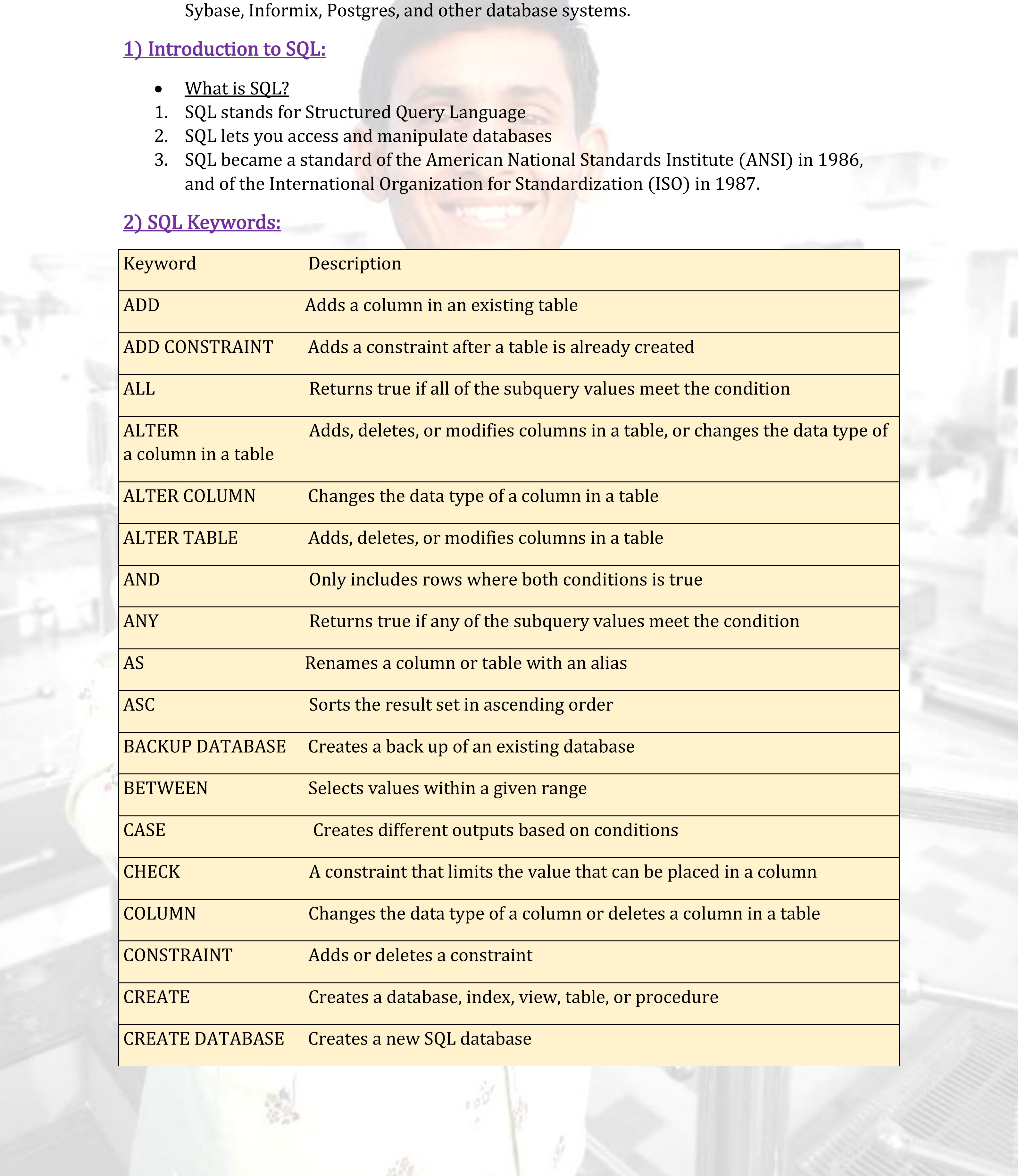


SQL

SQL is a standard language for storing, manipulating and retrieving data in databases. Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

- Our SQL tutorial will teach you how to use SQL in: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

1) Introduction to SQL:

- What is SQL?

1. SQL stands for Structured Query Language
2. SQL lets you access and manipulate databases
3. SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

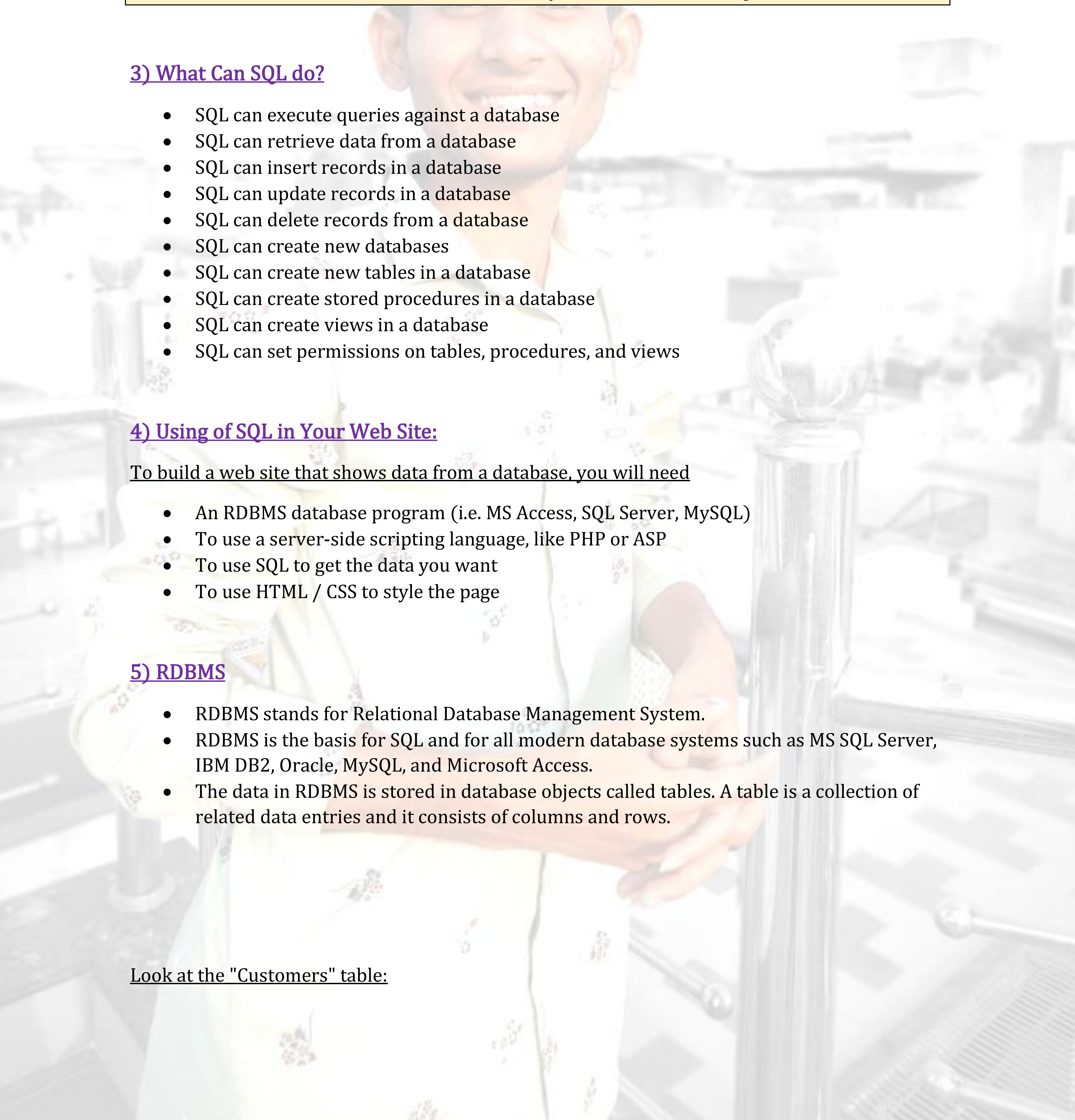
2) SQL Keywords:

Keyword	Description
ADD	Adds a column in an existing table
ADD CONSTRAINT	Adds a constraint after a table is already created
ALL	Returns true if all of the subquery values meet the condition
ALTER	Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table
ALTER COLUMN	Changes the data type of a column in a table
ALTER TABLE	Adds, deletes, or modifies columns in a table
AND	Only includes rows where both conditions is true
ANY	Returns true if any of the subquery values meet the condition
AS	Renames a column or table with an alias
ASC	Sorts the result set in ascending order
BACKUP DATABASE	Creates a back up of an existing database
BETWEEN	Selects values within a given range
CASE	Creates different outputs based on conditions
CHECK	A constraint that limits the value that can be placed in a column
COLUMN	Changes the data type of a column or deletes a column in a table
CONSTRAINT	Adds or deletes a constraint
CREATE	Creates a database, index, view, table, or procedure
CREATE DATABASE	Creates a new SQL database

CREATE INDEX	Creates an index on a table (allows duplicate values)
CREATE OR REPLACE VIEW	Updates a view
CREATE TABLE	Creates a new table in the database
CREATE PROCEDURE	Creates a stored procedure
CREATE UNIQUE INDEX	Creates a unique index on a table (no duplicate values)
CREATE VIEW	Creates a view based on the result set of a SELECT statement
DATABASE	Creates or deletes an SQL database
DEFAULT	A constraint that provides a default value for a column
DELETE	Deletes rows from a table
DESC	Sorts the result set in descending order
DISTINCT	Selects only distinct (different) values
DROP	Deletes a column, constraint, database, index, table, or view
DROP COLUMN	Deletes a column in a table
DROP CONSTRAINT	Deletes a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint
DROP DATABASE	Deletes an existing SQL database
DROP DEFAULT	Deletes a DEFAULT constraint
DROP INDEX	Deletes an index in a table
DROP TABLE	Deletes an existing table in the database
DROP VIEW	Deletes a view
EXEC	Executes a stored procedure
EXISTS	Tests for the existence of any record in a subquery
FOREIGN KEY	A constraint that is a key used to link two tables together
FROM	Specifies which table to select or delete data from
FULL OUTER JOIN	Returns all rows when there is a match in either left table or right table
GROUP BY--Groups the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG)	
HAVING	Used instead of WHERE with aggregate functions
IN	Allows you to specify multiple values in a WHERE clause
INDEX	Creates or deletes an index in a table

INNER JOIN	Returns rows that have matching values in both tables
INSERT INTO	Inserts new rows in a table
INSERT INTO SELECT	Copies data from one table into another table
IS NULL	Tests for empty values
IS NOT NULL	Tests for non-empty values
JOIN	Joins tables
LEFT JOIN	Returns all rows from the left table, and the matching rows from the right table
LIKE	Searches for a specified pattern in a column
LIMIT	Specifies the number of records to return in the result set
NOT	Only includes rows where a condition is not true
NOT NULL	A constraint that enforces a column to not accept NULL values
OR	Includes rows where either condition is true
ORDER BY	Sorts the result set in ascending or descending order
OUTER JOIN	Returns all rows when there is a match in either left table or right table
PRIMARY KEY	A constraint that uniquely identifies each record in a database table
PROCEDURE	A stored procedure
RIGHT JOIN	Returns all rows from the right table, and the matching rows from the left table
ROWNUM	Specifies the number of records to return in the result set
SELECT	Selects data from a database
SELECT DISTINCT	Selects only distinct (different) values
SELECT INTO	Copies data from one table into a new table
SELECT TOP	Specifies the number of records to return in the result set
SET	Specifies which columns and values that should be updated in a table
TABLE	Creates a table, or adds, deletes, or modifies columns in a table, or deletes a table or data inside a table
TOP	Specifies the number of records to return in the result set
TRUNCATE TABLE	Deletes the data inside a table, but not the table itself
UNION	Combines the result set of two or more SELECT statements (only distinct values)

UNION ALL	Combines the result set of two or more SELECT statements (allows duplicate values)
UNIQUE	A constraint that ensures that all values in a column are unique
UPDATE	Updates existing rows in a table
VALUES	Specifies the values of an INSERT INTO statement
VIEW	Creates, updates, or deletes a view
WHERE	Filters a result set to include only records that fulfill a specified condition



3) What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

4) Using of SQL in Your Web Site:

To build a web site that shows data from a database, you will need

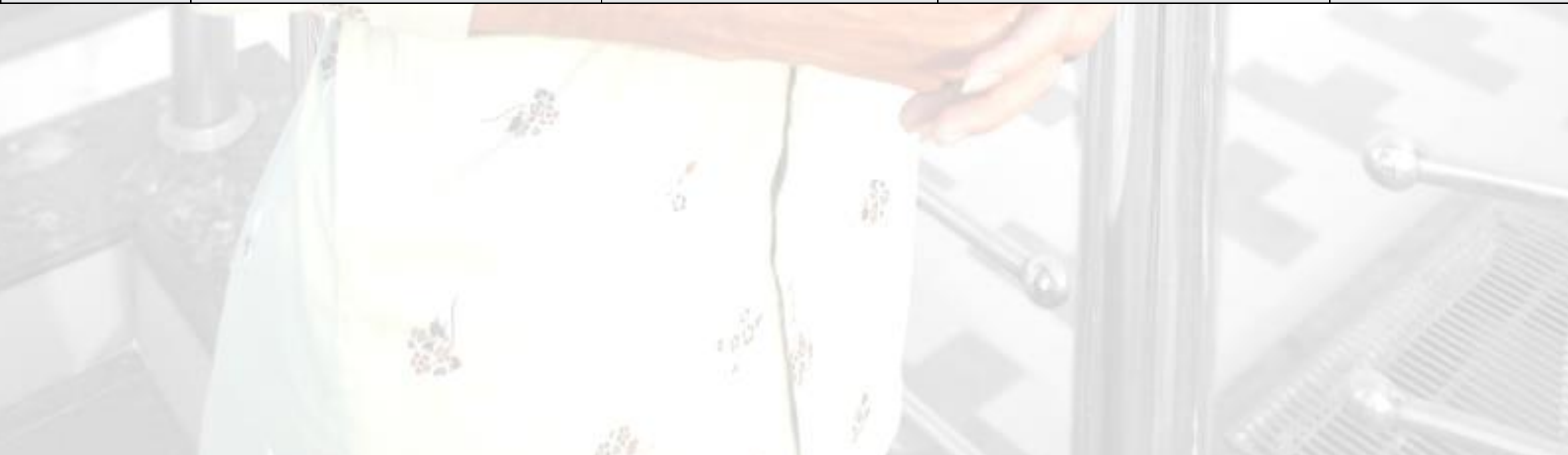
- An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
- To use a server-side scripting language, like PHP or ASP
- To use SQL to get the data you want
- To use HTML / CSS to style the page

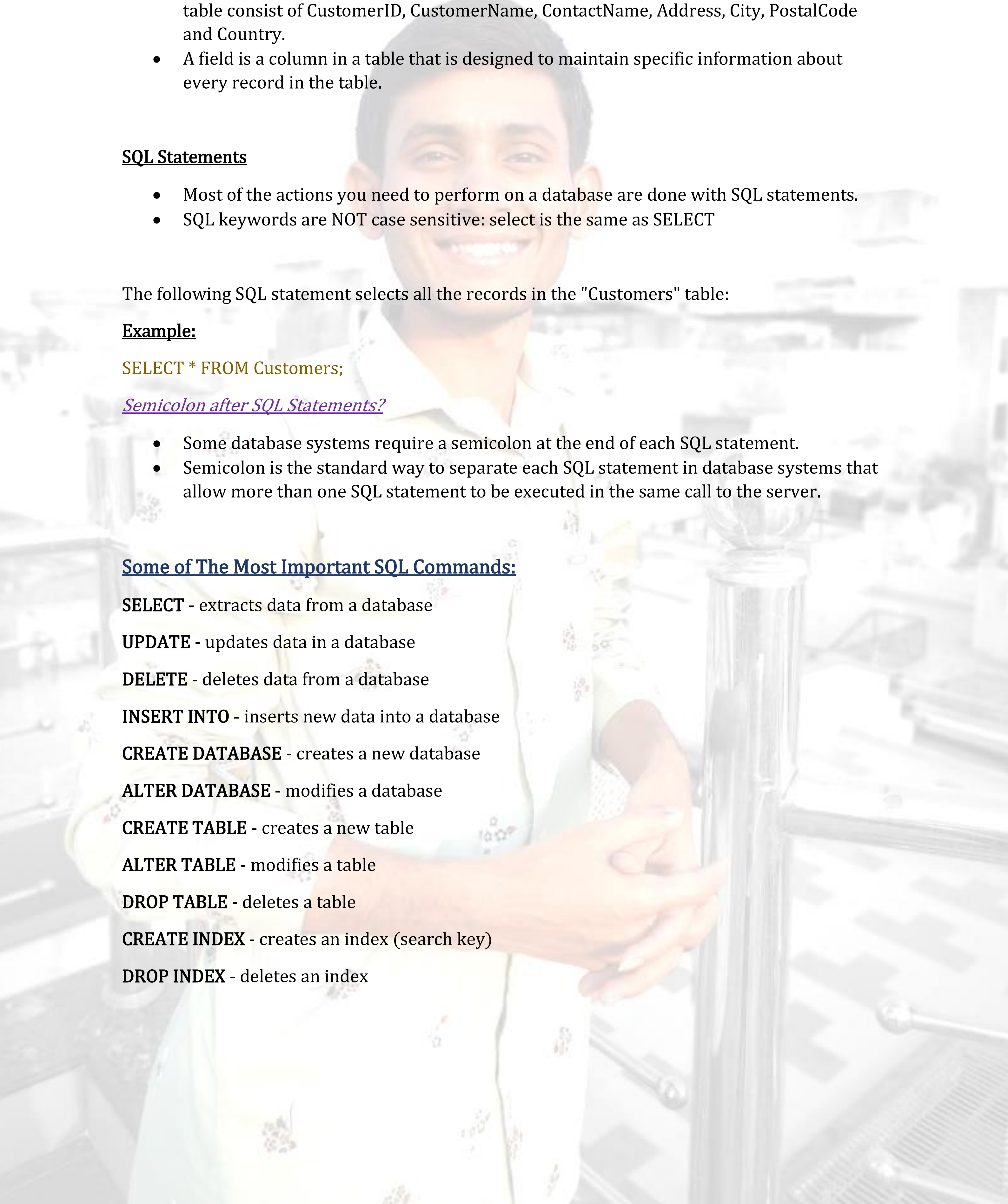
5) RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Look at the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London





Example:

```
SELECT * FROM Customers;
```

- Every table is broken up into smaller entities called fields. The fields in the Customers table consist of CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country.
- A field is a column in a table that is designed to maintain specific information about every record in the table.

