2021-09-29T18:00:00+02:00
Martyna Sławińska

Why Do Relational Databases Use Primary Keys and Foreign Keys?

Relational databases organize data in a standardized manner. All data is entered into tables, the so-called relations. The tables are linked with one another to create connections between the data. The backbone of any relational database is the primary and foreign keys. Read along to find out why they are so important.

In this article, we focus on the reasons why relational databases use primary and foreign keys. First, we review what primary and foreign keys are. Next, we do a deep dive into why primary and foreign keys are used in relational databases, followed by some examples. We then discuss the drawbacks of using primary and foreign keys. We close the article by mentioning some engines that do not enforce the usage of primary and foreign keys.

Do you know much about database management systems? Here is something interesting.

Let's get started!

Let's Review: Primary and Foreign Keys in Relational Databases

Primary and foreign keys help create structure in relational databases.

A primary key ensures unique row identification. This results in faster sorting, searching, and querying operations.

A foreign key creates a link between two tables. It maintains referential integrity between the referencing column(s) and the referenced

column(s).

LearnSQL.com is an online platform designed to help you master SQL. It offers 30 interactive courses that range in difficulty from beginner to advanced. Each course delivers both theoretical knowledge and hands-on exercises so that you can solidify these new ideas.

What Is a Primary Key?

A primary key is a column or a set of columns that uniquely identifies each row in a table. It must obey the `UNIQUE` and `NOT NULL` database constraints.

Let's look at the **`Stock`** table, which stores information about the products in stock.

Stock		
ProductId	int	PK
QuantityAvailable	int	
StoredIn	varchar(20)	

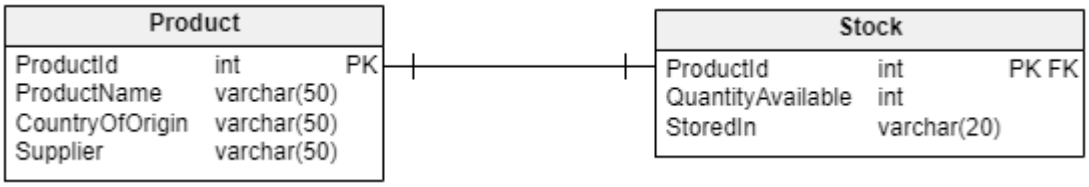
The `ProductId` column is the primary key of the **`Stock`** table. The column uniquely identifies each row in the **`Stock`** table; it stores unique values and does not store `NULL` values.

Check out our article "What is a Primary Key in SQL?" to learn more about primary keys.

What Is a Foreign Key?

A foreign key is a column or a set of columns linking one table to another. The column or columns appear in both tables, creating a link between the tables.

Let's use our **`Stock`** table together with the **`Product`** table, which stores details about each product.



The `ProductId` column is the primary key of the **`Product`** table. It is also the primary and foreign key in the **`Stock`** table. The **`Product`** and **`Stock`** tables are linked via the `ProductId` column in a one-to-one relationship; each product in the **`Product`** table must have exactly one entry in the **`Stock`** table.

Check out our article "What Is a Foreign Key in SQL?" to learn more about foreign keys. You can also take a look at these cookbooks:

> How to Create a Table with a Foreign Key in SQL.
>
> How to Delete a Foreign Key Constraint in SQL.

Reasons for Using Relational Database Keys

There are several reasons for using relational database keys. This section presents the benefits of using them in your database.

Why Relational Databases Use Primary Keys

Primary keys allow unique row identification. And the rest follows from that.

Unique Row Identification

The primary key ensures each row in a table has a unique identifier. There are two types of primary keys:

> Natural primary key: a column that stores relevant data and at the same time uniquely identifies each row.
>
> Surrogate primary key: a column that is added to the table only to identify each row uniquely.

Let's look at the **`Customer`** table, which stores customer data.

CustomerId	FirstName	LastName	Email

CustomerId	FirstName	LastName	Email
432	Grace	Johns	grace@email.com
433	Robert	Black	robert.black@email.com
434	Alice	Kelley	a.kelley@email.com

Here, the `CustomerId` column is a surrogate primary key. It uniquely identifies each row but does not store any relevant data. The `Email` column may be used as a natural primary key if each customer provides a unique email address.

