1. Preventing any null values in your dataset or prevent the user to skip one information not to be filled in the table create table promotions. Use NOT NULL



create table promotion

(

id int,

name varchar (100) not null,

category varchar (15)

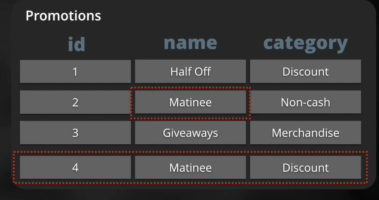
);

Constraints can help with these shortcomings

* Prevent NULL values
* Ensure column values are unique
* Provide additional validations

1. Preventing any unwanted duplicates in the name attribute column. Use UNIQUE sql constraints.

This is called column constraint

create table promotion

(

id int,

name varchar (100) not null unique,

category varchar (15)

);

1. Defining error message for easier trouble shooting

when duplicate values or text were placed in the

table. use constraint unique\_”title “ unique

(column1, column2 etc)

This is called table constraint

create table promotion

(

id int,

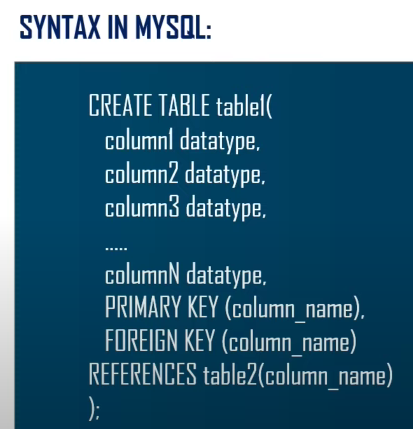
name varchar (100) not null ,

category varchar (15),

constraint unique\_name unique (name, category)

);

1. Preventing ID column to be unique and not null. Use Primary Key sql constraints



create table promotion

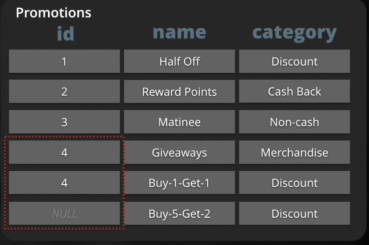
(

id int primary key, or

name varchar (100) not null ,

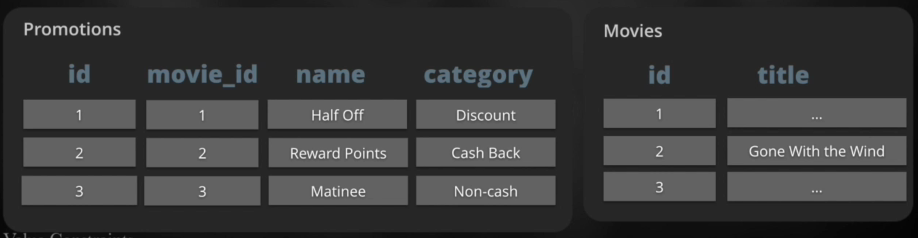
category varchar (15),

constraint unique\_name unique (name, category)

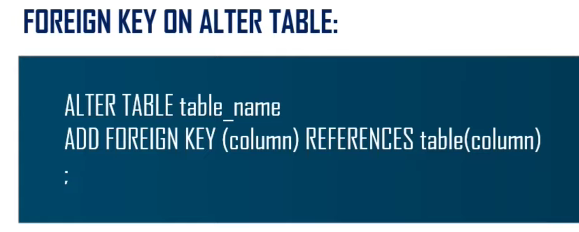
);

