Primary keys and foreign keys are essential concepts in relational databases and are used for maintaining data integrity and establishing relationships between tables. Here's where you necessarily use primary keys and foreign keys:

1. Primary Key (PK):

- Each table should have a primary key column that uniquely identifies each row (record) in the table.

- The primary key ensures that there are no duplicate rows in the table.

- The primary key provides a fast way to locate specific rows in the table when used in queries or joins.

- You should use the primary key when you need a unique identifier for each row in the table.

2. Foreign Key (FK):

- A foreign key is a column or set of columns in one table that refers to the primary key in another table, establishing a relationship between the two tables.

- It enforces referential integrity, ensuring that values in the foreign key column must match values in the primary key column of the referenced table or be NULL.

- The foreign key is used to create relationships between tables, such as one-to-many or many-to-many relationships.

- You should use foreign keys when you need to establish and enforce relationships between different tables in the database.

For example, consider the following two tables:

Customers:

- customer\_id (primary key)

- customer\_name

- customer\_email

Orders:

- order\_id (primary key)

- customer\_id (foreign key referencing Customers)

- order\_date

- order\_amount

In this example, the "customer\_id" in the "Orders" table is a foreign key that references the primary key "customer\_id" in the "Customers" table. This foreign key relationship links each order to a specific customer.

Using primary keys and foreign keys helps maintain data integrity, prevents orphaned or inconsistent data, and allows for efficient querying and joining of related data in a relational database. It's an essential practice when designing and managing relational databases.

you do not need to explicitly specify your foreign keys and primary keys in order to use joins in SQL. Joins are used to combine rows from two or more tables based on related columns between them. While foreign keys and primary keys help define the relationships between tables, they are not mandatory for performing joins.

Joins are primarily based on the common columns between tables, and the most common join is the INNER JOIN, where only the matching rows between tables are returned.

Here's an example of how to perform a join between two tables without explicitly specifying foreign keys or primary keys:

Let's say you have two tables, "Orders" and "Customers," with the following structures:

Orders:

order\_id (primary key)

customer\_id

order\_date

other\_order\_columns...

Customers:

customer\_id (primary key)

customer\_name

customer\_email

other\_customer\_columns...

You can perform an INNER JOIN between the "Orders" and "Customers" tables based on the common column "customer\_id" as follows:

SELECT Orders.order\_id, Orders.order\_date, Customers.customer\_name

FROM Orders

INNER JOIN Customers ON Orders.customer\_id = Customers.customer\_id;

In this query, we are joining the "Orders" table with the "Customers" table on the "customer\_id" column. The query will return the "order\_id," "order\_date," and "customer\_name" for each matching row where the "customer\_id" in "Orders" matches the "customer\_id" in "Customers."

So, while foreign keys and primary keys are useful for maintaining data integrity and enforcing relationships between tables, they are not a requirement for performing joins in SQL. Joins are based on the common columns used to connect the data between tables, whether or not foreign keys and primary keys are explicitly defined.

To define a primary key in a table, you need to specify which column or set of columns uniquely identifies each row in the table. In most database management systems (DBMS), you can define the primary key constraint during the table creation or add it later using the ALTER TABLE statement.

Here are two ways to define a primary key:

Defining Primary Key During Table Creation:

When creating a new table, you can define the primary key as part of the table definition.

Here's the syntax:

CREATE TABLE table\_name (

column1 data\_type PRIMARY KEY,

column2 data\_type,

...);

In this syntax:

table\_name is the name of your table.

column1 is the column that you want to set as the primary key.

data\_type is the data type of the column1.

For example, let's say you have a "Customers" table and you want to set the "customer\_id" column as the primary key:

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY,

customer\_name VARCHAR(50),

customer\_email VARCHAR(100)

);

Adding Primary Key After Table Creation:

If your table is already created and you want to add a primary key, you can use the ALTER TABLE statement:

ALTER TABLE table\_name

ADD PRIMARY KEY (column1);

For example, to add a primary key to the "Customers" table for the "customer\_id" column:

ALTER TABLE Customers

ADD PRIMARY KEY (customer\_id);

Please note that a primary key must have unique and non-null values. So, ensure that the column you choose as the primary key satisfies these requirements. Also, a table can have only one primary key, and it cannot be null.