Uniquely identifying rows is important because it enables:

- Performing efficient row update and row removal operations.
- Improving the performance of sorting, searching, and querying operations.
- Creating referential integrity between tables.

These are the derivative benefits of the primary key. We'll discuss them in greater detail in the following sections.

Let's learn SQL from A to Z in our complete track!

Updating and Removing Only Specific Rows

Because a primary key identifies each row uniquely, we can use it to update or remove just the specific rows.

Let's consider a situation in which a customer wants to update his/her email address. To do this efficiently, we use the primary key column of the `Customer` table to identify the row to be updated.

```
UPDATE Customer
SET Email = 'g.johns@email.com'
WHERE CustomerId = 432;
```

Now, the email address of a customer with ID 432 is updated.

Similarly, when removing a customer, we use the primary key column for row identification:

```
DELETE FROM Customer WHERE CustomerId = 434;
```

Instead of:

```
DELETE FROM Customer WHERE FirstName = 'Alice' AND LastName = 'Kelley';
```

Now, the customer Alice Kelley is no longer in our database.

When we use a primary key for row identification, we shorten the query length and ensure fast execution of the update or remove operation. This is because of the index created by the primary key constraint on a column that implements it.

Preventing Insertion of Duplicates

This is quite straightforward. The primary key constraint encapsulates the `UNIQUE` and `NOT NULL` constraints. The `UNIQUE` constraint, as its name indicates, ensures the uniqueness of values in the primary key column(s).

The values in a primary key column should be unique and not null, like the `CustomerId` column here:

CustomerId	FirstName	LastName
432	Grace	Johns
433	Robert	Black
434	Alice	Kelley

Because of the `UNIQUE` constraint, the following in the `CustomerId` column is not valid as a primary key:

CustomerId	FirstName	LastName
432	Grace	Johns
433	Robert	Black
433	Alice	Kelley

Hence, using a primary key prevents the insertion of duplicate values. If you try to insert the same value twice in a primary key column, the database will not let you!

Faster Sorting, Searching, and Query Execution

Using primary keys improves the performance of sorting, searching, and query execution. This is because of the index created on the primary key column(s).

The index is created by default on the primary key column(s). Depending on the nature of the primary key, this index is either single-column or multicolumn.

If you are still wondering about how an index works, imagine searching for a keyword in an encyclopedia. You don't go page by page to find the desired keyword. That could take ages! Instead, you go to the index section to look up the keyword and find the page numbers where it is mentioned.

Sorting Data in a Table by the Primary Key Column(s)

In SQL, we use the `ORDER BY` clause to get the query results sorted according to our specifications. However, by default, the data is sorted based on the primary key column in ascending order.

When we run a `SELECT` statement on the **`Customer`** table, we get the query result sorted by the primary key column in ascending order.

CustomerId	FirstName	LastName
432	Grace	Johns
433	Robert	Black
434	Alice	Kelley

And if we use the `Email` column as the natural primary key, we get the following.

Email	FirstName	LastName
a.kelley@email.com	Alice	Kelley
g.johns@email.com	Grace	Johns
robert.black@email.com	Robert	Black

The `ORDER BY` clause is used to specify the column on which the sorting takes place. Consider the following query as a comparison:

```
SELECT FirstName, LastName
FROM Customer
```

```
ORDER BY FirstName DESC;
```

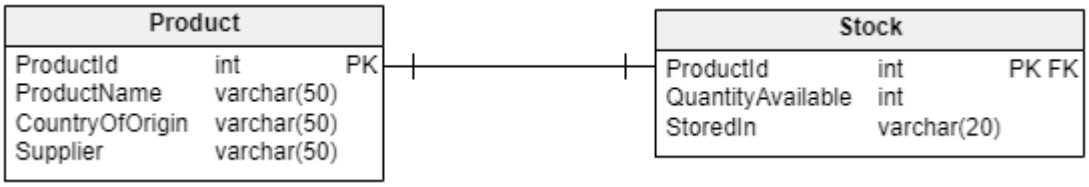
This is what we get if we run the query:

FirstName	LastName
Robert	Black
Grace	Johns
Alice	Kelley

Sorting is a process that arranges data in meaningful order according to specifications. When the data is sorted, the process to understand, analyze, and visualize data is much smoother.