SQL Statements

- Most of the actions you need to perform on a database are done with SQL statements.
- SQL keywords are NOT case sensitive: select is the same as SELECT

The following SQL statement selects all the records in the "Customers" table:

Example:

```
SELECT * FROM Customers;
```

Semicolon after SQL Statements?

- Some database systems require a semicolon at the end of each SQL statement.
- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

Some of The Most Important SQL Commands:

SELECT - extracts data from a database

UPDATE - updates data in a database

DELETE - deletes data from a database

INSERT INTO - inserts new data into a database

CREATE DATABASE - creates a new database

ALTER DATABASE - modifies a database

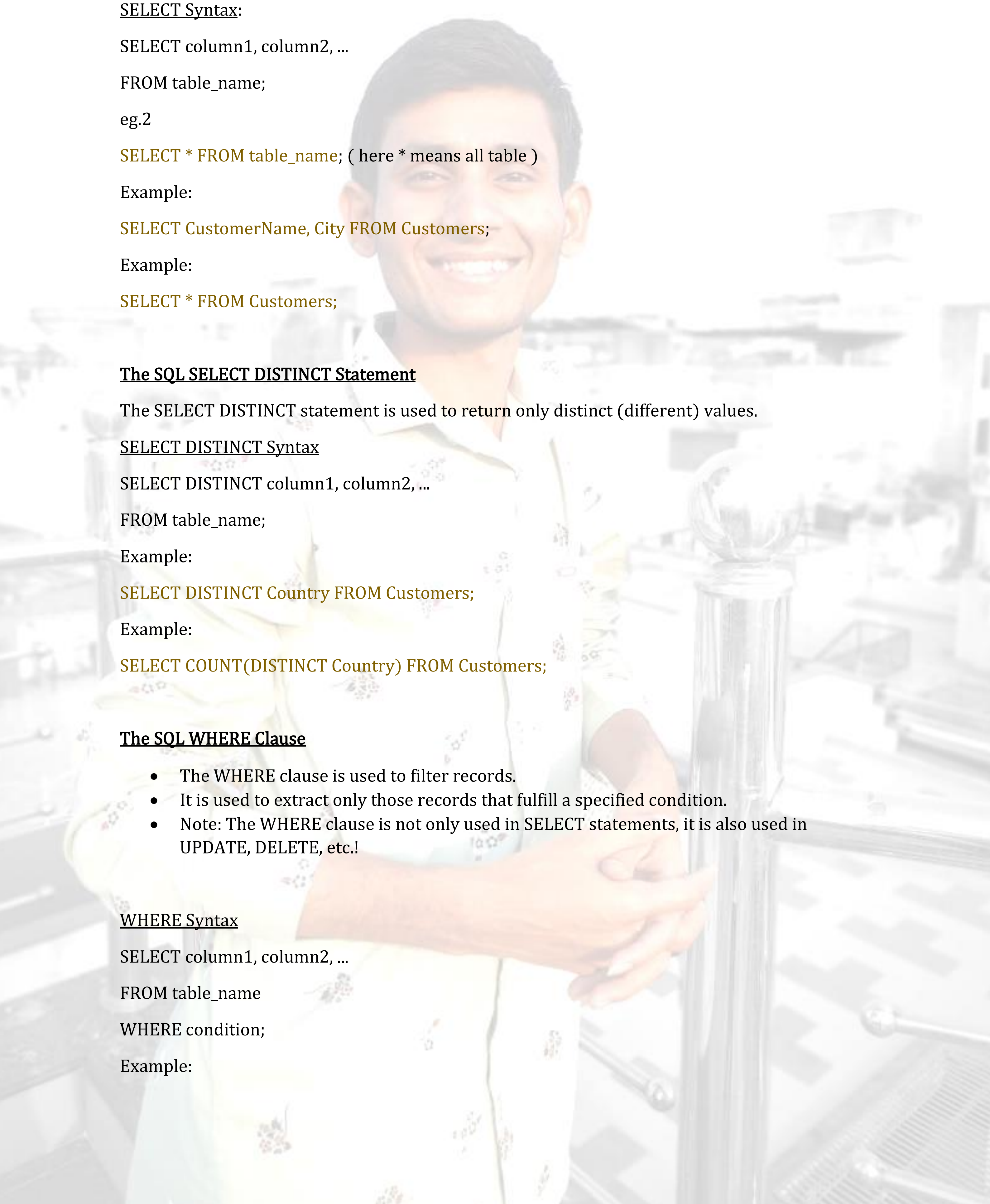
CREATE TABLE - creates a new table

ALTER TABLE - modifies a table

DROP TABLE - deletes a table

CREATE INDEX - creates an index (search key)

DROP INDEX - deletes an index



<u>The SQL SELECT Statement:</u>

- The SELECT statement is used to select data from a database.

<u>SELECT Syntax</u>:

```
SELECT column1, column2, ...
FROM table_name;
```

eg.2

```
SELECT * FROM table_name; ( here * means all table )
```

Example:

```
SELECT CustomerName, City FROM Customers;
```

Example:

```
SELECT * FROM Customers;
```

<u>The SQL SELECT DISTINCT Statement</u>

The SELECT DISTINCT statement is used to return only distinct (different) values.

<u>SELECT DISTINCT Syntax</u>

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Example:

```
SELECT DISTINCT Country FROM Customers;
```

Example:

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

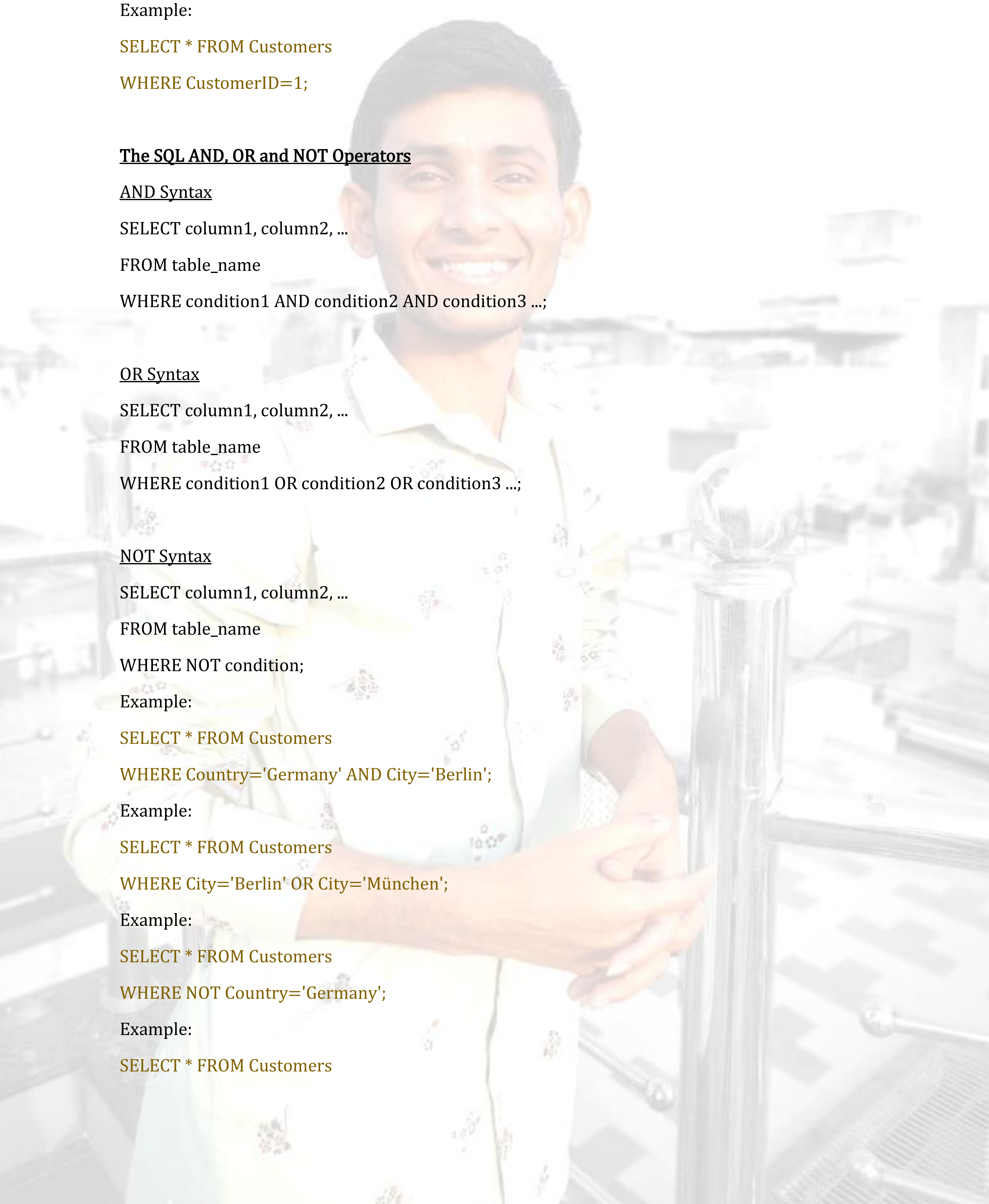
<u>The SQL WHERE Clause</u>

- The WHERE clause is used to filter records.
- It is used to extract only those records that fulfill a specified condition.
- Note: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

<u>WHERE Syntax</u>

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:



```
SELECT * FROM Customers

WHERE Country='Mexico';
```

Example:

```
SELECT * FROM Customers

WHERE CustomerID=1;
```

<u>The SQL AND, OR and NOT Operators</u>

<u>AND Syntax</u>

```
SELECT column1, column2, ...

FROM table_name

WHERE condition1 AND condition2 AND condition3 ...;
```

<u>OR Syntax</u>

```
SELECT column1, column2, ...

FROM table_name

WHERE condition1 OR condition2 OR condition3 ...;
```

<u>NOT Syntax</u>

```
SELECT column1, column2, ...

FROM table_name

WHERE NOT condition;
```

Example:

```
SELECT * FROM Customers

WHERE Country='Germany' AND City='Berlin';
```

Example:

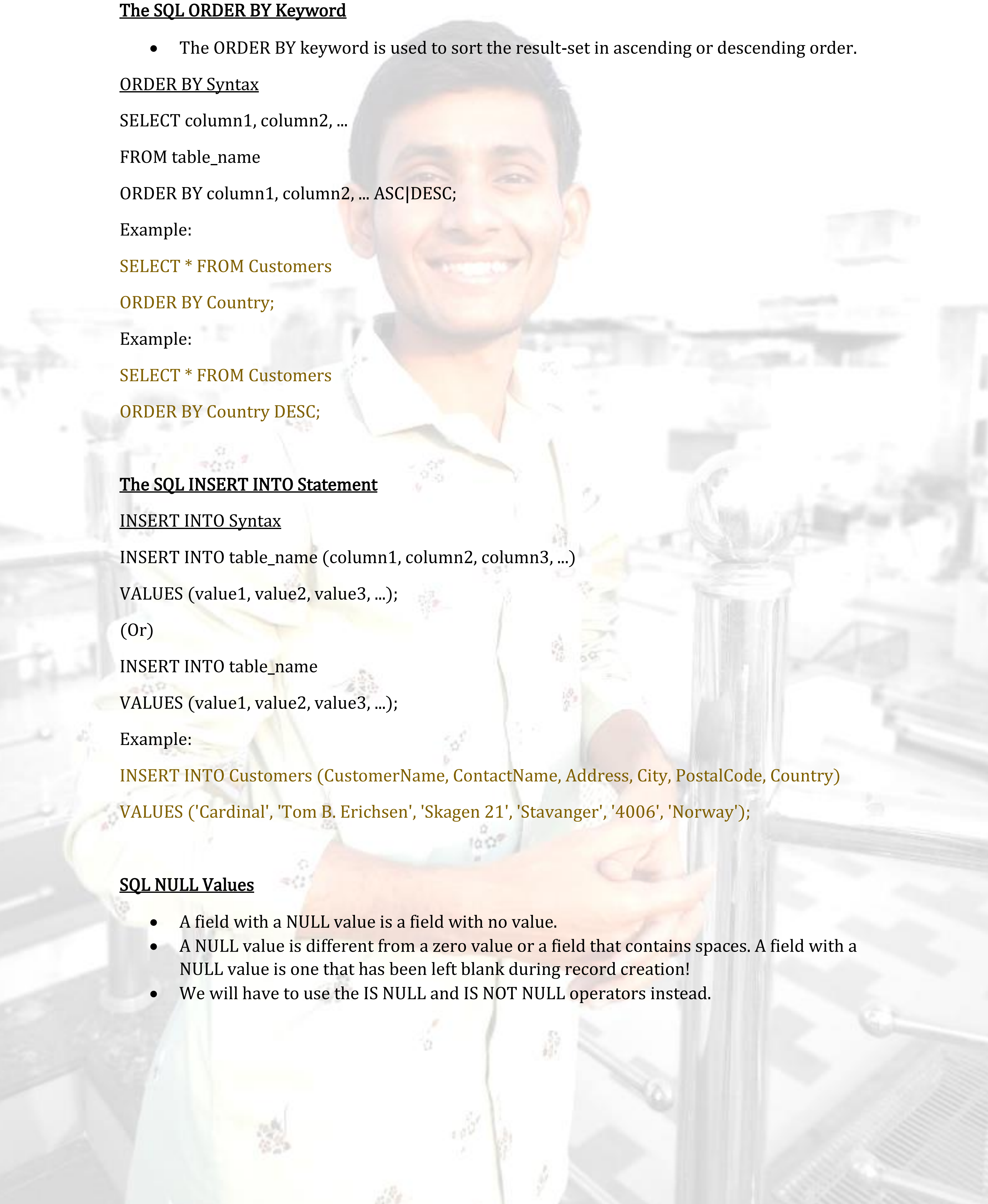
```
SELECT * FROM Customers

WHERE City='Berlin' OR City='München';
```

Example:

```
SELECT * FROM Customers

WHERE NOT Country='Germany';
```

Example:

```
SELECT * FROM Customers
```



```
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

<u>The SQL ORDER BY Keyword</u>

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.

<u>ORDER BY Syntax</u>

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example:

```
SELECT * FROM Customers
ORDER BY Country;
```

Example:

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

<u>The SQL INSERT INTO Statement</u>

<u>INSERT INTO Syntax</u>

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

(Or)

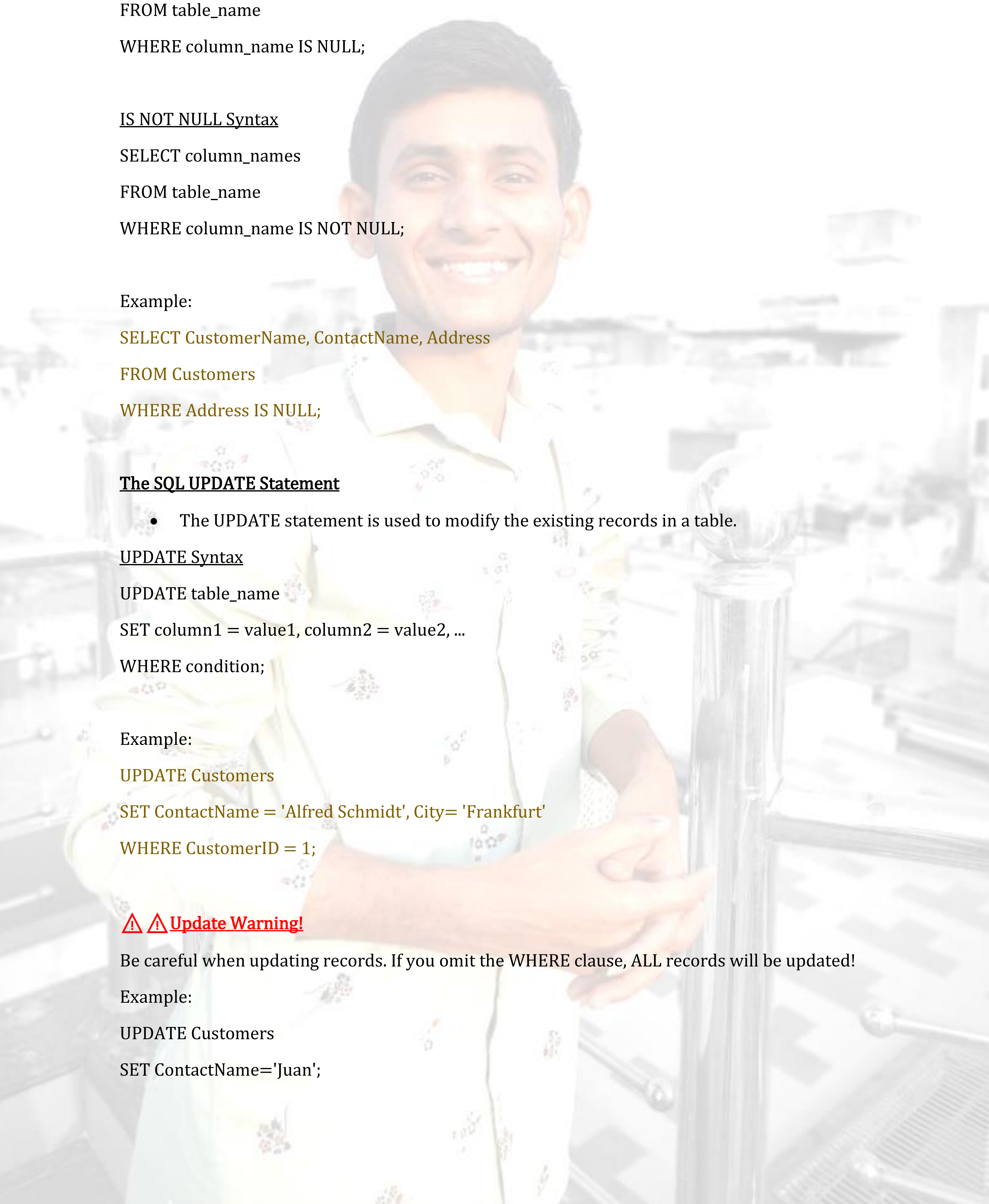
```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

Example:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

<u>SQL NULL Values</u>

- A field with a NULL value is a field with no value.
- A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!
- We will have to use the IS NULL and IS NOT NULL operators instead.



IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

Example:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

The SQL UPDATE Statement

- The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example:

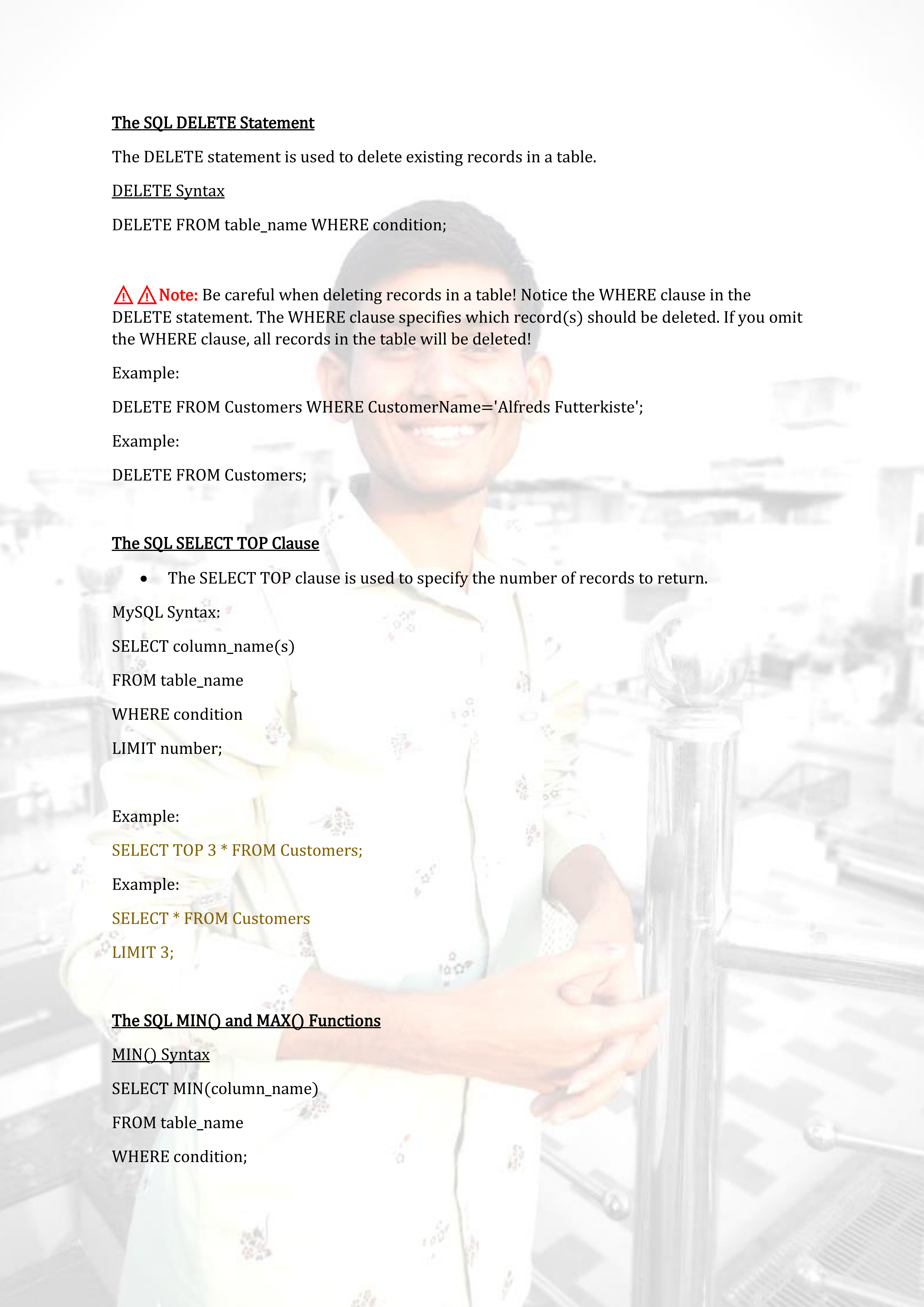
```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

⚠ ⚠ **Update Warning!**

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

Example:

```
UPDATE Customers
SET ContactName='Juan';
```



The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

⚠⚠**Note:** Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

Example:

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Example:

```
DELETE FROM Customers;
```

The SQL SELECT TOP Clause

- The SELECT TOP clause is used to specify the number of records to return.

MySQL Syntax:

```
SELECT column_name(s)

FROM table_name

WHERE condition

LIMIT number;
```

Example:

```
SELECT TOP 3 * FROM Customers;
```

Example:

```
SELECT * FROM Customers

LIMIT 3;
```

The SQL MIN() and MAX() Functions

MIN() Syntax

```
SELECT MIN(column_name)

FROM table_name

WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)

FROM table_name

WHERE condition;
```

Example:

```
SELECT MIN(Price) AS SmallestPrice

FROM Products;
```

Example:

```
SELECT MAX(Price) AS LargestPrice

FROM Products;
```

The SQL COUNT(), AVG() and SUM() Functions

OUNT() Syntax

```
SELECT COUNT(column_name)

FROM table_name

WHERE condition;
```

- The AVG() function returns the average value of a numeric column.

AVG() Syntax

```
SELECT AVG(column_name)

FROM table_name

WHERE condition;
```

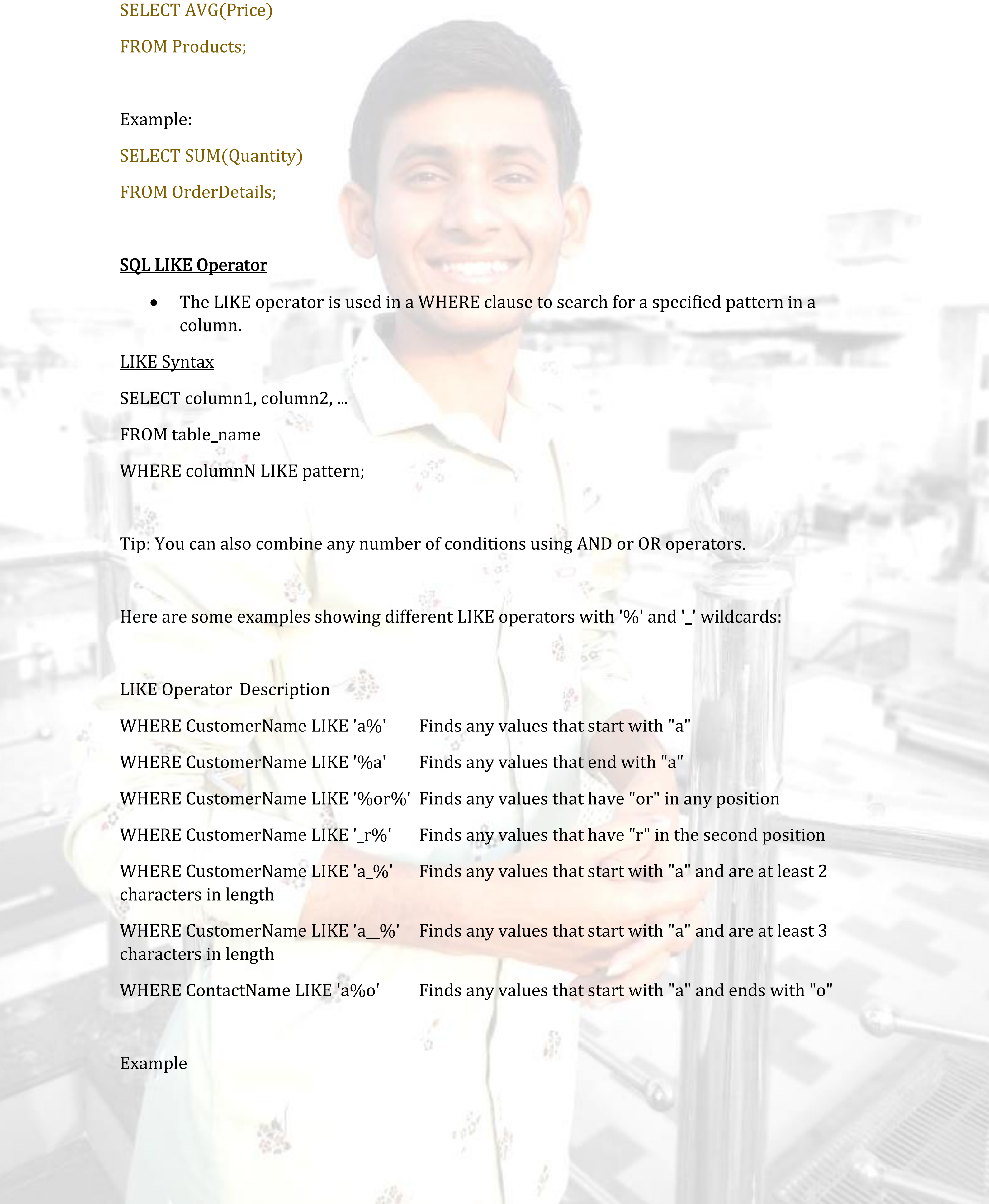
- The SUM() function returns the total sum of a numeric column.

SUM() Syntax

```
SELECT SUM(column_name)

FROM table_name

WHERE condition;
```

Example:

```
SELECT COUNT(ProductID)

FROM Products;
```



Note: NULL values are not counted.

Example:

```
SELECT AVG(Price)
FROM Products;
```

Example:

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

SQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

LIKE Syntax

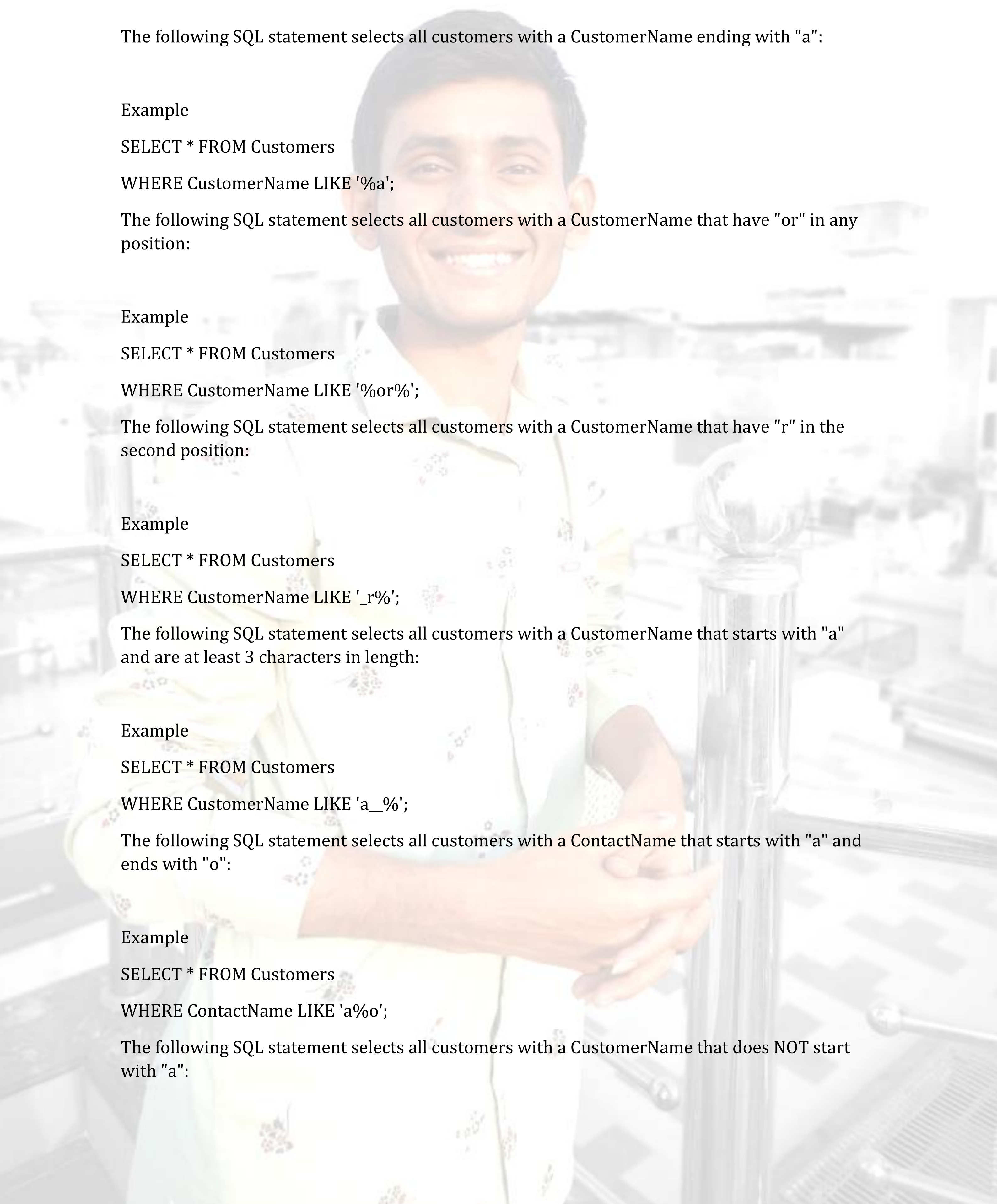
```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Tip: You can also combine any number of conditions using AND or OR operators.

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Example



```
SELECT * FROM Customers

WHERE CustomerName LIKE 'a%';
```

The following SQL statement selects all customers with a CustomerName ending with "a":

Example

```
SELECT * FROM Customers

WHERE CustomerName LIKE '%a';
```

The following SQL statement selects all customers with a CustomerName that have "or" in any position:

Example

```
SELECT * FROM Customers

WHERE CustomerName LIKE '%or%';
```

The following SQL statement selects all customers with a CustomerName that have "r" in the second position:

Example

```
SELECT * FROM Customers

WHERE CustomerName LIKE '_r%';
```

The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

Example

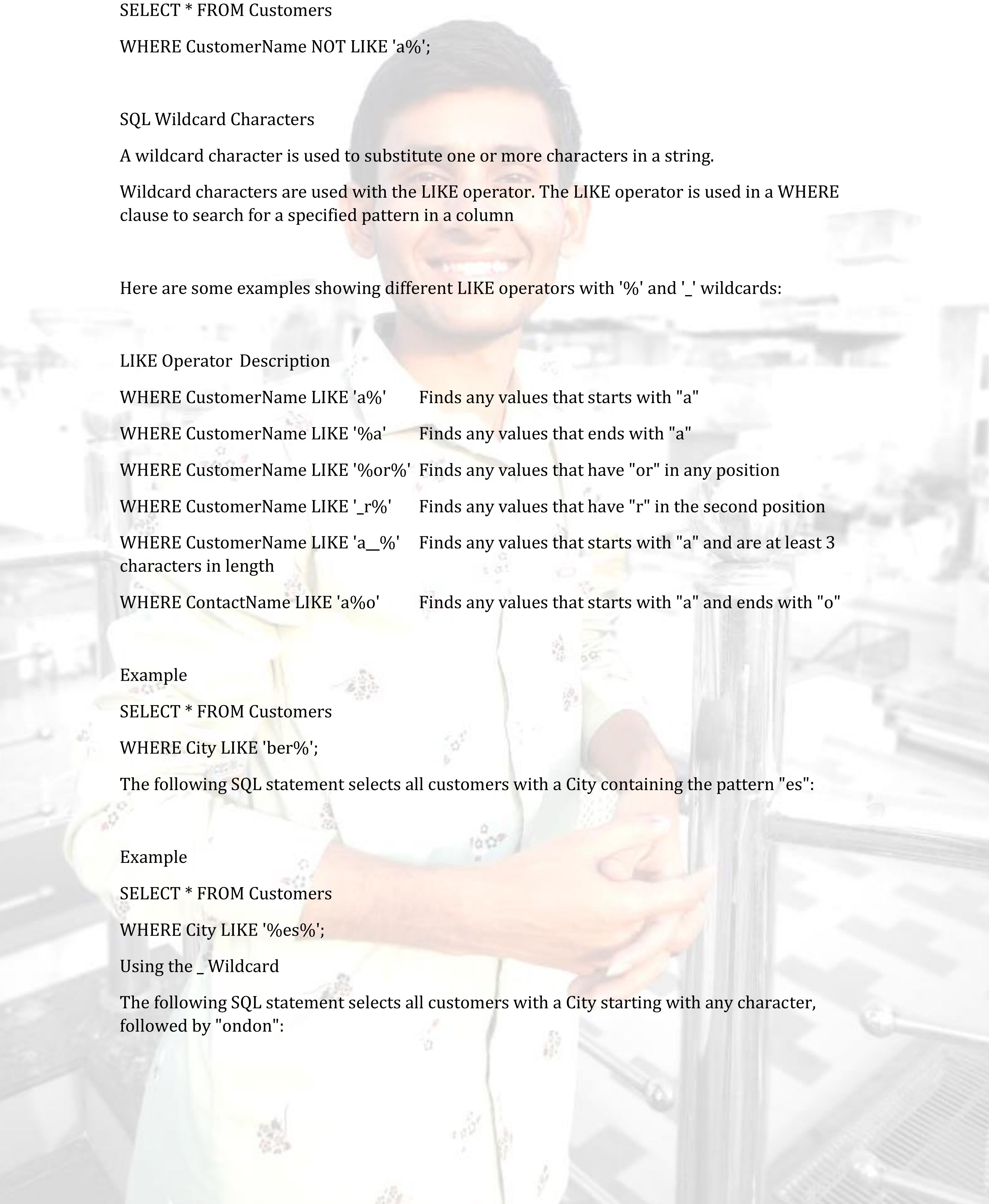
```
SELECT * FROM Customers

WHERE CustomerName LIKE 'a__%';
```

The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":

Example

```
SELECT * FROM Customers

WHERE ContactName LIKE 'a%o';
```

The following SQL statement selects all customers with a CustomerName that does NOT start with "a":