1. Creating valid references between 2 tables. Use FOREIGN KEY sql constraints

* REFERENCES keyword can be used to make FOREIGN KEY constraint
* The table being referenced must be created first



create table movies

(

id int primary key

title varchar(100) not null unique

);

create table promotions

(

id int primary key,

movie\_id int references movies (id),

name varchar(50) not null unique,

category varchar (15) not null

);

or

create table promotions

(

id int primary key,

movie\_id int references movies ,

name varchar(50) not null unique,

category varchar (15) not null

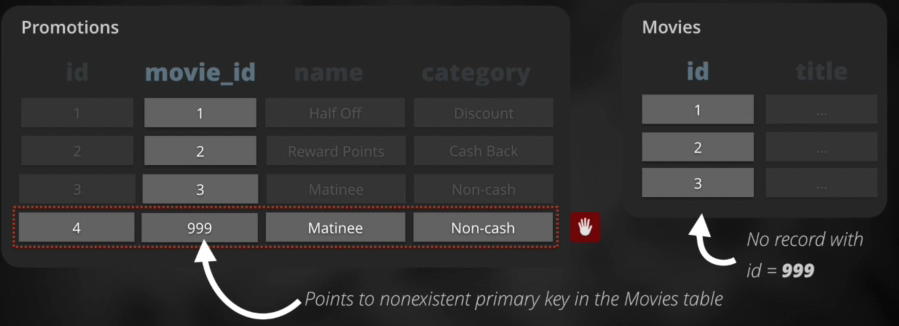
);

1. Prevent invalid mismatch of data between tables when registering or insert new values. Use FOREIGN Key constraint.

Creating valid references between 2 tables. Use FOREIGN KEY sql constraints

• REFERENCES keyword can be used to make FOREIGN KEY constraint

• The table being referenced must be created first



create table movies

(

id int primary key

title varchar(100) not null unique

);

create table promotions

(

id int primary key,

movie\_id int references movies

name varchar(50) not null unique

category varchar (15) not null

);

1. Prevent Orphaned records Use FOREIGN KEY constraints.



create table movies

(

id int primary key

title varchar(100) not null unique

);

create table promotions

(

id int primary key,

movie\_id int references movies

name varchar(50) not null unique

category varchar (15) not null

);

* If you want to delete an orphaned record . Delete the original table first, then delete the orphaned or referenced table record.

delete from promotions where movie\_id=6;

delete from movies where id=6;

* if you want to drop a table. Drop the original table first and then drop the reference secondary table.

delete table promotions;

delete table movies;

1. Validating data insertion

We want to make sure that lets say the duration for each movie has no negative values.

Use CHECK constraints



create table movies

(

id int primary key

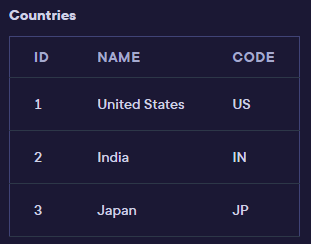
title varchar(100) not null unique,

genre varchar(100),

duration int check (duration>0)

);

------------------------------------------------------------------------------------------------------------------------------------------------------------



1. Adding a foreign key to the actors table that references data

in the countries table.

create table actors

(

id int primary key,

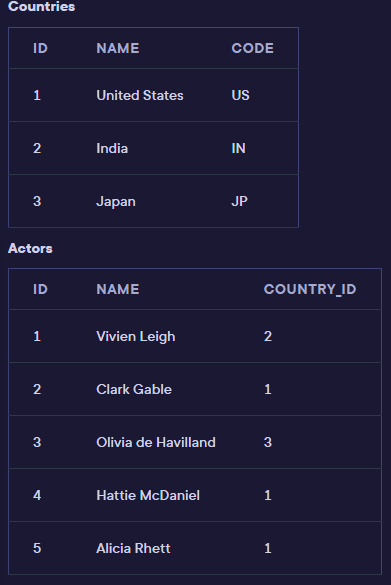
name varchar(50) not null unique,

country\_id int references countries (id)

);

1. convert the current FOREIGN KEY column constraint to

use the table constraint syntax.

1.  Some of our actors have been way underpaid.

Add a CHECK constraint to prevent values less than

500 from being assigned to the salary column.

create table actors

(

id int primary key,

name varchar(50) not null unique,

country\_id int references countries (id),

salary int check (salary>=500 )

);

1. On top of their salaries, actors will start receiving

bonuses. However, values for bonus need to be

less than values for salary. Add a CHECK column

constraint to enforce this condition on the bonus

column.

create table actors

(

id int primary key,

name varchar(50) not null unique,

country\_id int references countries (id),

salary int check (salary>=500 ),

bonus int check (bonus< salary)

);

--------------------------------------------------------------------------------------------------------------------------------------------------------

NORMALIZATION

First normal form rule:

Tables must not contain repeating groups of data in 1 column. The table shows there are two adventure genres.