Creating Referential Integrity Between Tables

The primary key column(s) can be used (along with the foreign key) to create a reference between two tables. As shown earlier, the primary key column of one table can be the foreign key column in another table. Thus, it creates a link between the two tables.



We'll have more on the links between tables in the next section. This article provides some more insights into why we use primary keys in database tables.

Why Relational Databases Use Foreign Keys

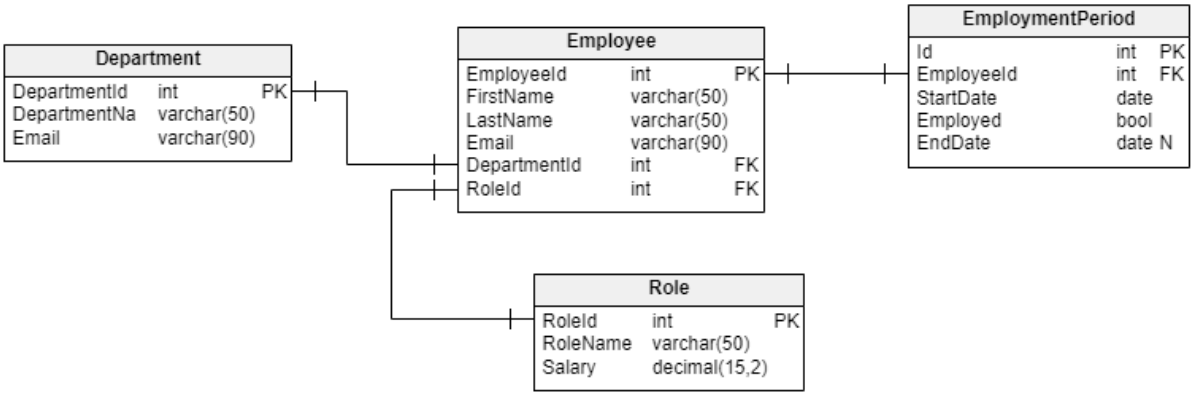
The main task of foreign keys is to create a link between the two tables.

Linking the Data

The foreign key constraint allows linking data between tables. This allows us to keep the data logically grouped but still linked to one another.

Another benefit of the foreign key is it is not restricted to the primary key. It can be defined on some other unique key column(s).

Let's look at an example that presents complex data linkage using foreign keys.



This may look like a challenge to track down all the connections. Let's break down the ER diagram above into smaller pieces.

The `Employee` table stores the data of each employee:

- The `EmployeeId` column is the primary key.
- The `FirstName`, `LastName`, and `Email` columns store the relevant data.
- The `DepartmentId` column is a foreign key that links the **`Employee`** table to the **`Department`** table; it assigns a department to each employee.
- The `RoleId` column is a foreign key that links the **`Employee`** table to the **`Role`** table; it assigns a role to each

The **`Department`** table stores the relevant data of each department, such as its name and email address.

The **`Role`** table stores the possible roles for the employees and the salaries associated with the roles.

The **`EmploymentPeriod`** table stores the dates of employment for every employee who worked or still works at the company. The `EndDate` column is nullable because it should allow null values (for example, in the case of an employee who still works at the company).

This is how the foreign key logically links data between all the tables.

Complete learning path for modern SQL. Try out our SQL from A to Z track!

Enforcing Referential Integrity

In addition to linking data, the foreign key maintains the correctness of every link. By doing so, it keeps the data clean.

The foreign key ensures the existence of a referenced record in the primary table. If the primary table does not contain a particular record, that record cannot be present in the foreign table.