Example

```
SELECT * FROM Customers

WHERE CustomerName NOT LIKE 'a%';
```

SQL Wildcard Characters

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

Example

```
SELECT * FROM Customers

WHERE City LIKE 'ber%';
```

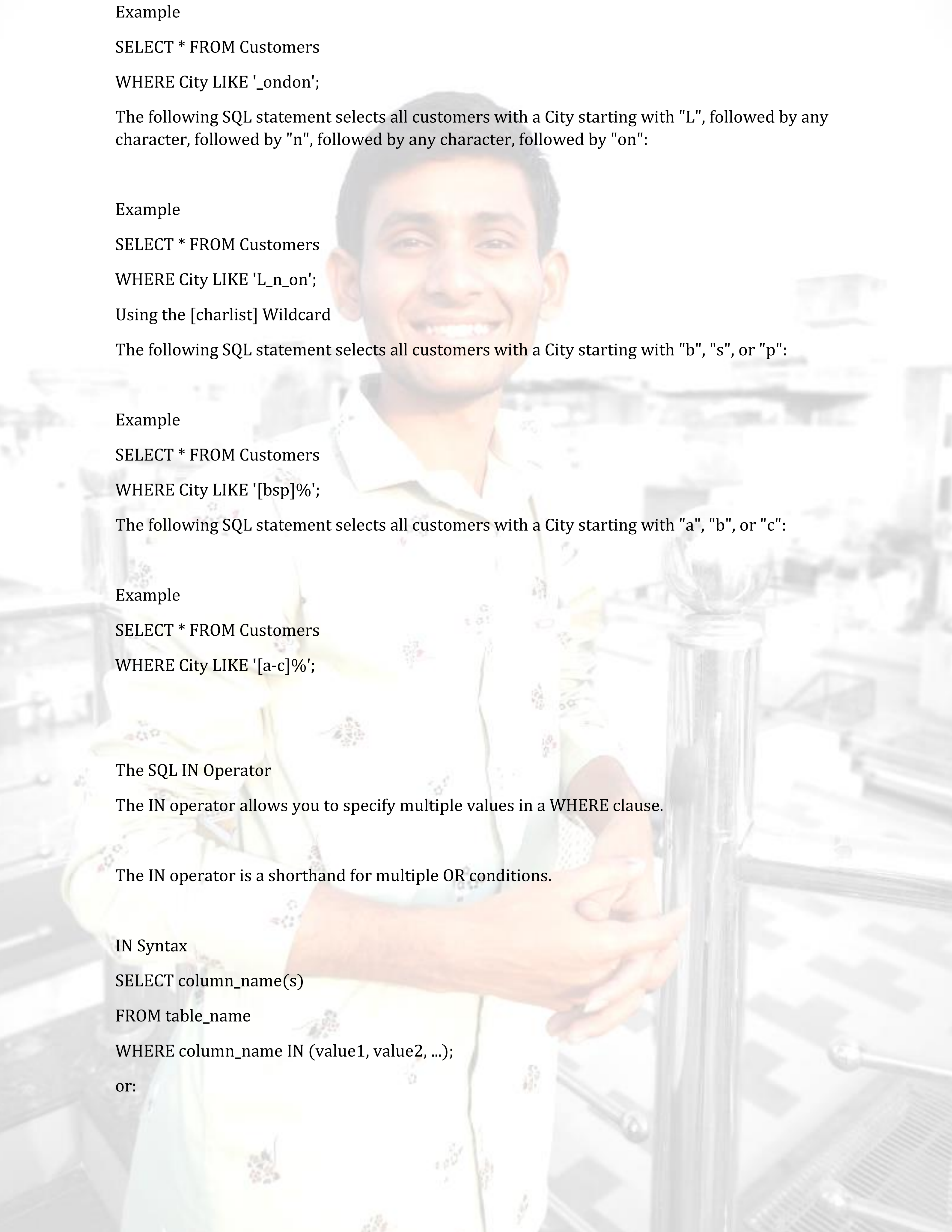
The following SQL statement selects all customers with a City containing the pattern "es":

Example

```
SELECT * FROM Customers

WHERE City LIKE '%es%';
```

Using the _ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "ondon":



Example

```
SELECT * FROM Customers
WHERE City LIKE '_ondon';
```

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

Example

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

Using the [charlist] Wildcard

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

Example

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

Example

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

The SQL IN Operator

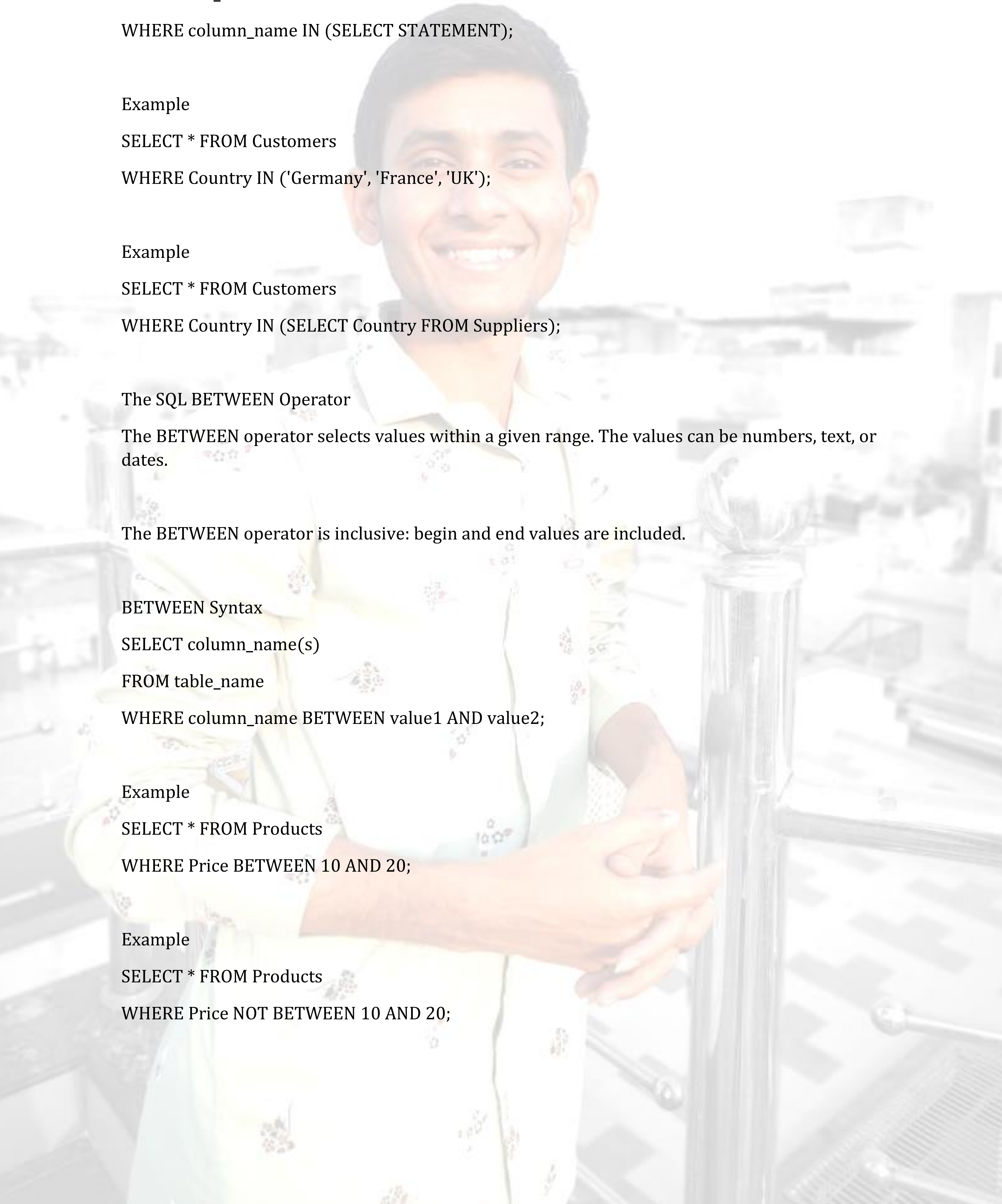
The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

or:



```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

Example

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

Example

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

The SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

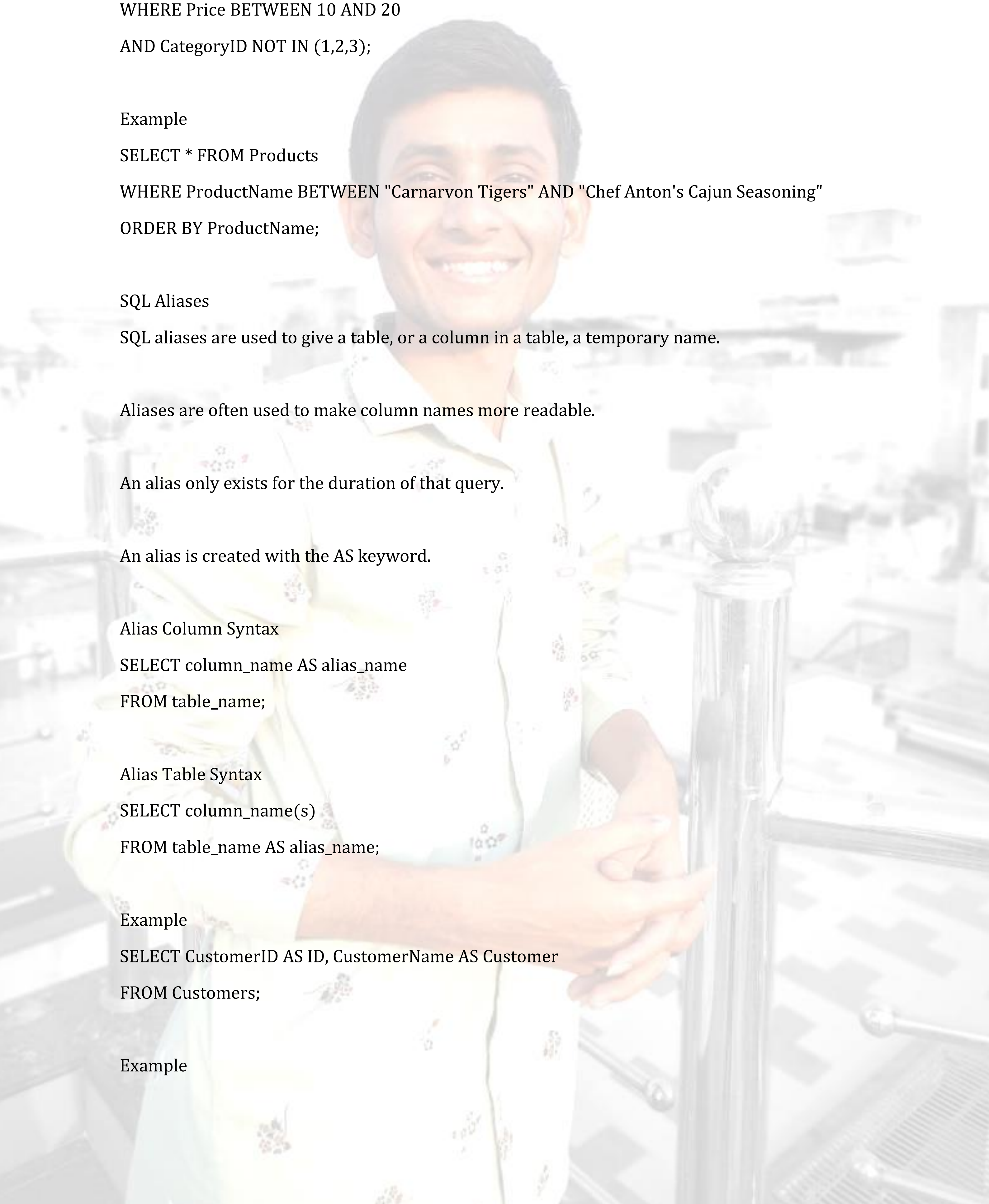
```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

Example

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```



Example

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

Example

```
SELECT * FROM Products
WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun Seasoning"
ORDER BY ProductName;
```

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

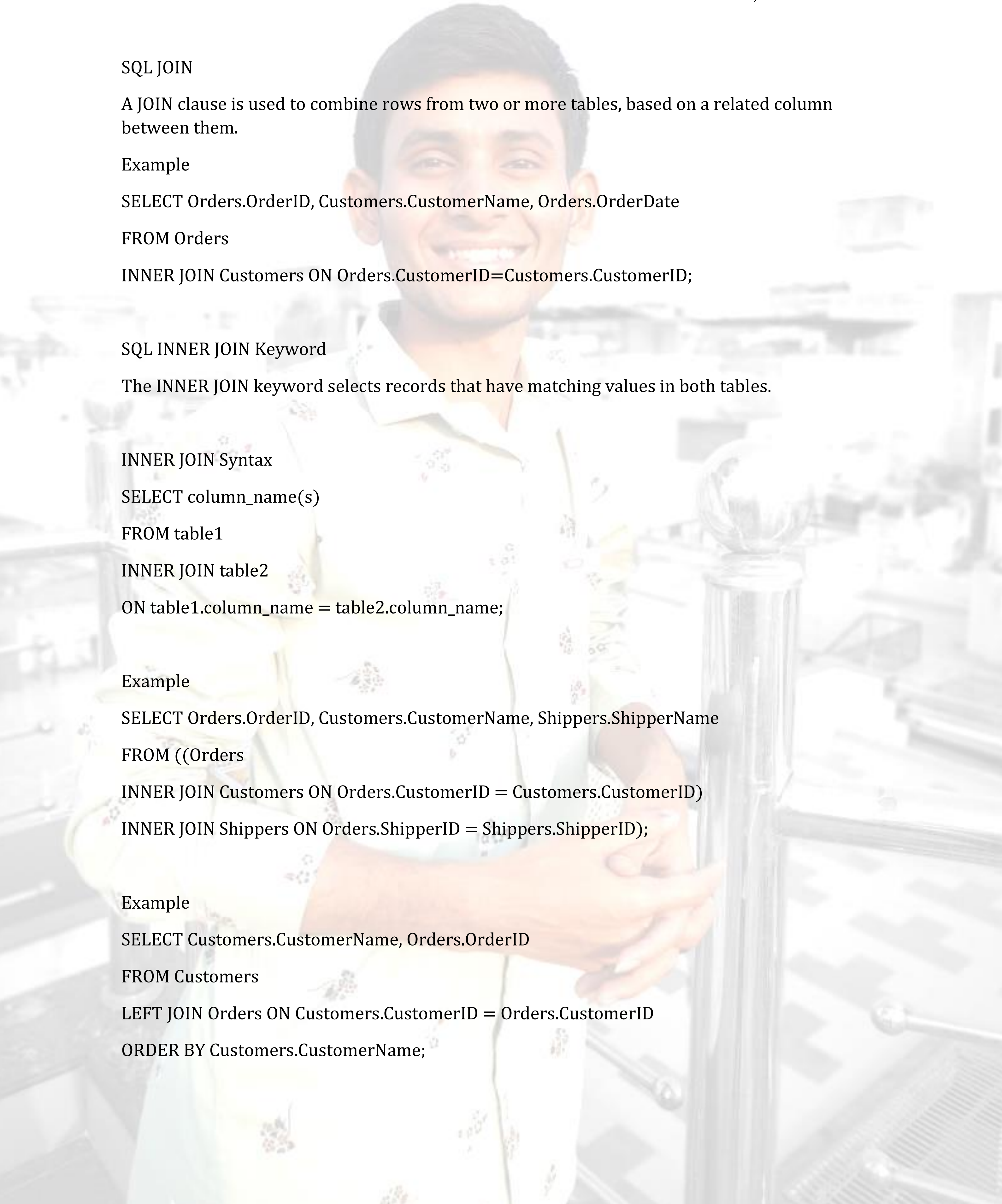
Alias Table Syntax

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Example

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Example



```
SELECT o.OrderID, o.OrderDate, c.CustomerName

FROM Customers AS c, Orders AS o

WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders

INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

SQL INNER JOIN Keyword

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

INNER JOIN Syntax

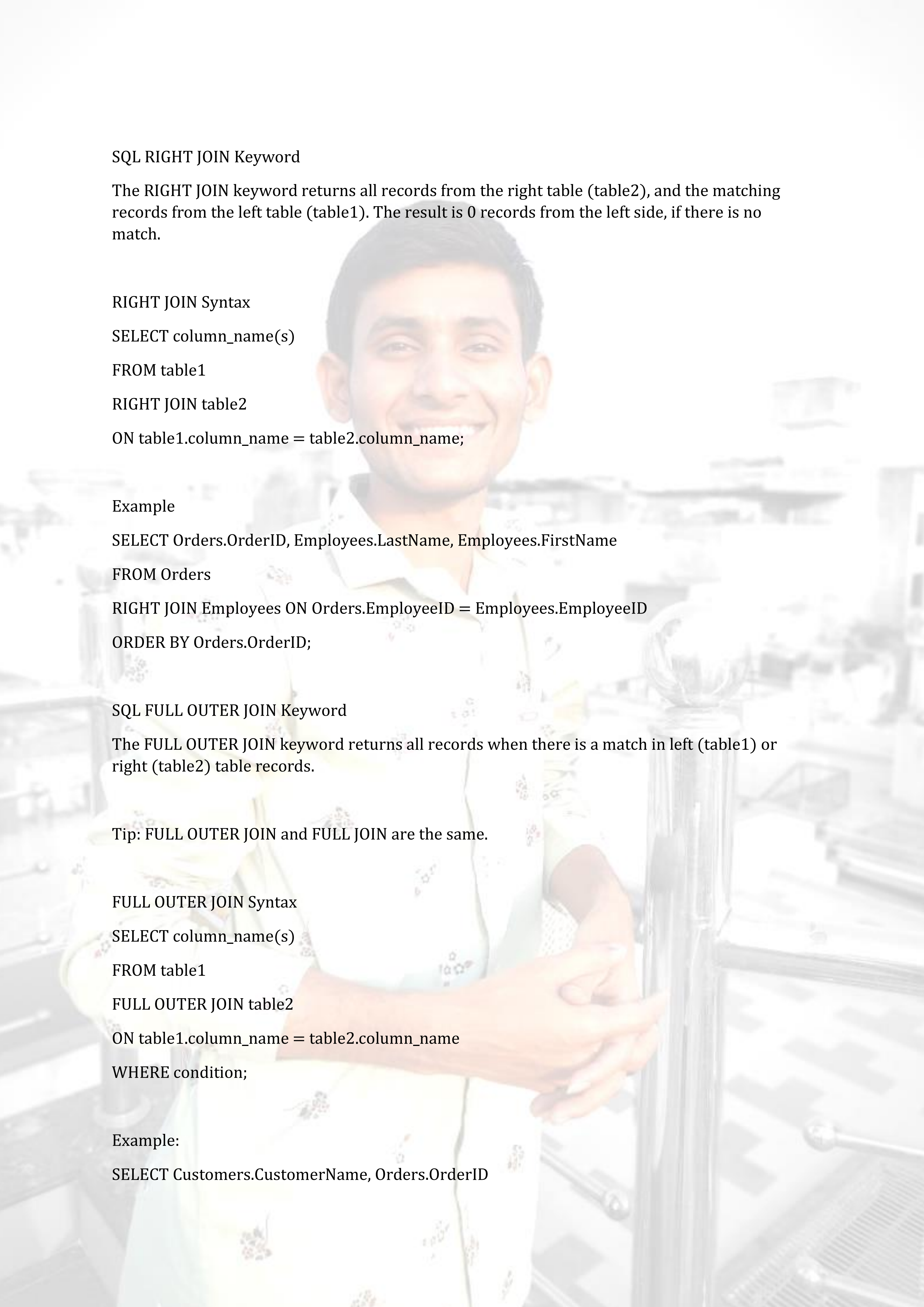
```
SELECT column_name(s)

