**\*The text within this document is from internet sources, and the tables and examples are invented.**

**SQL Primary Key**

In order for a table to be qualified as a relational table it must have a primary key.

The primary key consists of one or more columns whose data contained within is used to uniquely identify each row in the table. You can think of them as an address. If the rows in a table were mail boxes, then the primary key would be the listing of street addresses.

When a PK is composed of multiple columns, the data from each column is used to determine whether a row is unique.

In order to be a primary key, several conditions must be fulfilled:

* Unique;
* We’re referent to a data within the rows, not the column names themselves;
* No value in the columns can be blank or NULL;
* A table has just one PK, and its definition is mandatory.

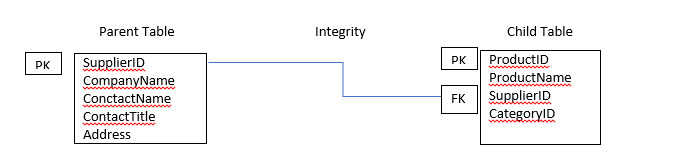
The PK for each table is stored in an index. The index Is used to enforce the uniqueness requirement. It also makes it easy for foreign key values to refer back to corresponding PK values.

**SQL Foreign Key**

Foreign key ensures values in one table must be present in another table

In order to be a FK, several conditions must be fulfilled:

* NULL is allowed;
* The table being referenced is called the Parent Table;
* The table with the foreign key is called Child Table;
* The foreign key in child table references the PK in the parent table;
* This parent-child relationship enforces the rule which is known as ‘’Referential Integrity’’.

**

You can use SQL syntax,, DESCRIBE suppliers;’’ to describe the structure of a table

**SQL INNER JOIN**

SQL JOINS are used to combine records from two or more tables in a database

The INNER JOIN keyword selects records that have matching values in both tables

Example:

We want to get the following information using a query:

|  |  |
| --- | --- |
| CompanyName | ProductName |
| ----- | ----- |
| ----- | ----- |

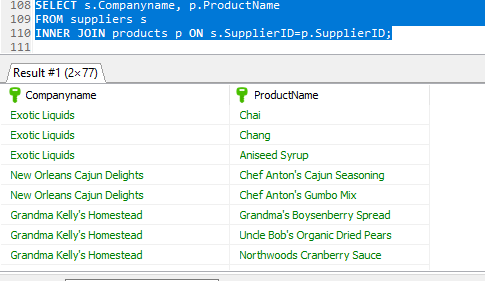
The JOIN is used to join the PK in one table with the FK in another table.

SQL Syntax:

**SELECT** s.Companyname, p.ProductName

**FROM** suppliers s

**INNER** **JOIN** products p **ON** s.SupplierID=p.SupplierID



Also, you can add WHERE clause (condition) or Limit:

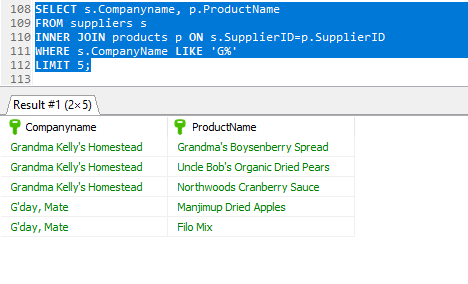
**SELECT** s.Companyname, p.ProductName

**FROM** suppliers s

**INNER** **JOIN** products p **ON** s.SupplierID=p.SupplierID

**WHERE** s.CompanyName **LIKE** 'G%'

**LIMIT** 5;



If there are records in ‘’Suppliers’’ table that do not have matches in ‘’Products’’, these records will not be shown

**SQL LEFT JOIN**

LEFT JOIN returns all rows from the left table whether or not there is a matching row in the right table.

Example:

Write a query to show UnitPrice, ProductName, and Discount, as result name: Price, Product. Then list the records from the highest price, with Price more than 20

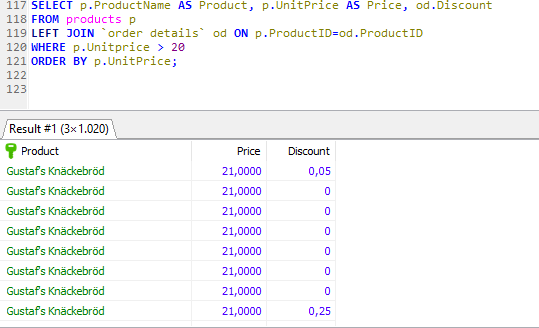
**SELECT** p.ProductName **AS** Product, p.UnitPrice **AS** Price, od.Discount

**FROM** products p

**LEFT** **JOIN** `order details` od **ON** p.ProductID=od.ProductID

**WHERE** p.UnitPrice> 20

**ORDER** **BY** p.UnitPrice;



**SQL RIGHT JOIN**

The RIGHT JOIN starts selecting data from the right table and matching with the rows from the left table. The RIGHT JOIN returns a result set that includes all rows in the right table, whether or not have matching rows from the left table

