestler\_House\_No, Wrestler\_Street, Wrestler\_City, Wrestler\_State\_Country)

    AS SELECT WRESTLER.Ring\_Name, WRESTLER.Gender, STAFF.First\_Name,

STAFF.Last\_Name, STAFF.Date\_Of\_Birth, STAFF.Phone\_No, STAFF.House\_No,

STAFF.Street, STAFF.City, STAFF.State\_Country

    FROM WRESTLER, STAFF

    WHERE WRESTLER.Ring\_Name = STAFF.Ring\_Name;

CREATE VIEW TITLE\_WINS (

    TITLE\_WIN\_WRESTLER, TITLE\_WIN\_CHAMPIONSHIP, TITLE\_WIN\_MATCH)

    AS SELECT WRESTLER\_IN\_MATCH.Ring\_Name, MATCH.Defence\_Of,

WRESTLER\_IN\_MATCH.Match\_ID

    FROM WRESTLER\_IN\_MATCH, MATCH

    WHERE WRESTLER\_IN\_MATCH.Match\_ID = Match\_ID

AND MATCH.Defence\_Of IS NOT NULL AND WRESTLER\_IN\_MATCH.Win = 1;