FROM table1

INNER JOIN table2

ON table1.column_name = table2.column_name;
```

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName

FROM ((Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)

INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

Example

```
SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;
```



SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Example

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

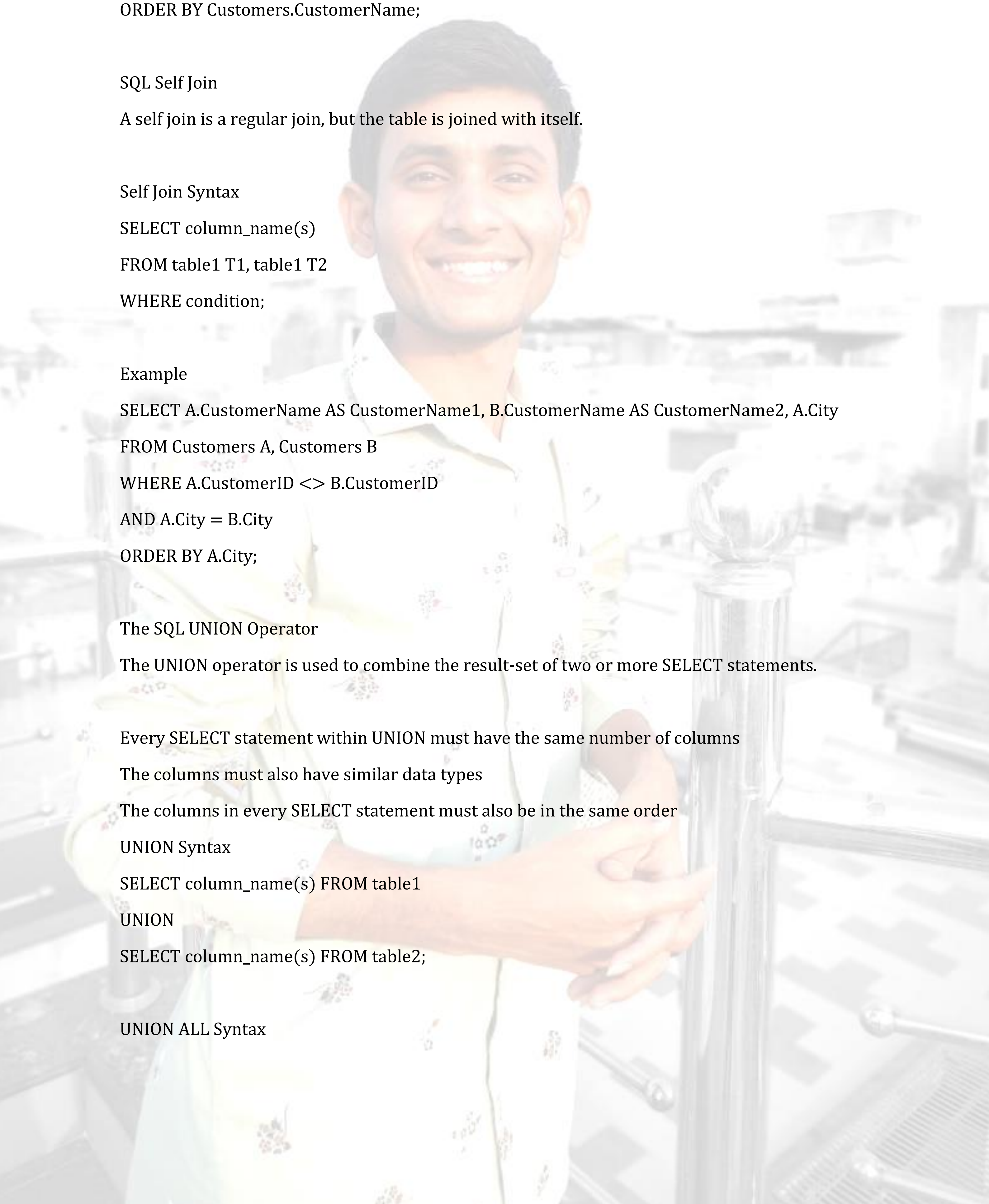
Tip: FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Example:

```
SELECT Customers.CustomerName, Orders.OrderID
```



```
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

SQL Self Join

A self join is a regular join, but the table is joined with itself.

Self Join Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

Example

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Every SELECT statement within UNION must have the same number of columns

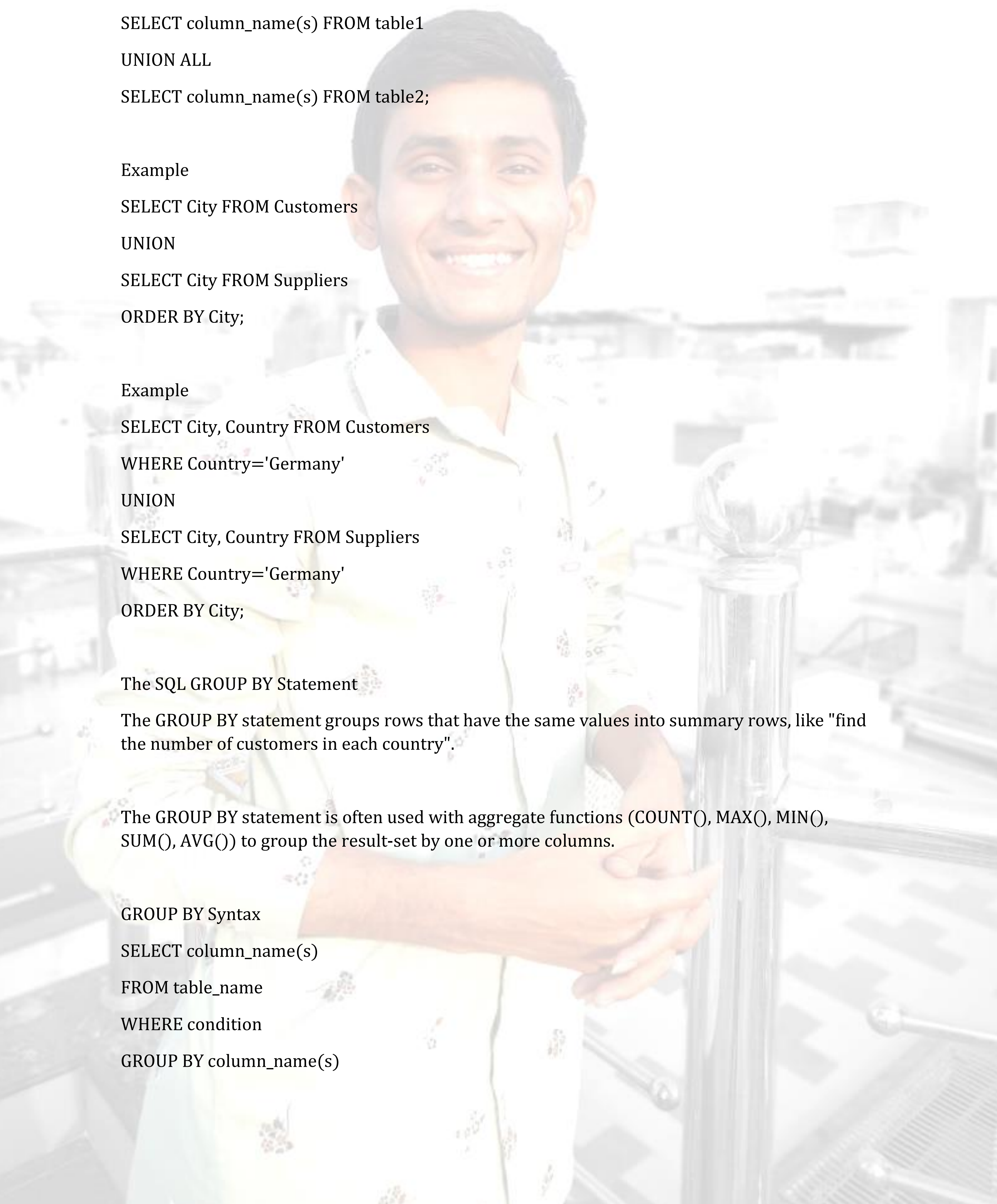
The columns must also have similar data types

The columns in every SELECT statement must also be in the same order

UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION ALL Syntax



The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1

UNION ALL

SELECT column_name(s) FROM table2;
```

Example

```
SELECT City FROM Customers

UNION

SELECT City FROM Suppliers

ORDER BY City;
```

Example

```
SELECT City, Country FROM Customers

WHERE Country='Germany'

UNION

SELECT City, Country FROM Suppliers

WHERE Country='Germany'

ORDER BY City;
```

The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)

FROM table_name

WHERE condition

GROUP BY column_name(s)
```

```
ORDER BY column_name(s);
```

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

Example

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

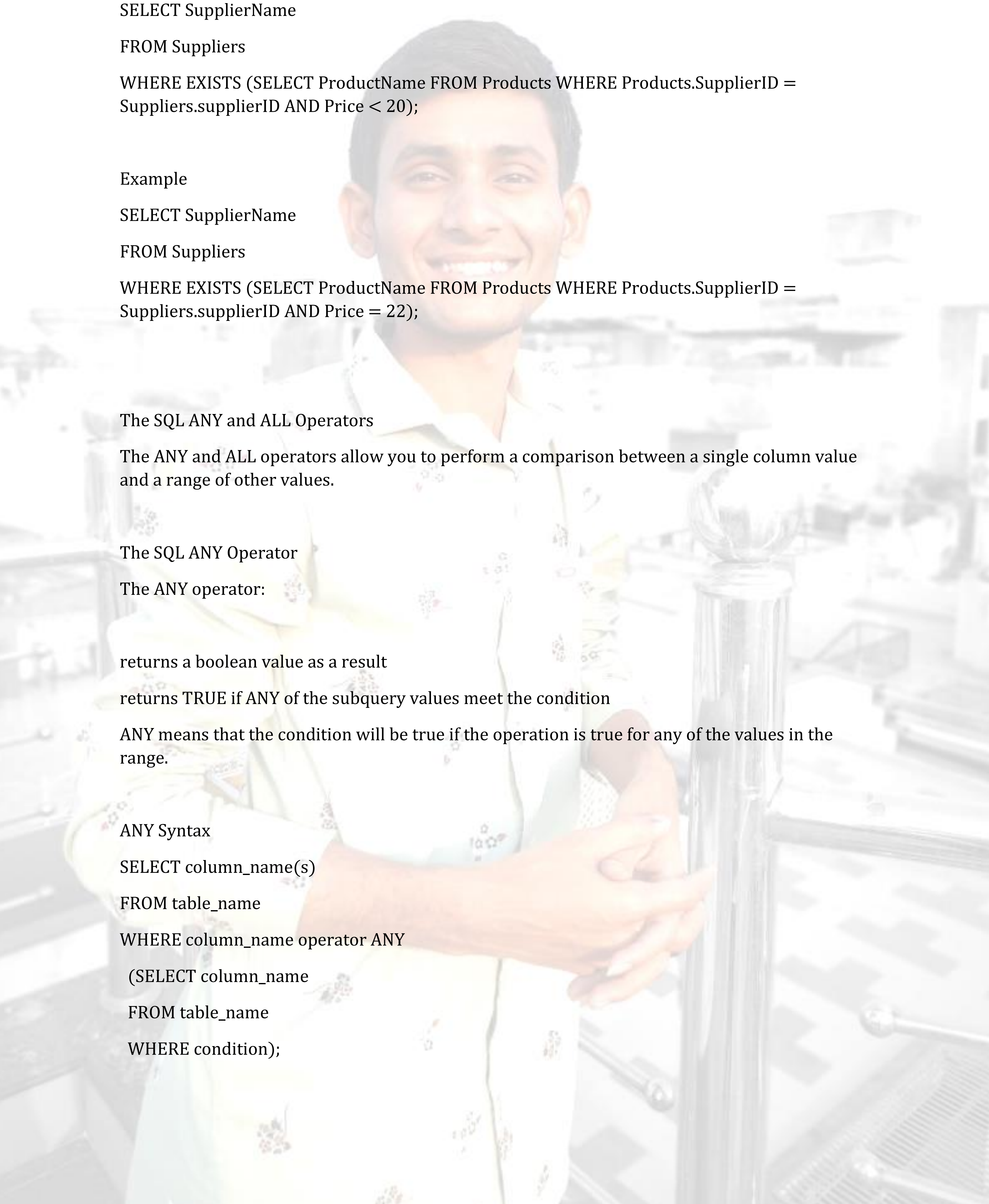
The SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```



Example

```
SELECT SupplierName

FROM Suppliers

WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND Price < 20);
```

Example

```
SELECT SupplierName

FROM Suppliers

WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND Price = 22);
```

The SQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

The SQL ANY Operator

The ANY operator:

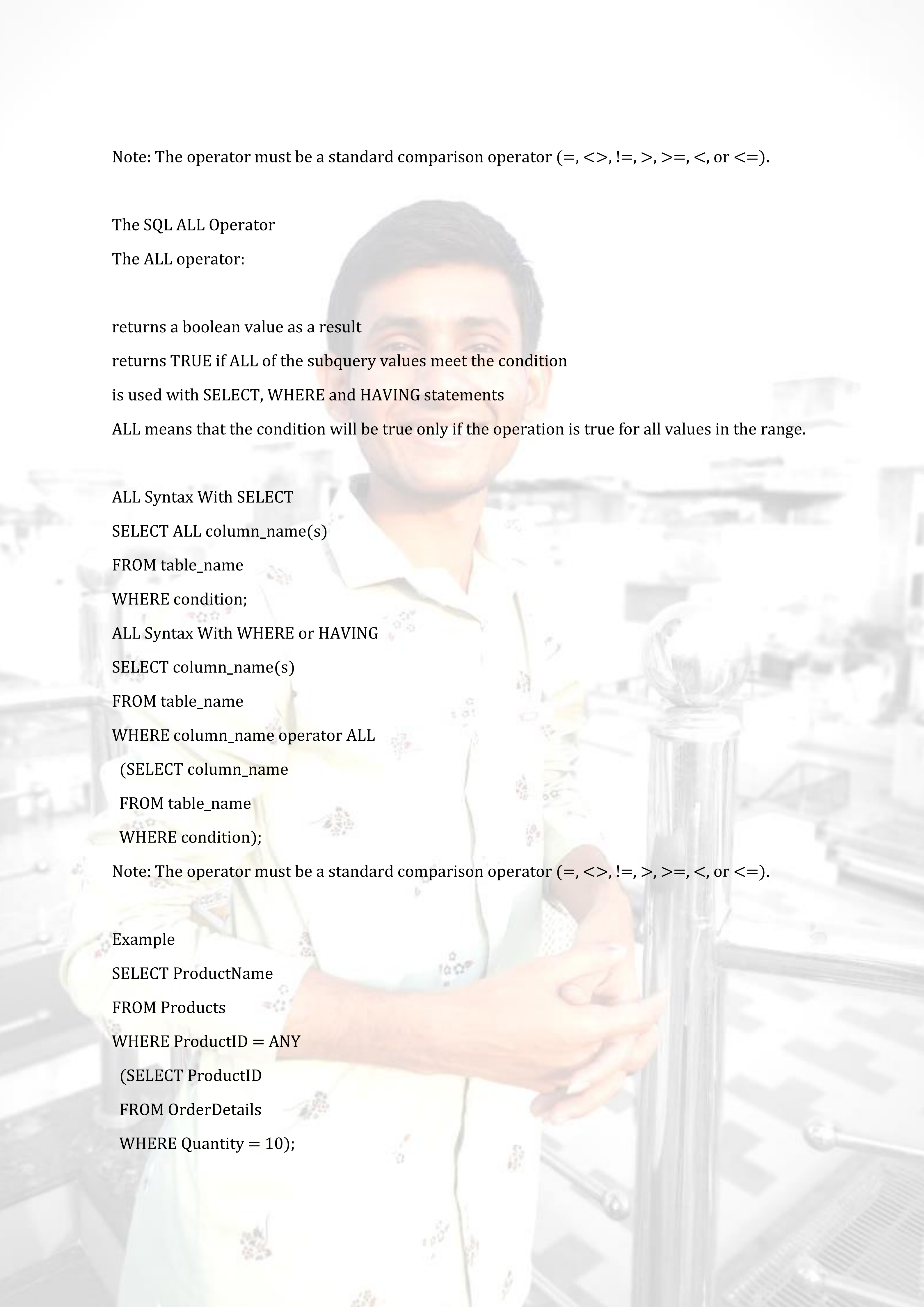
returns a boolean value as a result

returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax

```
SELECT column_name(s)

FROM table_name

WHERE column_name operator ANY

  (SELECT column_name

  FROM table_name

  WHERE condition);
```



Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

The SQL ALL Operator

The ALL operator:

returns a boolean value as a result

returns TRUE if ALL of the subquery values meet the condition

is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax With SELECT

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

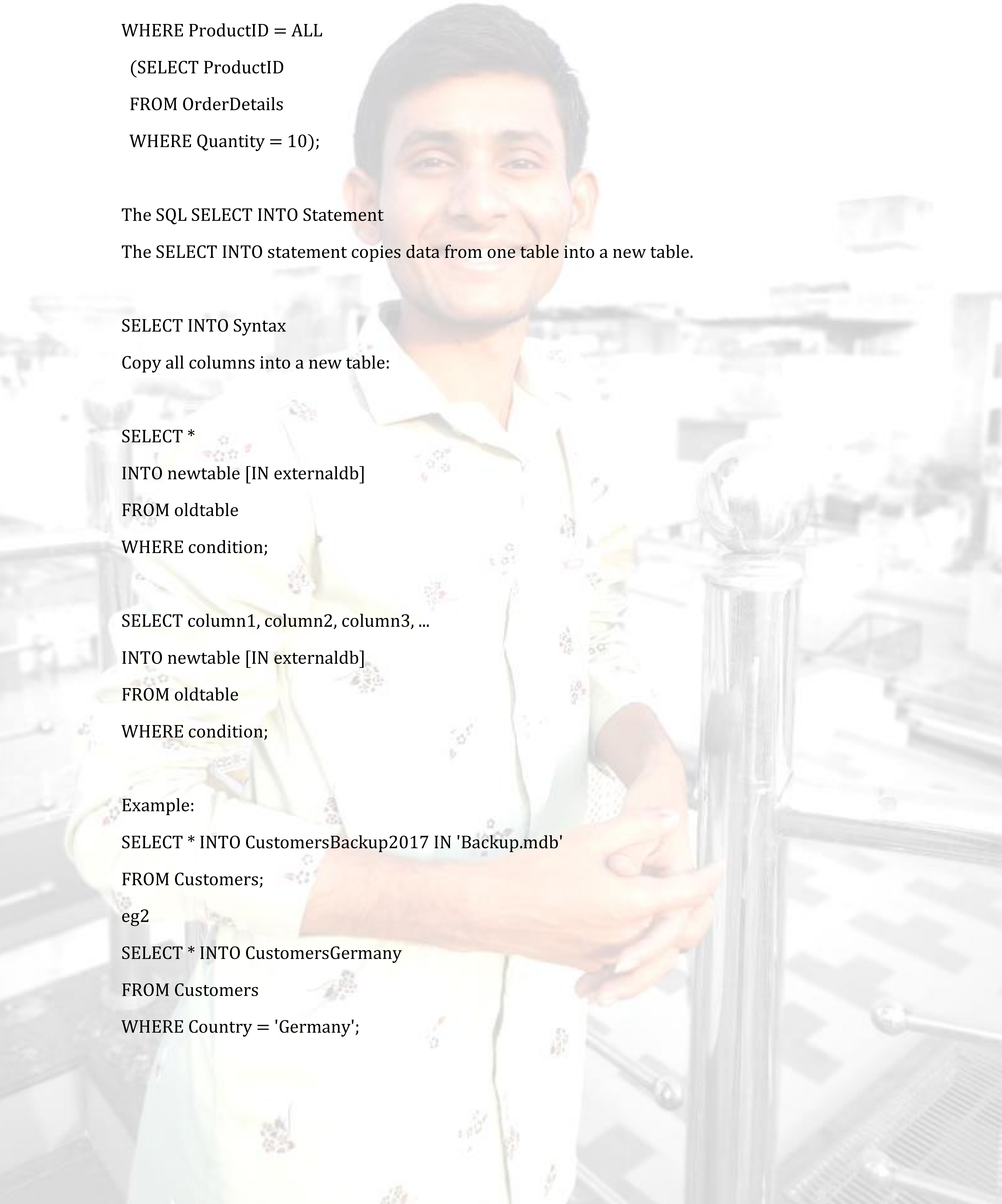
ALL Syntax With WHERE or HAVING

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

Example

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```



Example

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

The SQL SELECT INTO Statement

The SELECT INTO statement copies data from one table into a new table.