To define foreign keys in a database table, you need to specify which column(s) in your table will reference the primary key column(s) of another table. Foreign keys establish relationships between tables, ensuring that the values in the referencing column(s) must match the values in the referenced primary key column(s) or be NULL. This maintains referential integrity and enforces data consistency.

There are two ways to define foreign keys:

1. Defining Foreign Keys During Table Creation:

When creating a new table, you can define the foreign key constraints as part of the table definition. Here's the syntax:

CREATE TABLE table\_name (

column1 data\_type,

column2 data\_type,

foreign\_key\_column data\_type,

...

FOREIGN KEY (foreign\_key\_column) REFERENCES referenced\_table(referenced\_column)

);

In this syntax:

- `table\_name` is the name of your table.

- `column1`, `column2`, etc., are the columns in your table.

- `foreign\_key\_column` is the column in your table that will reference the primary key column in another table.

- `referenced\_table` is the name of the table you are referencing.

- `referenced\_column` is the primary key column in the referenced table.

For example, let's say you have an "Orders" table with a "customer\_id" column that references the "customer\_id" column in the "Customers" table:

CREATE TABLE Orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_date DATE,

-- other columns...

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

2. Adding Foreign Keys After Table Creation:

If your table is already created, and you want to add a foreign key constraint, you can use the `ALTER TABLE` statement:

ALTER TABLE table\_name

ADD FOREIGN KEY (foreign\_key\_column) REFERENCES referenced\_table(referenced\_column);

For example, to add a foreign key to the "Orders" table for the "customer\_id" column:

ALTER TABLE Orders

ADD FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id);

Please note that both the referenced table and the referencing table must exist in the database with the appropriate primary key column and data type before you can define foreign keys. Also, ensure that the data types of the referenced and referencing columns match for the foreign key constraint to work correctly.

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)

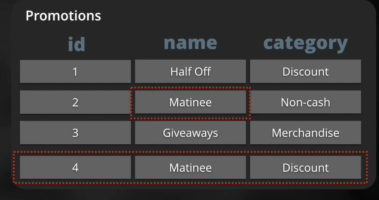
);

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,

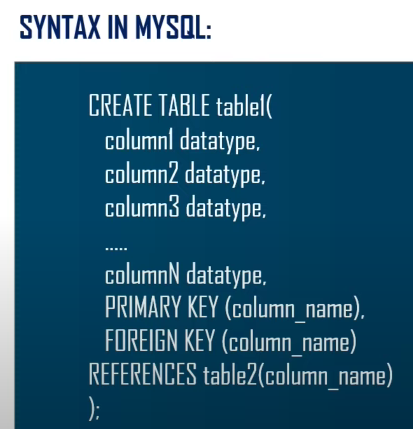
name varchar (100) not null ,

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

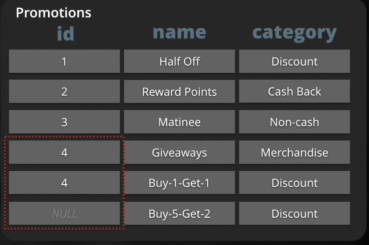
(

id int primary key, or

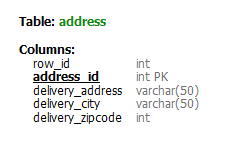
name varchar (100) not null ,

category varchar (15),

constraint unique\_name unique (name, category)

);

1. I noticed that pk or primary key is assigned automatically whether I put a row\_id or no. Thus when creating table in mysql workbench. There is no need to type primary key as it automatically assigns it.



create table Address

(

row\_id int,

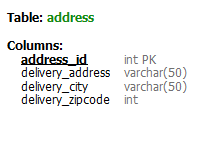
address\_id int not null unique,

delivery\_address varchar (50) null,

delivery\_city varchar(50) not null,

delivery\_zipcode int not null

);

use Altors\_Pizzeria;

create table Address

(

address\_id int not null unique,

delivery\_address varchar (50) null,

delivery\_city varchar(50) not null,

delivery\_zipcode int not null

);