Second normal form rule:

Tables must not contain redundancy ( unnecessary repeating information)



To correct it

|  |  |  |
| --- | --- | --- |
| **MOVIES** | | |
| **id** | **title** | **duration** |
| 1 | Don Juan | 110 |
| 2 | Peter Pan | 120 |
| 3 | The Lost World | 105 |
| 4 | Robin Hood | 143 |

|  |  |
| --- | --- |
| **GENRES** | |
| **Id** | **genre** |
| 1 | Romance |
| 2 | Adventure |
| 3 | Fantasy |
|  |  |



|  |  |
| --- | --- |
| **MOVIES\_GENRES** | |
| **movie\_id** | **genre\_id** |
| 1 | 1 |
| 2 | 2 |
| 2 | 3 |
| 3 | 3 |
| 4 | 2 |

now it is easier to do the following tasks.

* update a movie duration

update movies

set duration=134

where id=’2’;

* to add genre to a movie

insert into movies\_genres ( movie\_id, genre\_id)

values (4,3);

* find the genres of peter pan

select id from movies

where title= ‘Peter Pan”;

select genre\_id

from movies\_genres

where movie\_id=2;

select name

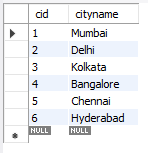
from genres

where id=2 or id=3; or where id in (2,3);

------------------------------------------------------------------------------------------------------------------------------------------

create database sales;

use sales;



create table city

(

cid int not null auto\_increment,

cityname varchar(50) not null,

primary key (cid)

);

insert into city (cityname)

values

('Mumbai'),

('Delhi'),

('Kolkata'),

('Bangalore'),

('Chennai'),

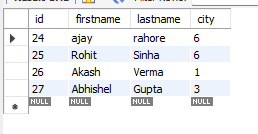
('Hyderabad')

;

create table customers (

id int not null auto\_increment,

firstname varchar(50) not null,

lastname varchar (50) not null,

city int not null,

primary key (id),

foreign key (city) references city(cid)--- it is always column1 table1 column 1

);

insert into customers (id, firstname, lastname, city)

values

(24,'ajay','rahore' ,'6'),

(25, 'Rohit', 'Sinha' ,'6'),

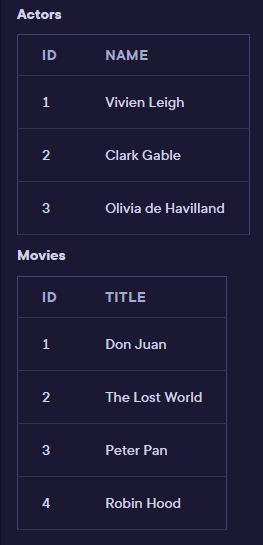
(26, 'Akash', 'Verma','1' ),

(27, 'Abhishel','Gupta' ,'3' );

\*/

create table definition by adding the proper

foreign keys.



First, let's add the new columns. The new columns we

add should have type int to match the data they will hold.

Per convention, let's name these columns actor\_id and

movie\_id.

create table actors

(

actors\_id int not null,

actors\_name varchar(150) not null,

primary key (actors\_id)

);

create table movies

(

movies\_id int not null,

movies\_title varchar(150) not null,

primary key (movies\_id)

);

create table Actor\_Movies

(

foreign key (movies\_id) references movies (movies\_id),

foreign key (actors\_id) references actors (actors\_id)

);

1. Now, let's add FOREIGN KEY constraints to both of these

columns, as column constraints, referencing the appropriate tables.

1. A new movie just came out, The Wolfman, starring our favorite actor,

Clark Gable. Both the movie and the actor are in our database,

but now they need an association created.

Update the two NULL values in the INSERT statement to the

appropriate actor\_id and movie\_id.