The foreign key also ensures the correctness of the reference between tables. When a value in the primary table is modified or removed, the values in foreign tables are adjusted accordingly. The possible actions include:

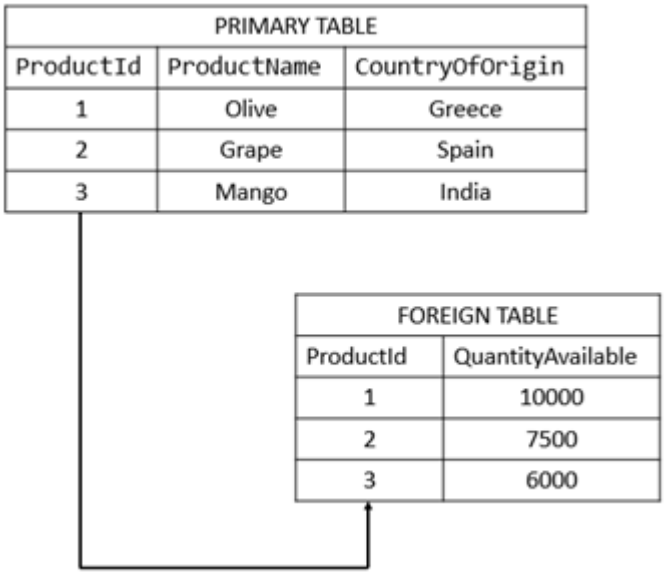
`CASCADE`: the corresponding record in a foreign table is updated/deleted whenever the referenced record from the primary table is updated/deleted.

`RESTRICT`: the corresponding records in the primary table referenced by the foreign table cannot be updated/deleted.

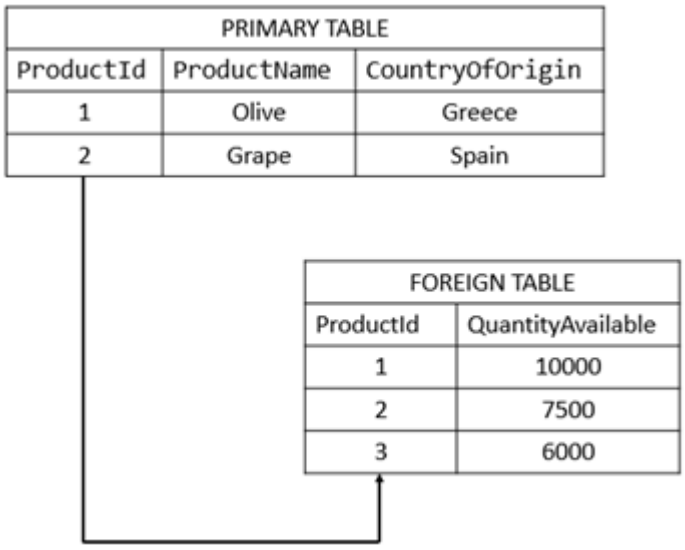
`SET NULL`: the records in foreign tables are set to NULL whenever the corresponding records in the primary table are updated/deleted.

`SET DEFAULT`: the records in foreign tables are set to their default value whenever the corresponding records in the primary table are updated/deleted.

This is correct:



and this is not correct:



PRIMARY TABLE		
ProductId	ProductName	CountryOfOrigin
1	Olive	Greece
2	Grape	Spain

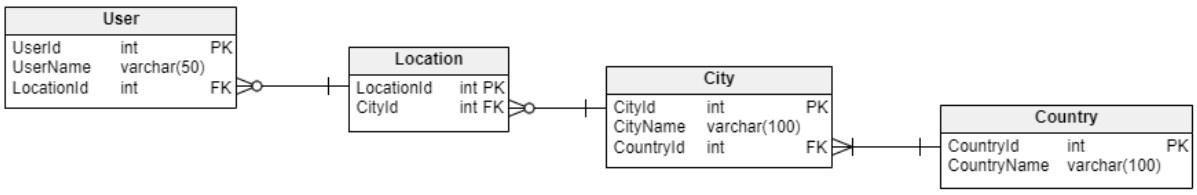
FOREIGN TABLE	
ProductId	QuantityAvailable
1	10000
2	7500
3	6000

because the foreign table references the record with `ProductId = 3`, which does not exist in the primary table. By default, the database does not let us remove the record with `ProductId = 3` from the primary table if the foreign table references it. Here, we can use the actions on `UPDATE/DELETE` that set the rules for the update or removal of records in the primary table. The foreign key not only creates the link between the two tables but also ensures referential integrity.