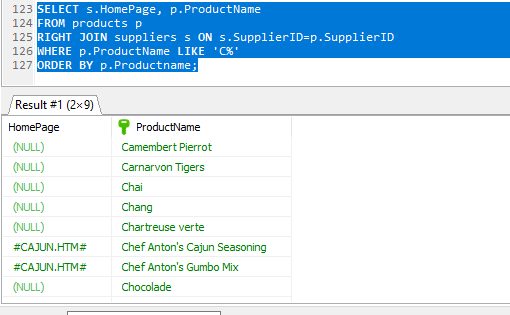
**SELECT** s.HomePage, p.ProductName

**FROM** products p

**RIGHT** **JOIN** suppliers s **ON** s.SupplierID=p.SupplierID

**WHERE** p.ProductName **LIKE** 'C%'

**ORDER** **BY** p.Productname;



**SQL FULL OUTER JOIN**

FULL OUTER JOIN returns a result set that matches rows from both left and right tables. When no matching rows exists for the row in the left table, the columns in the right table will have nulls. Similarly, when no matching rows exist for the row in the right table, the column of the left table will have nulls.

Example:

**SELECT** c.CompanyName, c.Address, c.Fax, o.ShipRegion

**FROM** customers c

**LEFT** **JOIN** orders o **ON** c.CustomerID=o.CustomerID

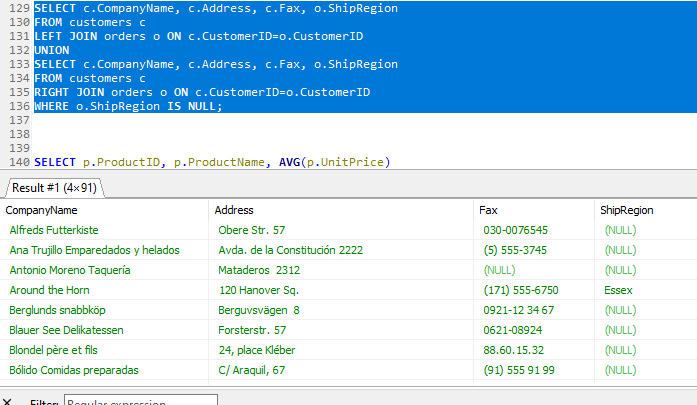
**UNION**

**SELECT** c.CompanyName, c.Address, c.Fax, o.ShipRegion

**FROM** customers c

**RIGHT** **JOIN** orders o **ON** c.CustomerID=o.CustomerID

**WHERE** o.ShipRegion **IS** **NULL**;



**NESTED**

In NESTED queries, a query is written inside a query and embedded within the where clause. The result of inner query is used in execution of outer query.

A subquery is used to return data that will be used in main query as a condition to further restrict the data to be retrieved.

* Subqueries must be enclosed within parentheses;
* Subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns;
* An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY;
* Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.

Example:

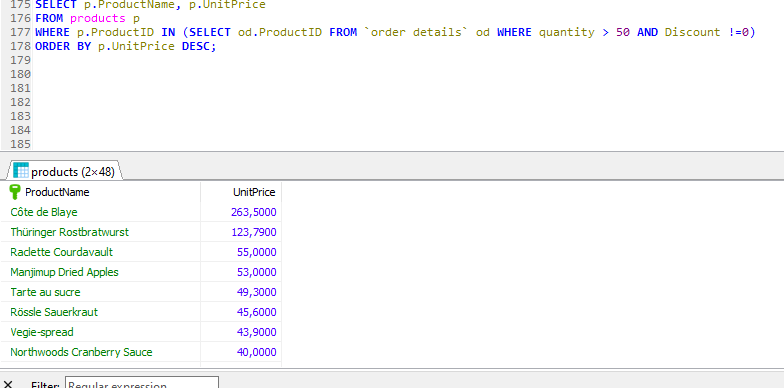
List Products with names, UnitPrice and order Quantities greater than 50, with discount and list the records from the highest price

**SELECT** p.ProductName, p.UnitPrice

**FROM** products p

**WHERE** p.ProductID **IN** (**SELECT** od.ProductID **FROM** `order details` od **WHERE** quantity > 50 **AND** Discount!=0)

**ORDER** **BY** p.UnitPrice **DESC**;



Користена литература:

<https://www.essentialsql.com/what-is-the-difference-between-a-primary-key-and-a-foreign-key/>

<https://www.w3resource.com/sql/joins/perform-an-inner-join.php>

<https://www.sqlservertutorial.net/sql-server-basics/sql-server-left-join/>

<https://www.mysqltutorial.org/mysql-left-join.aspx>

<https://www.w3schools.com/sql/sql_join_right.asp>

<https://www.w3schools.com/sql/sql_union.asp>

<https://www.tutorialspoint.com/sql/sql-sub-queries.htm>

<https://www.geeksforgeeks.org/nested-queries-in-sql/>