SELECT INTO Syntax

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

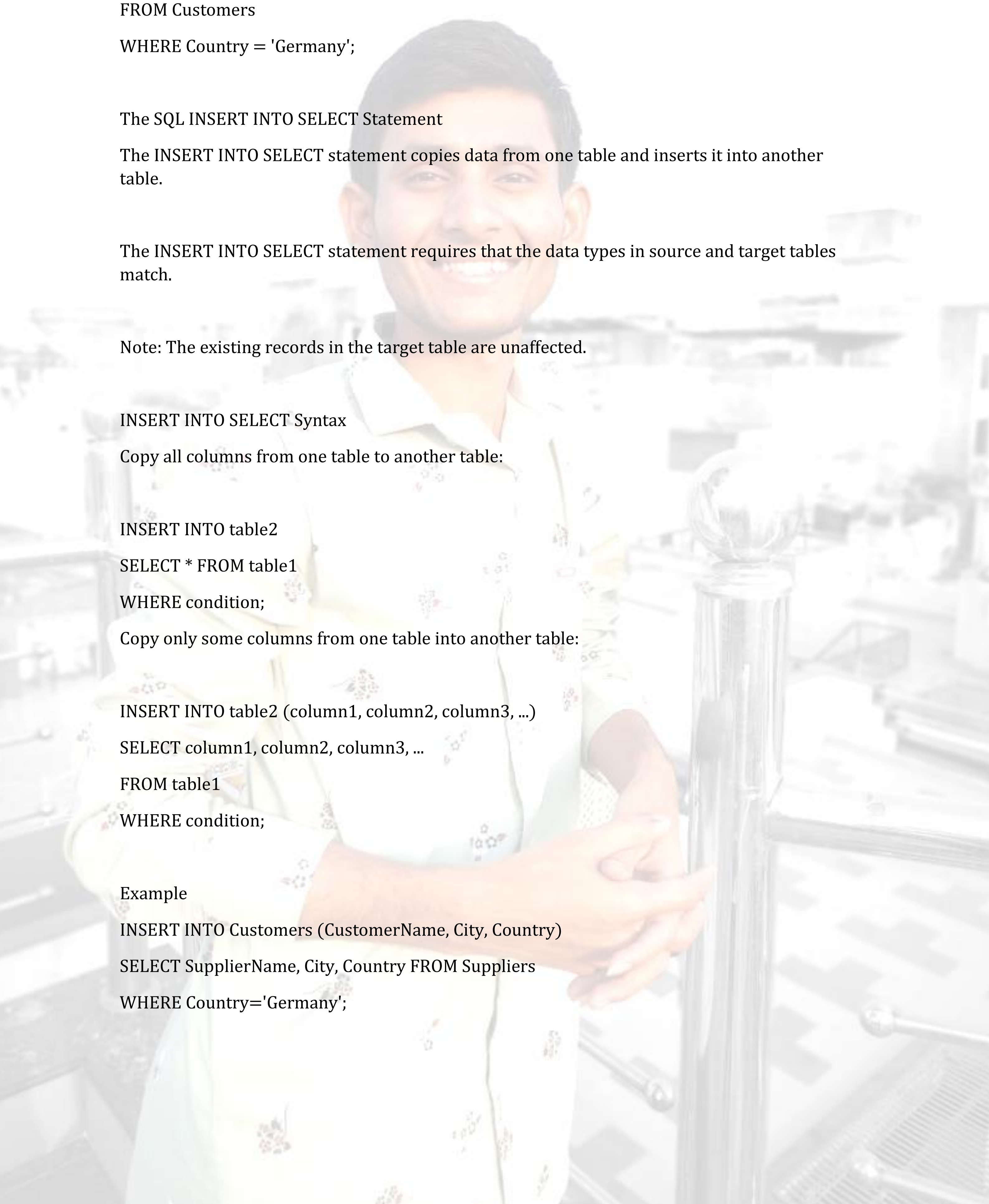
```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Example:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

eg2

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```



eg3

```
SELECT * INTO CustomersGermany

FROM Customers

WHERE Country = 'Germany';
```

The SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in source and target tables match.

Note: The existing records in the target table are unaffected.

INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

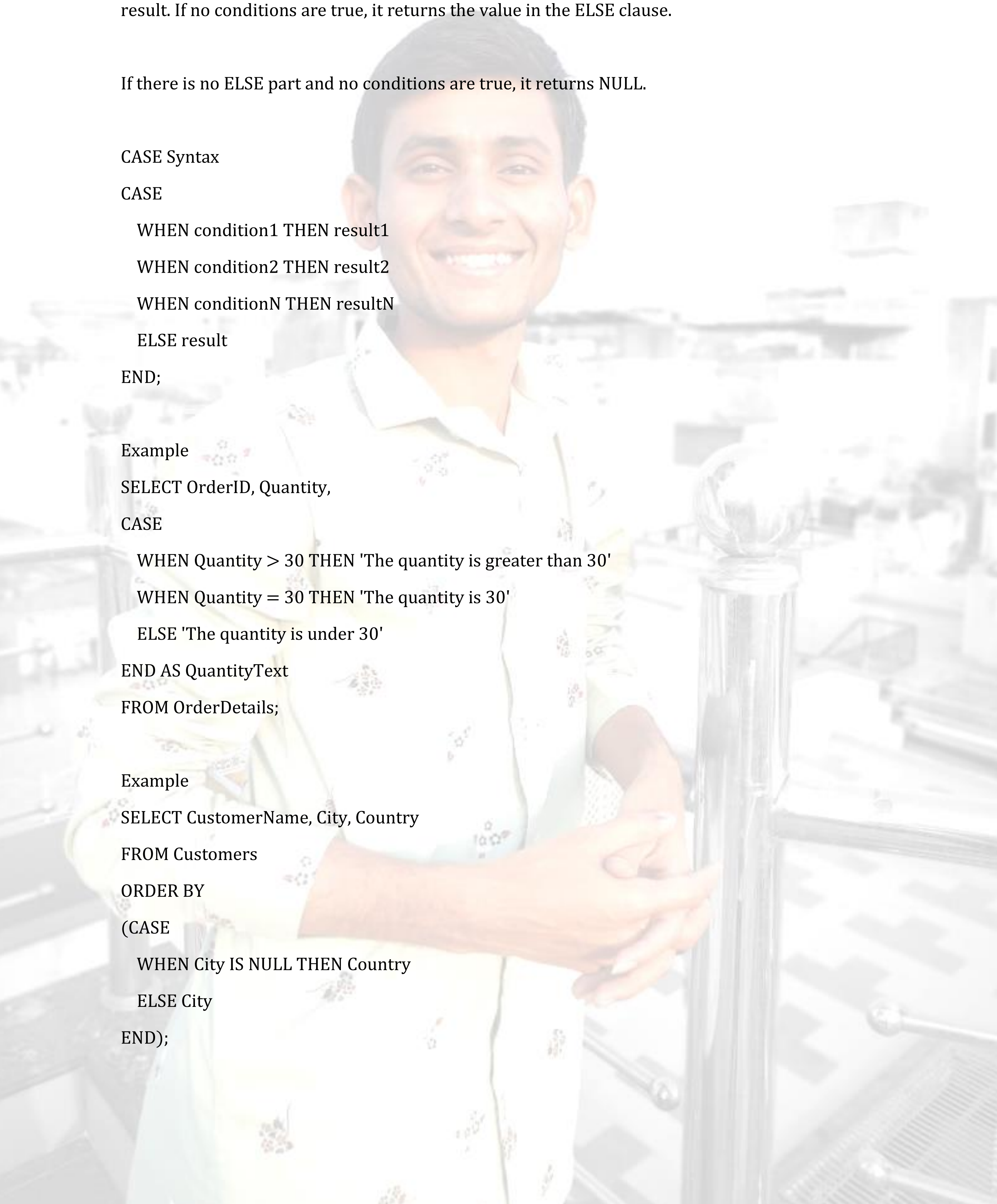
```
INSERT INTO table2

SELECT * FROM table1

WHERE condition;
```

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)

SELECT column1, column2, column3, ...

FROM table1

WHERE condition;
```

Example

```
INSERT INTO Customers (CustomerName, City, Country)

SELECT SupplierName, City, Country FROM Suppliers

WHERE Country='Germany';
```



The SQL CASE Statement

The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

CASE Syntax

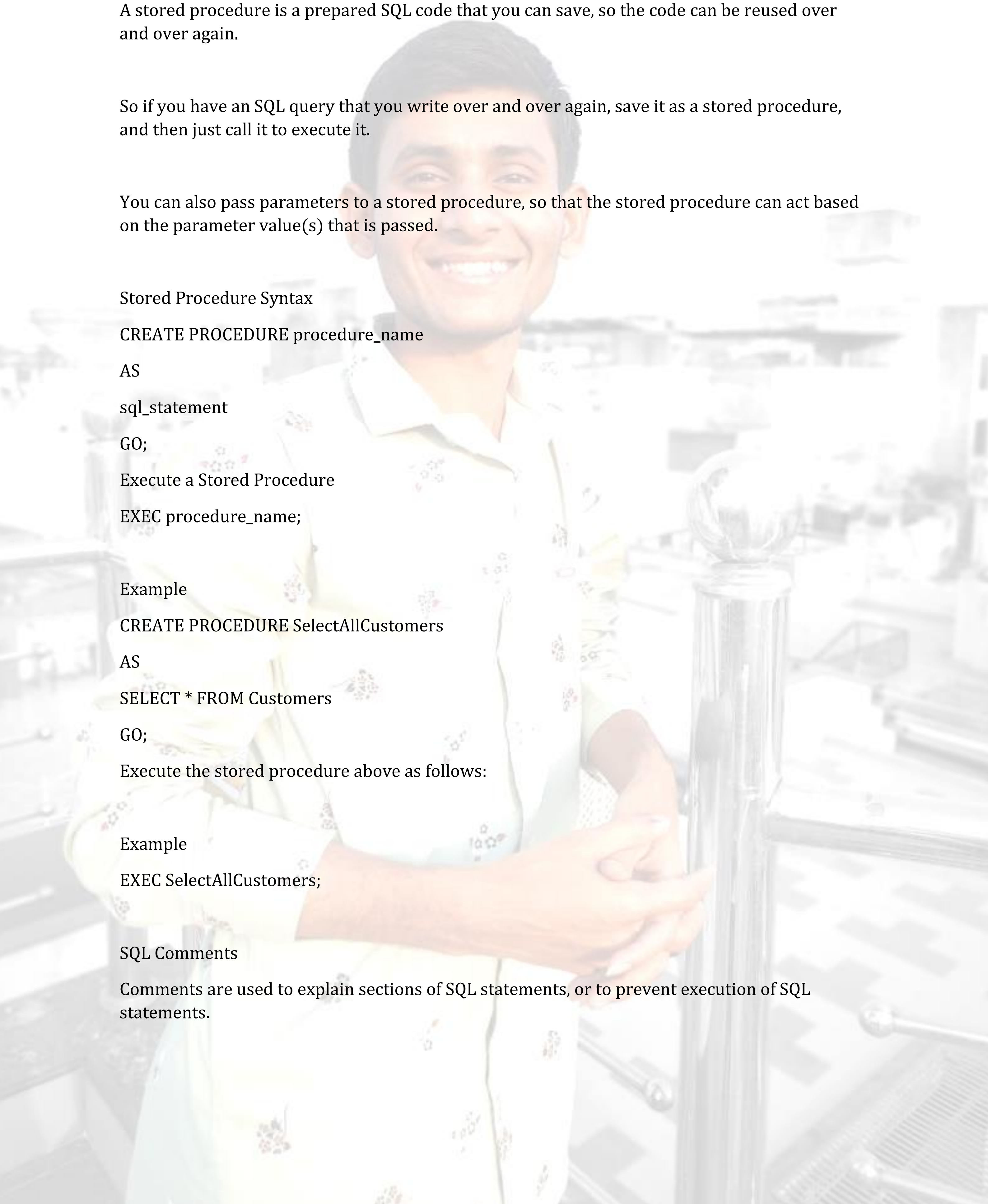
```
CASE
  WHEN condition1 THEN result1
  WHEN condition2 THEN result2
  WHEN conditionN THEN resultN
  ELSE result
END;
```

Example

```
SELECT OrderID, Quantity,
CASE
  WHEN Quantity > 30 THEN 'The quantity is greater than 30'
  WHEN Quantity = 30 THEN 'The quantity is 30'
  ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

Example

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
  WHEN City IS NULL THEN Country
  ELSE City
END);
```



What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name

AS

sql_statement

GO;
```

Execute a Stored Procedure

```
EXEC procedure_name;
```

Example

```
CREATE PROCEDURE SelectAllCustomers

AS

SELECT * FROM Customers

GO;
```

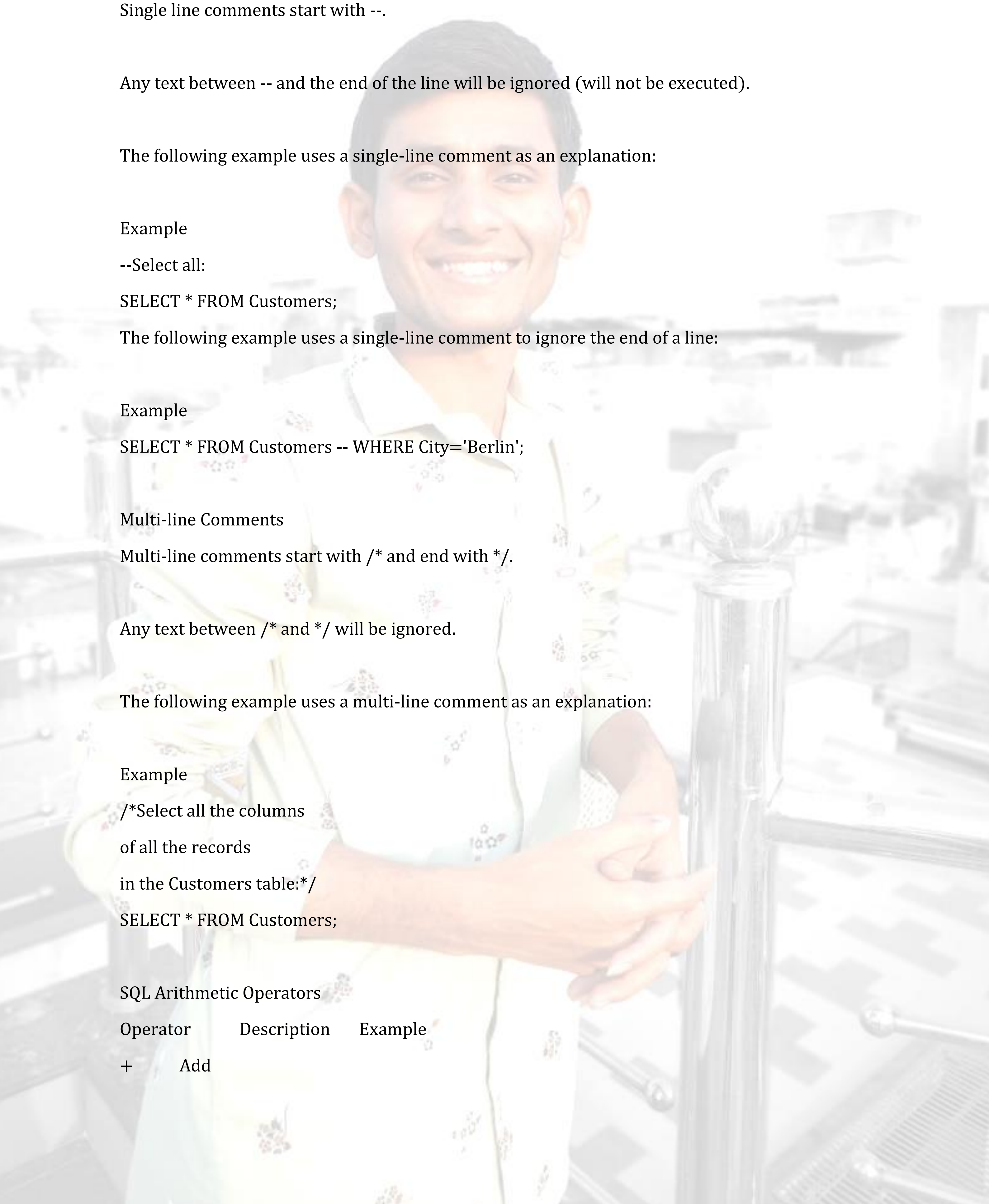
Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers;
```

SQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.