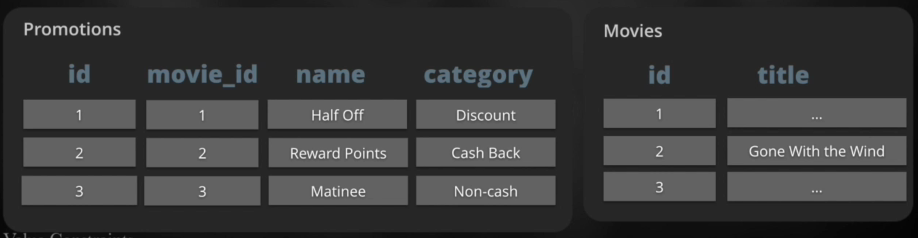
1. But if you want to add pk in the table you have already created . Do this

Alter table Address

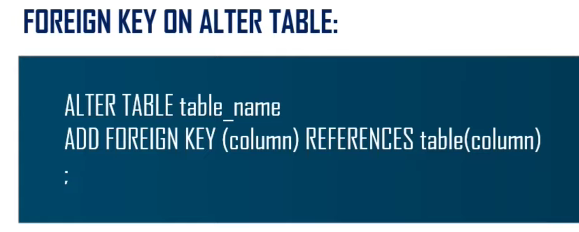
add primary key (address\_id);

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

);

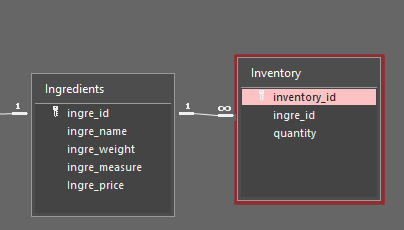
1. You can assign the foreign key

Please note that both the referenced table and the referencing table must exist in the database with the appropriate primary key column and data type before you can define foreign keys. Also, ensure that the data types of the referenced and referencing columns match for the foreign key constraint to work correctly.

if you forgot to add your foreign key you can use the alter table table or just drop the table and create another one with references on it

create table Ingredients

(

ingre\_id VARCHAR (50) not null unique,

int\_name varchar(50),

ingre\_measure varchar(20) not null,

ingre\_price decimal (10,2)

);

create table Inventory

(

inv\_id int not null unique,

ingre\_id varchar(50) not null REFERENCES INGREDIENTS,

quantity int not null not null,

);

or

create table Inventory

(

inv\_id int not null unique,

ingre\_id varchar(50) not null,

quantity int not null not null,

CONSTRAINT FOREIGN KEY (ingre\_id) REFERENCES Ingredients(ingre\_id)

);

or

create table Inventory

(

inv\_id int not null unique,

ingre\_id varchar(50) not null,

quantity int not null not null,

);

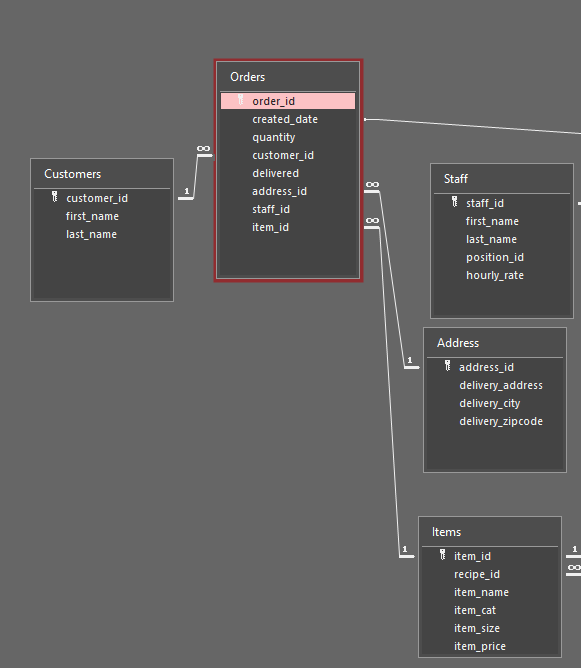
alter table Inventory

ADD CONSTRAINT FK\_Orders\_Inventory

FOREIGN KEY (ingre\_id) REFERENCES Inventory(ingre\_id),

1. Lets say you have already created a table but forgot to assign your 1 primary and 3 foreign keys

NOTE: INFINITE Means Foreign key in here. 1 means Primary Key

create table Orders

(

order\_id int not null,

created\_date datetime,

quantity int not null,

customer\_id int not null,

delivery int not null,

address\_id int not null,

item\_id int not null

);