Investigating Database Dependencies

The foreign keys make it easier to investigate dependencies between tables.

By going up or down the foreign key hierarchy, we can visualize the entire database design and complete the ER diagrams.



Looking at the **`User`** table, we see the `LocationId` column is a foreign key. Let's go down that road. Now, we get to the **`Location`** table which has the `CityId` column as a foreign key. Going down the hierarchy, we get to the **`City`** table. The **`City`** table stores the name of the city and has the `CountryId` column as a foreign key. Finally, we get to the **`Country`**

table that stores all the countries. The countries are linked to the cities in the `City` table.

Improved Performance in Data Retrieval

The links between tables created by foreign keys make it easier for us to track down the data. Now, you can imagine how helpful it is for the database engine when running queries or other operations!

The links between tables improve the performance of data retrieval operations. It is easier for the database engine to locate the necessary connections between the tables, especially when using JOINs.

Here, you can find some more insight into the benefits of foreign keys.

Using Primary and Foreign Keys Together

Although a unique key can be used in place of a primary key as mentioned earlier, we usually use a primary key whenever a foreign key link is made. Here is a quick recap of why we use primary and foreign keys together in relational databases.

Defining the Relationship Between Tables

When linking two tables, each table gets its title:

One is the primary table.
The other is the foreign table.

The primary table provides the primary/unique key. The foreign table accepts this key and incorporates it within itself.

Easier Data Update/Removal

The update or removal of data from the primary table is regulated by the defined actions, such as `CASCADE`, `RESTRICT`, `SET NULL`, and `SET DEFAULT`, as described earlier.

The foreign key constraint ensures the data integrity of the link between the two tables.

Drawbacks of Using Primary and Foreign Keys

Now that we have gone through the benefits of using primary and foreign keys in relational databases, you may be wondering if there are any drawbacks. Yes, there are. However, the benefits usually outweigh the drawbacks. The drawbacks of using keys in relational databases include the following:

- The primary key automatically creates an index on a column or a set of columns on which it is implemented. Any index takes storage space. If you are running out of storage, this may be a problem.
- The primary key implemented on a column or a set of columns prevents the insertion of duplicate values. Think twice before deciding on the column(s) to use as the primary key to ensure this does not lead to problems.
- Similarly, the primary key does not allow null values. So, think twice before deciding which column(s) to choose for the primary key.
- There may be a small degradation in performance related to the maintenance of all the foreign keys.
- Every insertion into the linked tables is checked to ensure referential integrity. Therefore, it may take longer to insert bulk data. Also, data inserts must be performed in a specific order and under specific rules. For example, you cannot insert into rows a foreign table if their corresponding rows do not exist in the primary table.

Although the drawbacks are not major, evaluate them carefully when making decisions related to your database design.

Some Database Engines Do Not Enforce Primary and Foreign Keys

Some database engines, such as Snowflake, support defining and maintaining primary and foreign keys simply to be compatible with other database engines. They don't enforce these constraints.

At the time of this article, the Snowflake database implements primary keys using the `NOT NULL` constraint, which is the only primary key constraint enforced. It does not enforce unique values, even for the primary key column(s).

LearnSQL.com provides a one-stop-shop for all things SQL, covering basic to advanced concepts in one single platform. LearnSQL.com is specifically geared towards SQL. It offers 30 interactive SQL courses that range in difficulty from beginner to advanced and monthly SQL challenges to practice your SQL skills.

What's Next?

Now you see how important primary and foreign keys are in relational databases. All the benefits of primary and foreign keys come from two simple ideas: unique row identification (primary key) and connection links between tables (foreign key).

It is essential to know all there is to know about primary and foreign keys. Make sure to get enough practice! You'll instantly see your professional growth in relational databases. To get you started, check out our article on the most popular databases. Then get some practice by following our courses:

> The Basics of Creating Tables in SQL.
> Creating Database Structure.

Good luck and have fun!