Single Line Comments

Single line comments start with --.

Any text between -- and the end of the line will be ignored (will not be executed).

The following example uses a single-line comment as an explanation:

Example

```
--Select all:
SELECT * FROM Customers;
```

The following example uses a single-line comment to ignore the end of a line:

Example

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

Multi-line Comments

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored.

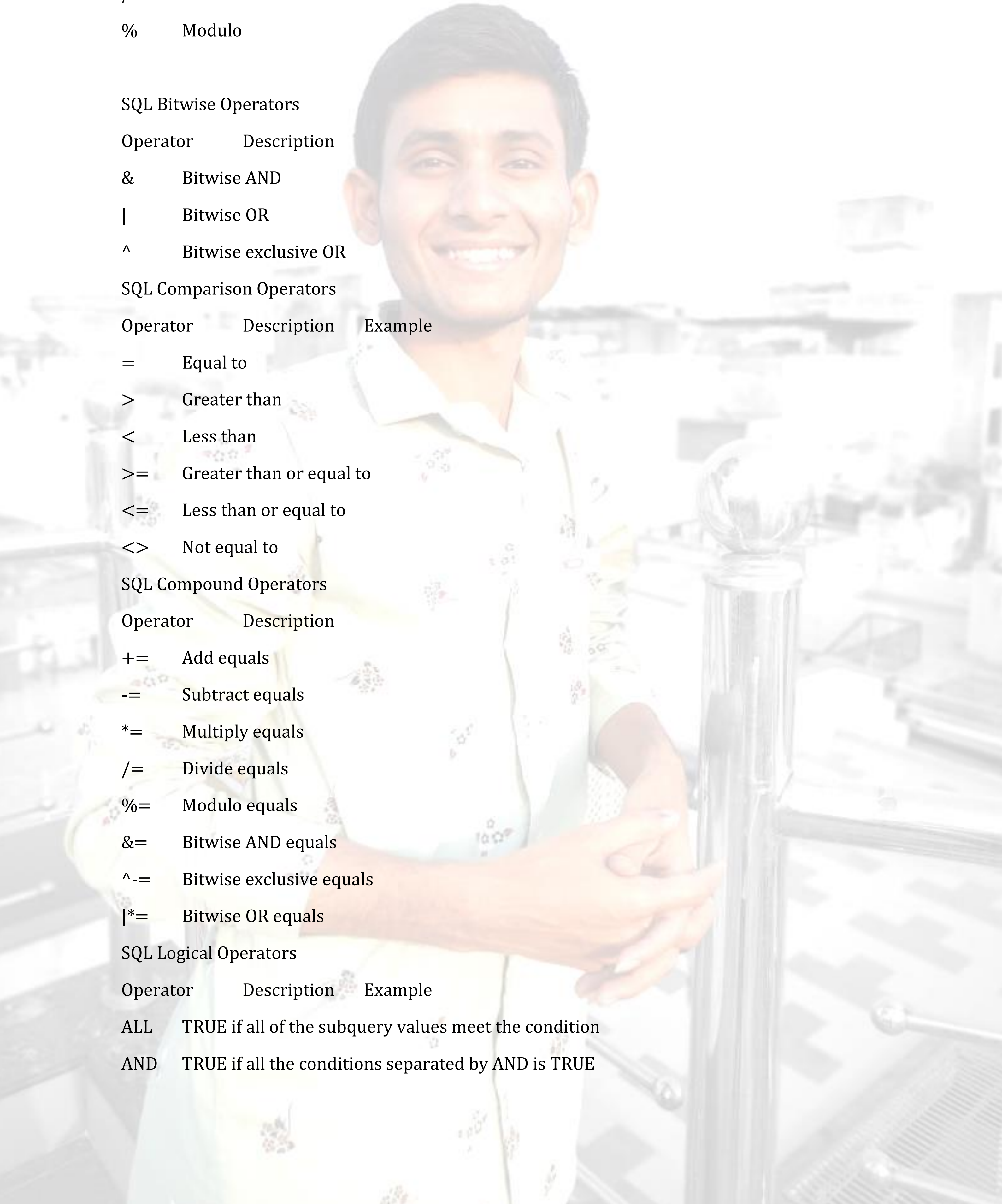
The following example uses a multi-line comment as an explanation:

Example

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

SQL Arithmetic Operators

Operator	Description	Example
+	Add	



-	Subtract
*	Multiply
/	Divide
%	Modulo

SQL Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators

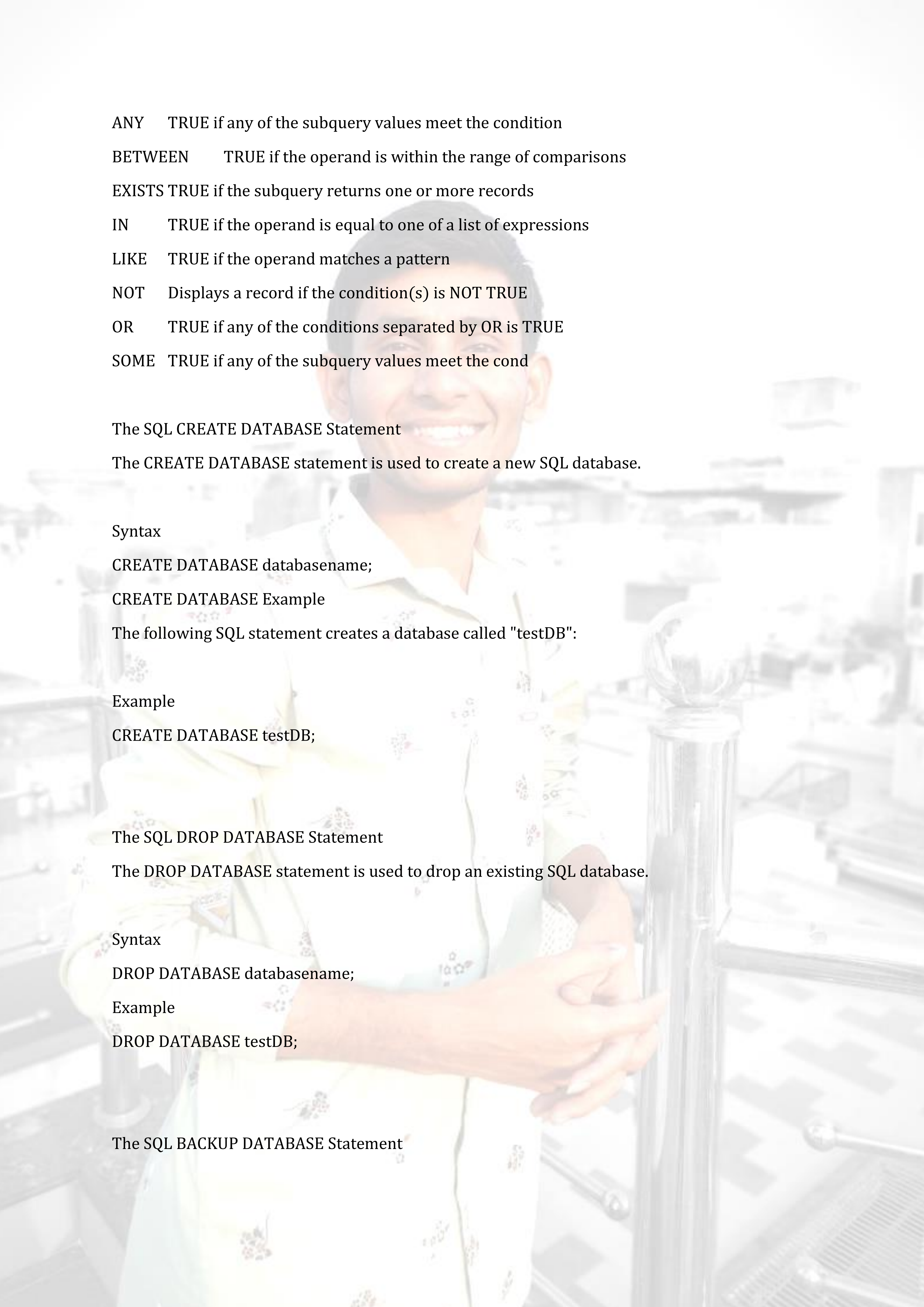
Operator	Description	Example
=	Equal to	
>	Greater than	
<	Less than	
>=	Greater than or equal to	
<=	Less than or equal to	
<>	Not equal to	

SQL Compound Operators

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

SQL Logical Operators

Operator	Description	Example
ALL	TRUE if all of the subquery values meet the condition	
AND	TRUE if all the conditions separated by AND is TRUE	



ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the cond

The SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.

Syntax

```
CREATE DATABASE databasename;
```

CREATE DATABASE Example

The following SQL statement creates a database called "testDB":

Example

```
CREATE DATABASE testDB;
```

The SQL DROP DATABASE Statement

The DROP DATABASE statement is used to drop an existing SQL database.

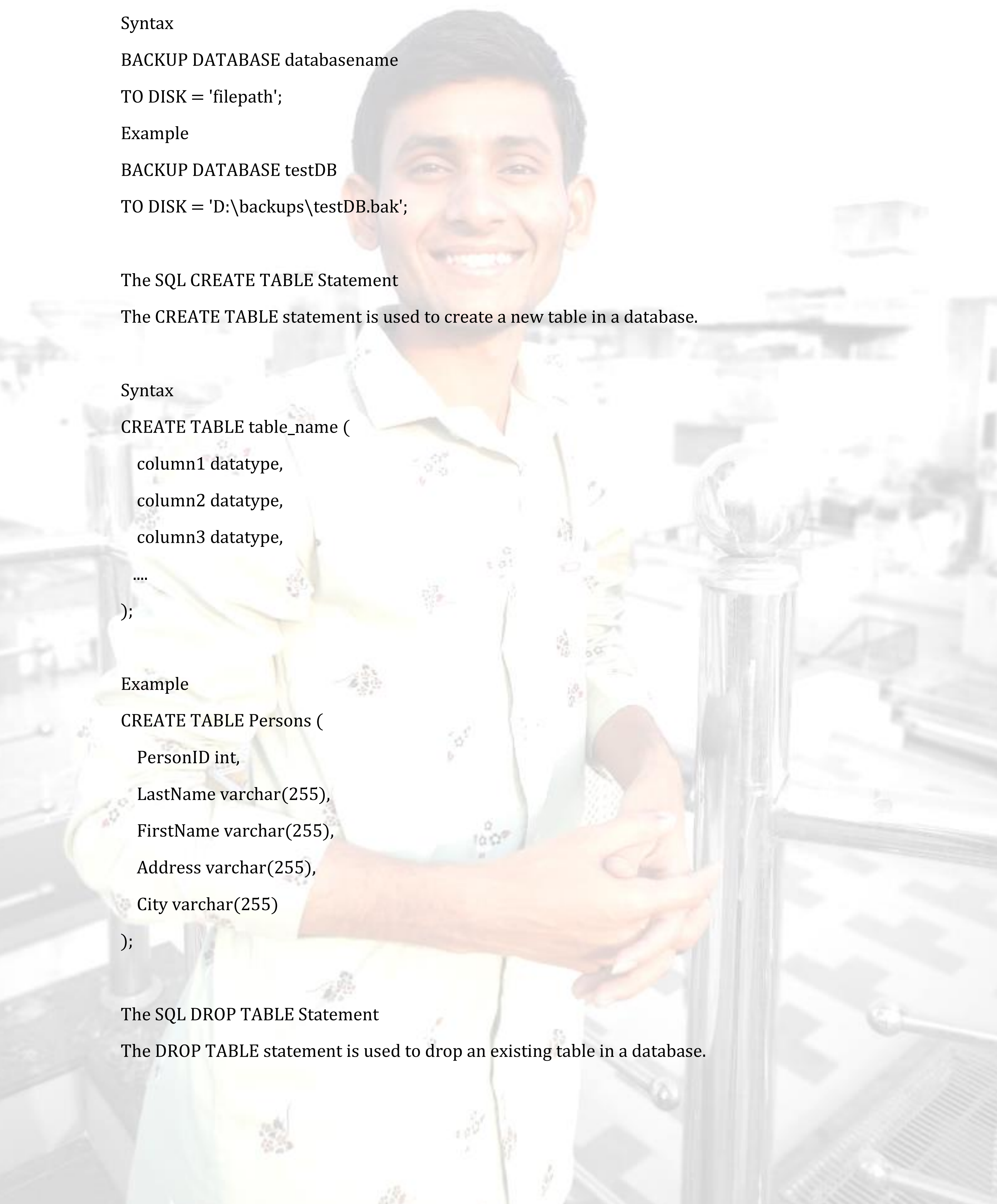
Syntax

```
DROP DATABASE databasename;
```

Example

```
DROP DATABASE testDB;
```

The SQL BACKUP DATABASE Statement



The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax

```
BACKUP DATABASE databasename
TO DISK = 'filepath';
```

Example

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';
```

The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

Syntax

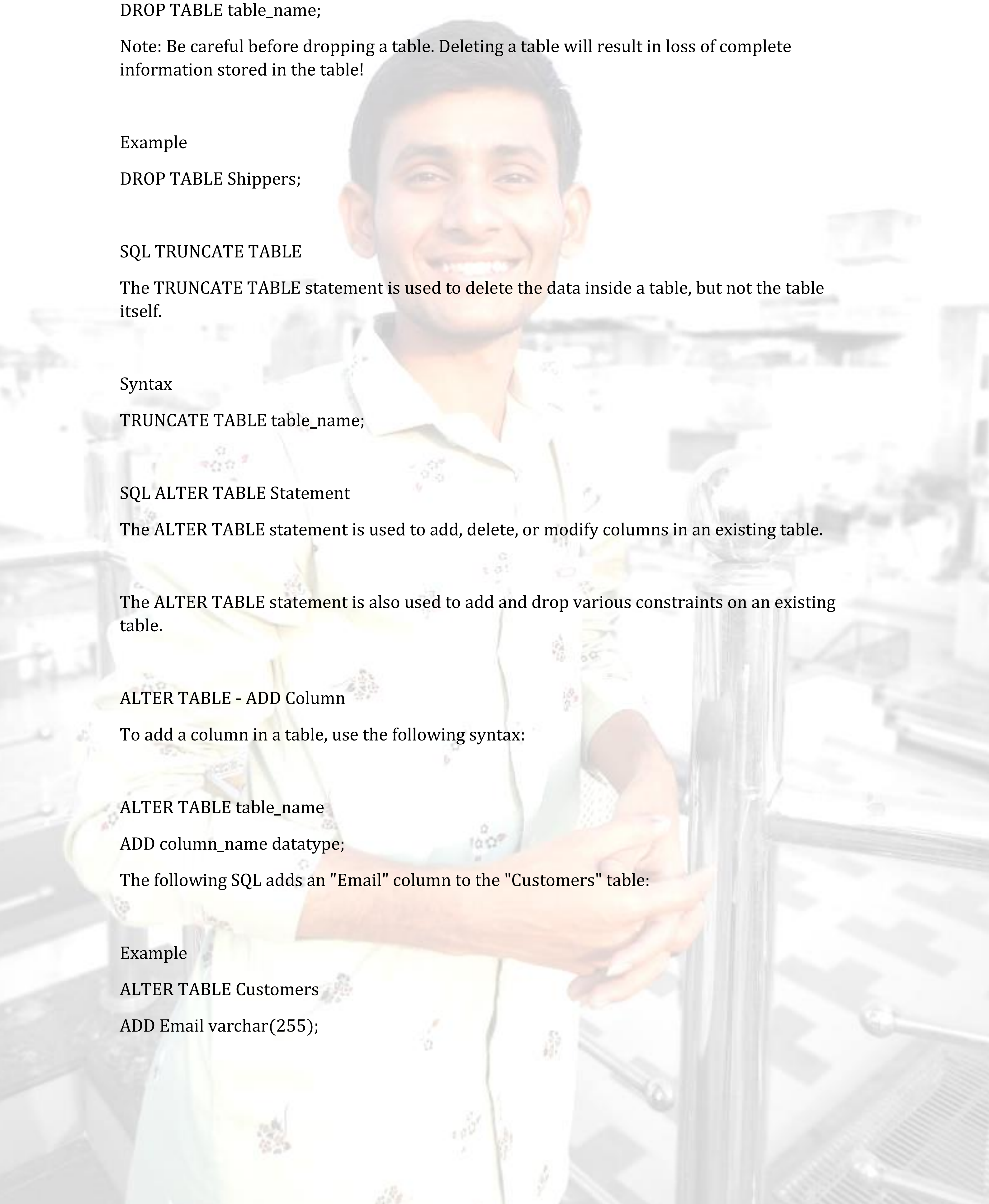
```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

Example

```
CREATE TABLE Persons (
   PersonID int,
   LastName varchar(255),
   FirstName varchar(255),
   Address varchar(255),
   City varchar(255)
);
```

The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.