alter table Orders

add primary key (order\_id),

ADD CONSTRAINT FK\_Orders\_Customers

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

ADD CONSTRAINT FK\_Orders\_Address

FOREIGN KEY (address\_id) REFERENCES Address(address\_id),

ADD CONSTRAINT FK\_Orders\_Items

FOREIGN KEY (item\_id) REFERENCES Items(item\_id);

BUT If you want to include them immediately in your code

CREATE TABLE Orders (

order\_id int PRIMARY KEY,

created\_date datetime,

quantity int not null,

customer\_id int not null,

delivery int not null,

address\_id int,

item\_id int,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id),

FOREIGN KEY (address\_id) REFERENCES Address(address\_id),

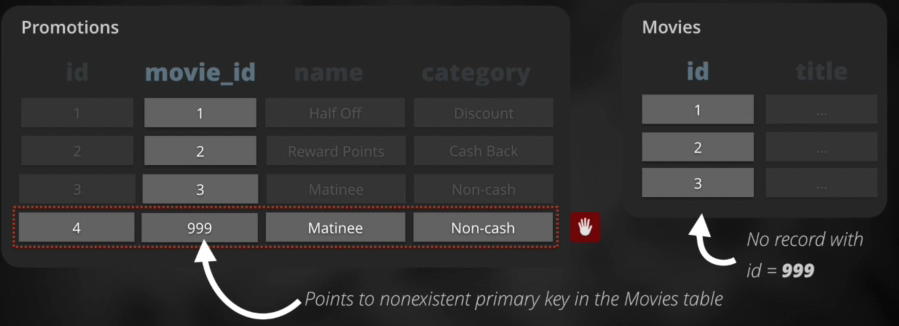
FOREIGN KEY (item\_id) REFERENCES Items(item\_id)

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

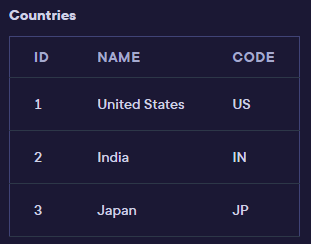
title varchar(100) not null unique,

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,

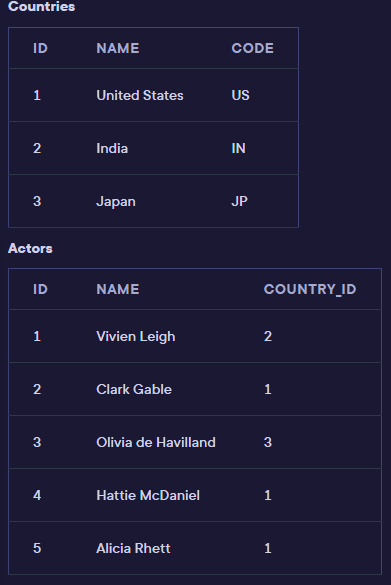
name varchar(50) not null unique,

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

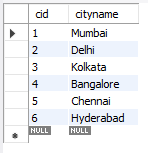
from genres

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'),

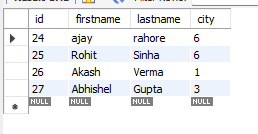
('Hyderabad')

;

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'),

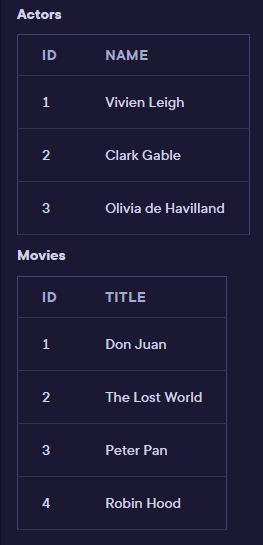
(26, 'Akash', 'Verma','1' ),

(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.