Syntax

```
DROP TABLE table_name;
```

Note: Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

Example

```
DROP TABLE Shippers;
```

SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

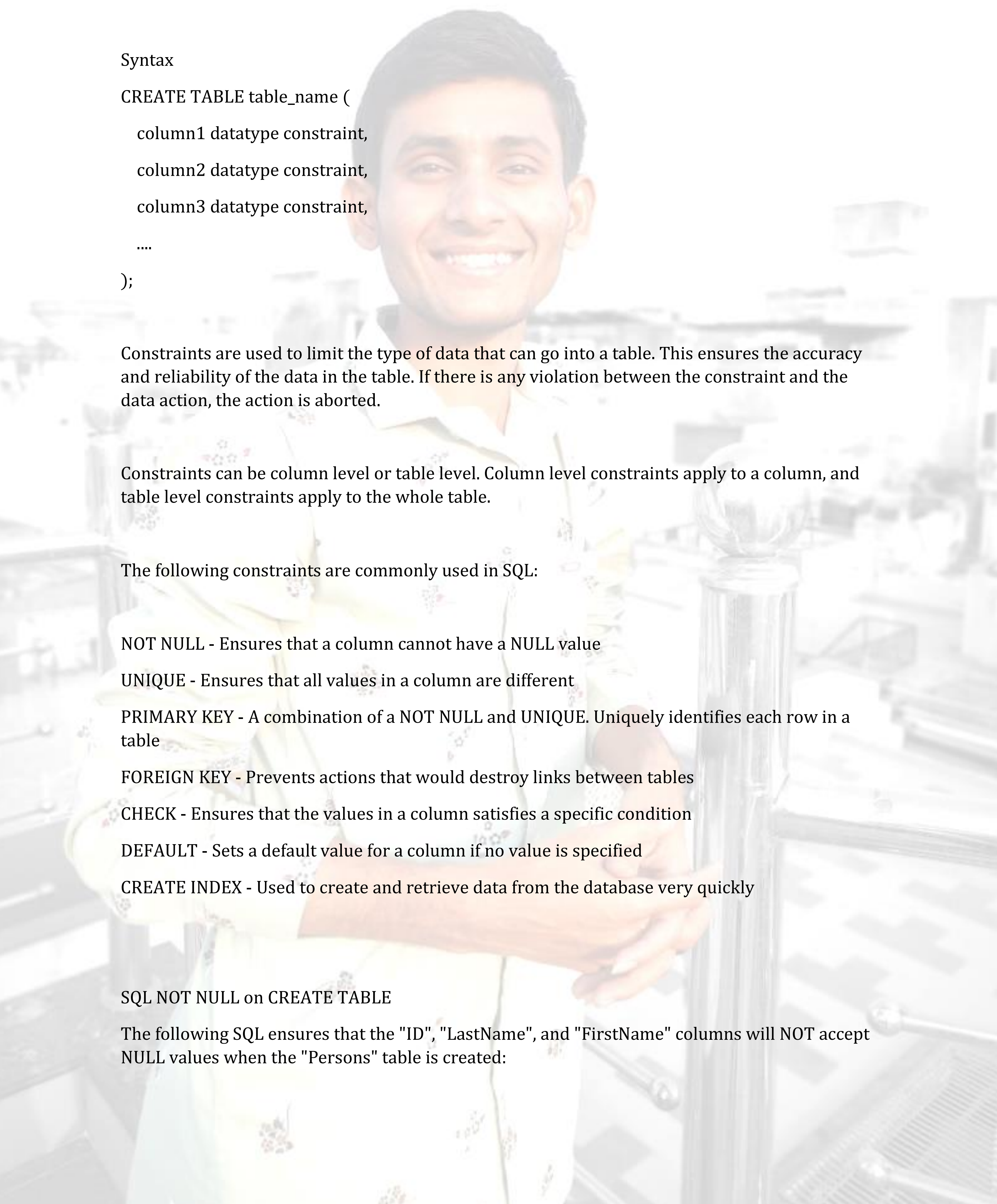
To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

The following SQL adds an "Email" column to the "Customers" table:

Example

```
ALTER TABLE Customers
ADD Email varchar(255);
```



SQL Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax

```
CREATE TABLE table_name (
  column1 datatype constraint,
  column2 datatype constraint,
  column3 datatype constraint,
  ....
);
```

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Prevents actions that would destroy links between tables

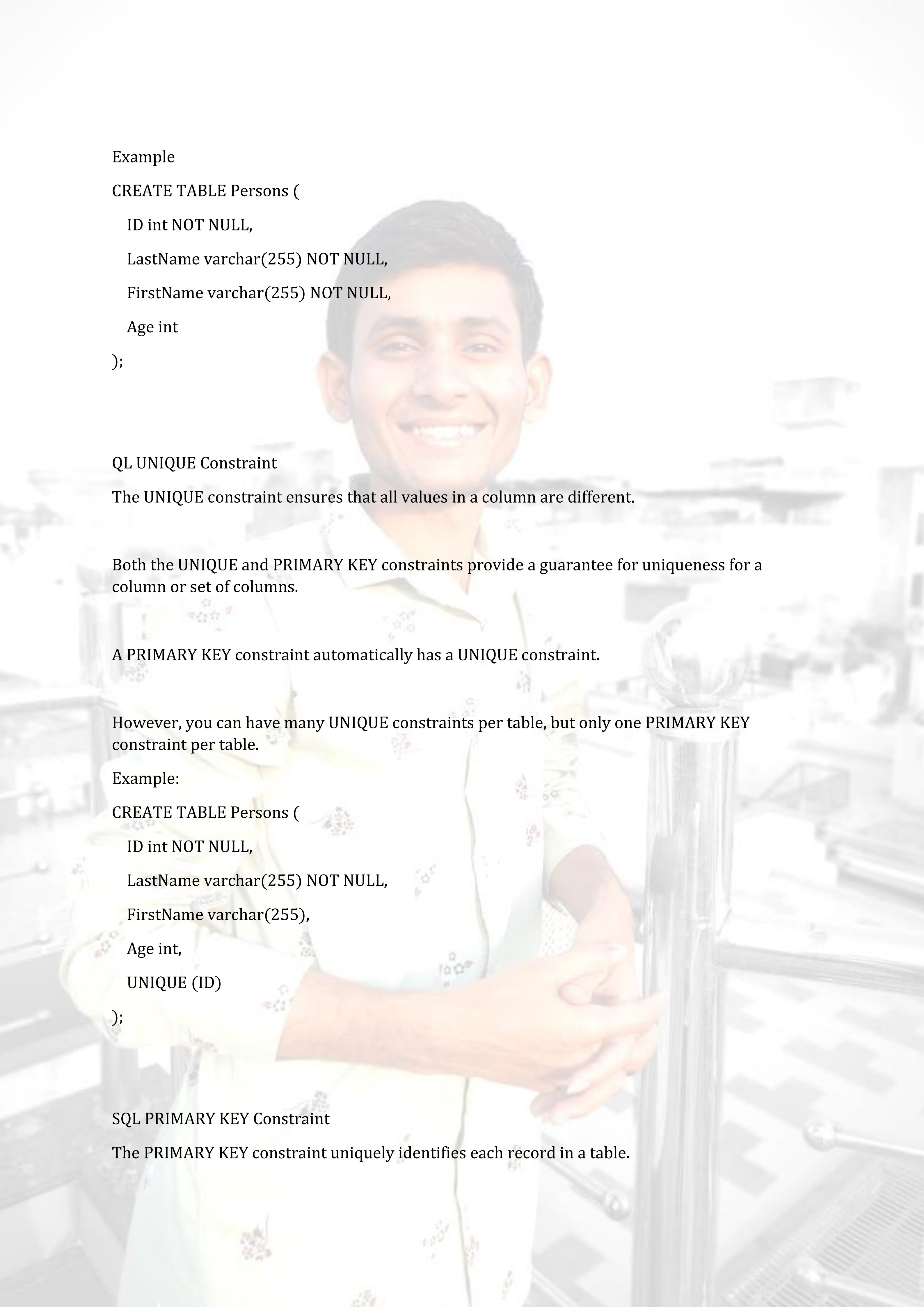
CHECK - Ensures that the values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column if no value is specified

CREATE INDEX - Used to create and retrieve data from the database very quickly

SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:



Example

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255) NOT NULL,
   Age int
);
```

QL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

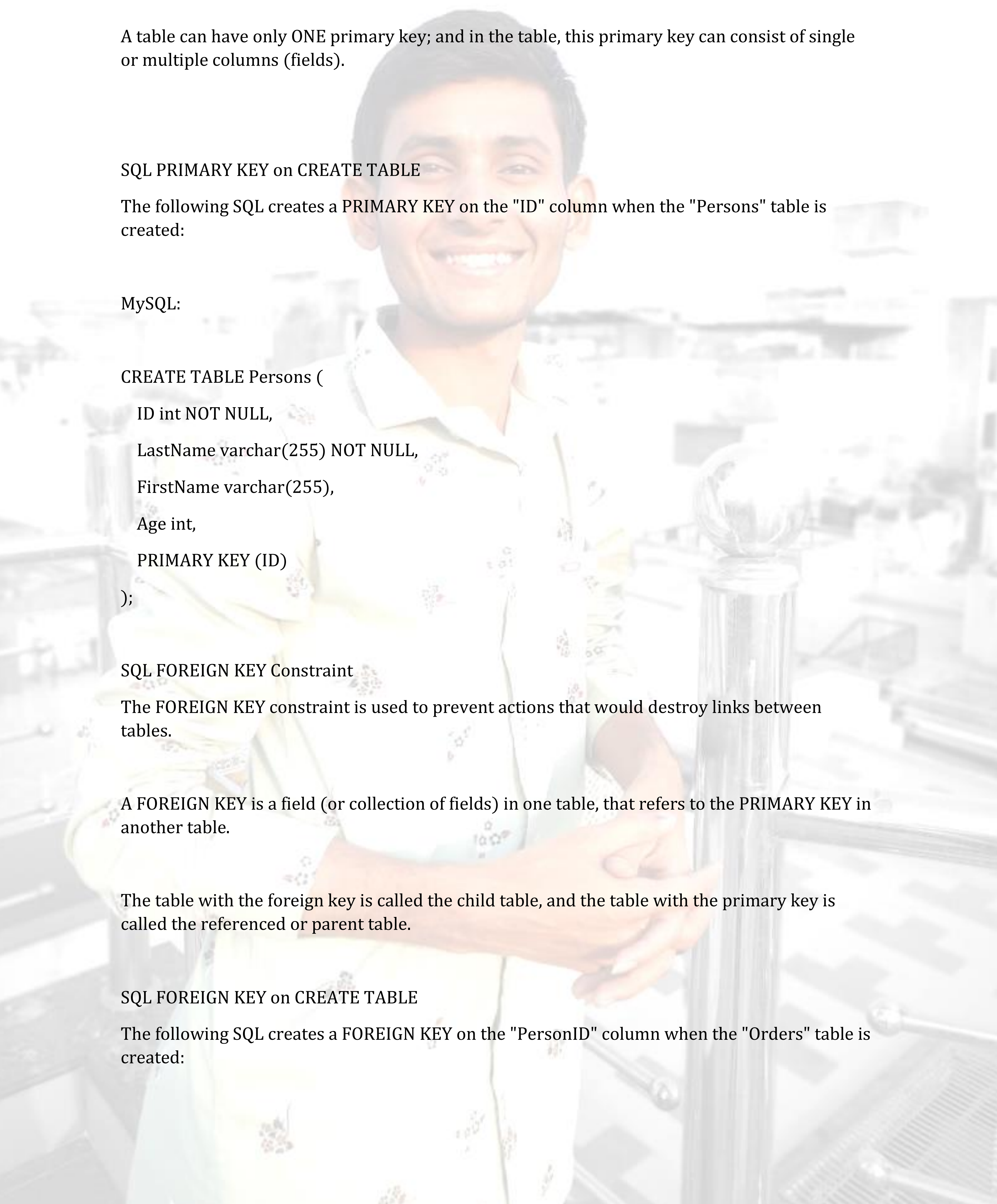
However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Example:

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   UNIQUE (ID)
);
```

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.



Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

MySQL:

```
CREATE TABLE Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  PRIMARY KEY (ID)
);
```

SQL FOREIGN KEY Constraint

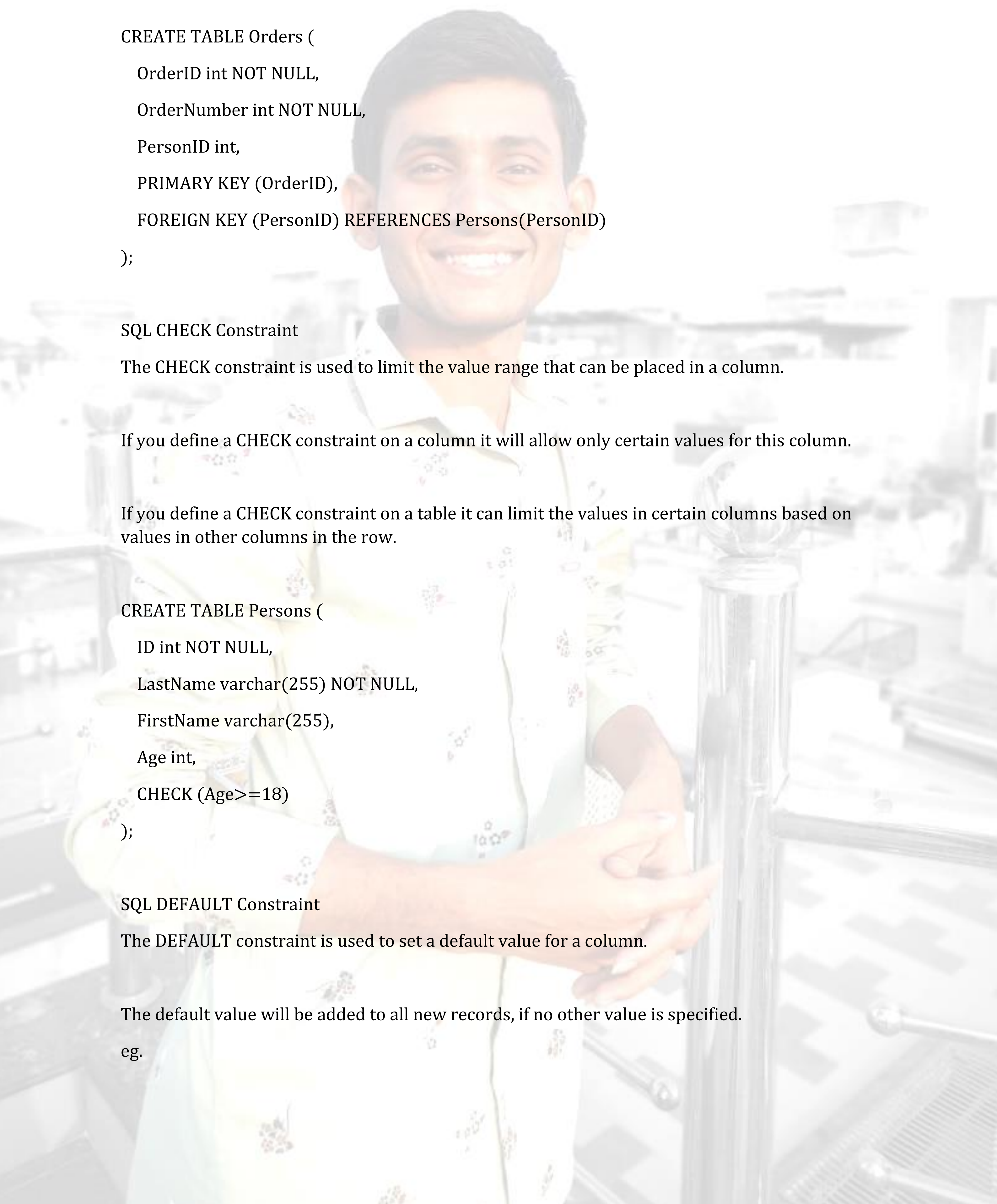
The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:



MySQL:

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

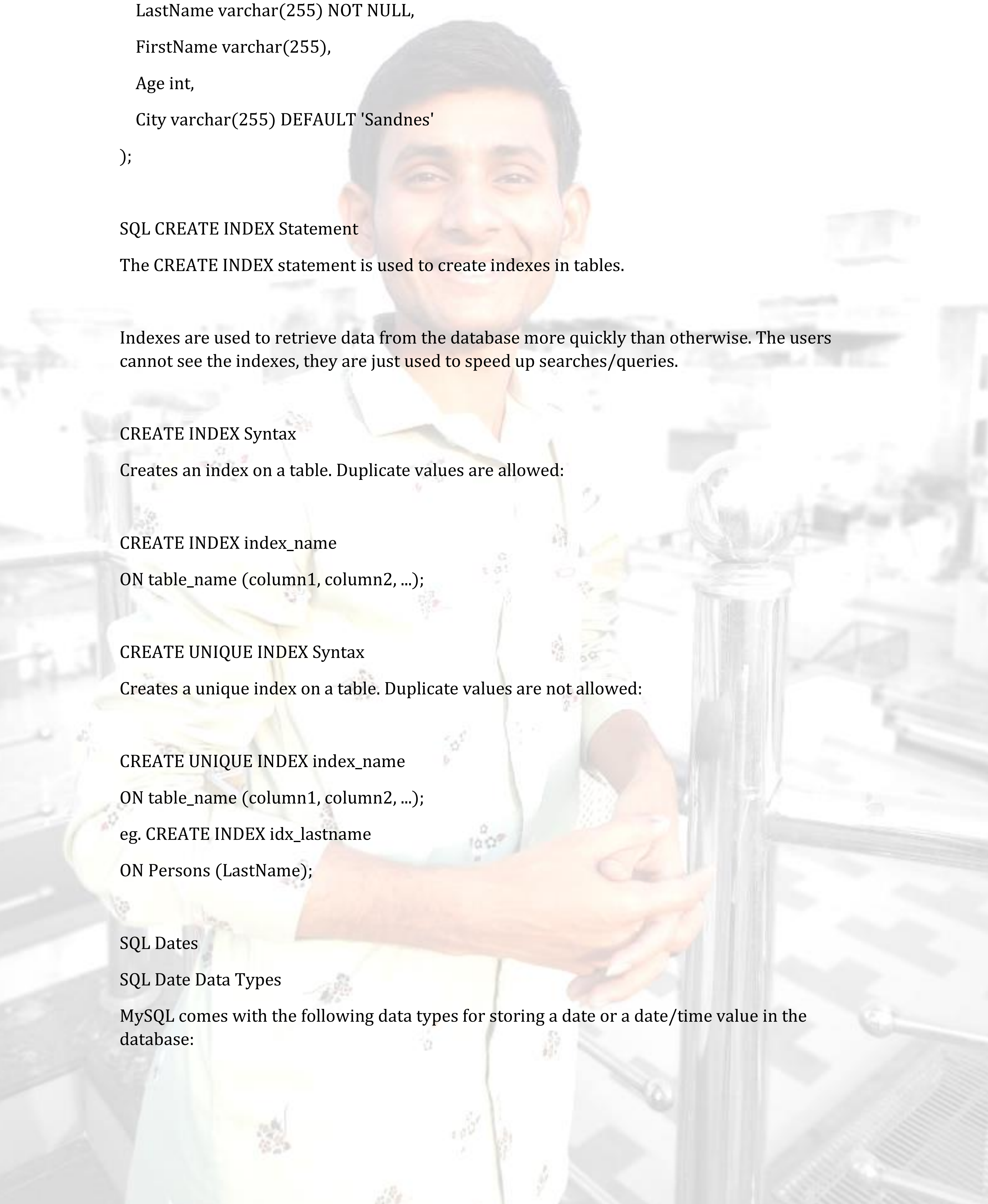
```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   CHECK (Age>=18)
);
```

SQL DEFAULT Constraint

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

eg.



```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   City varchar(255) DEFAULT 'Sandnes'
);
```

SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

CREATE UNIQUE INDEX Syntax

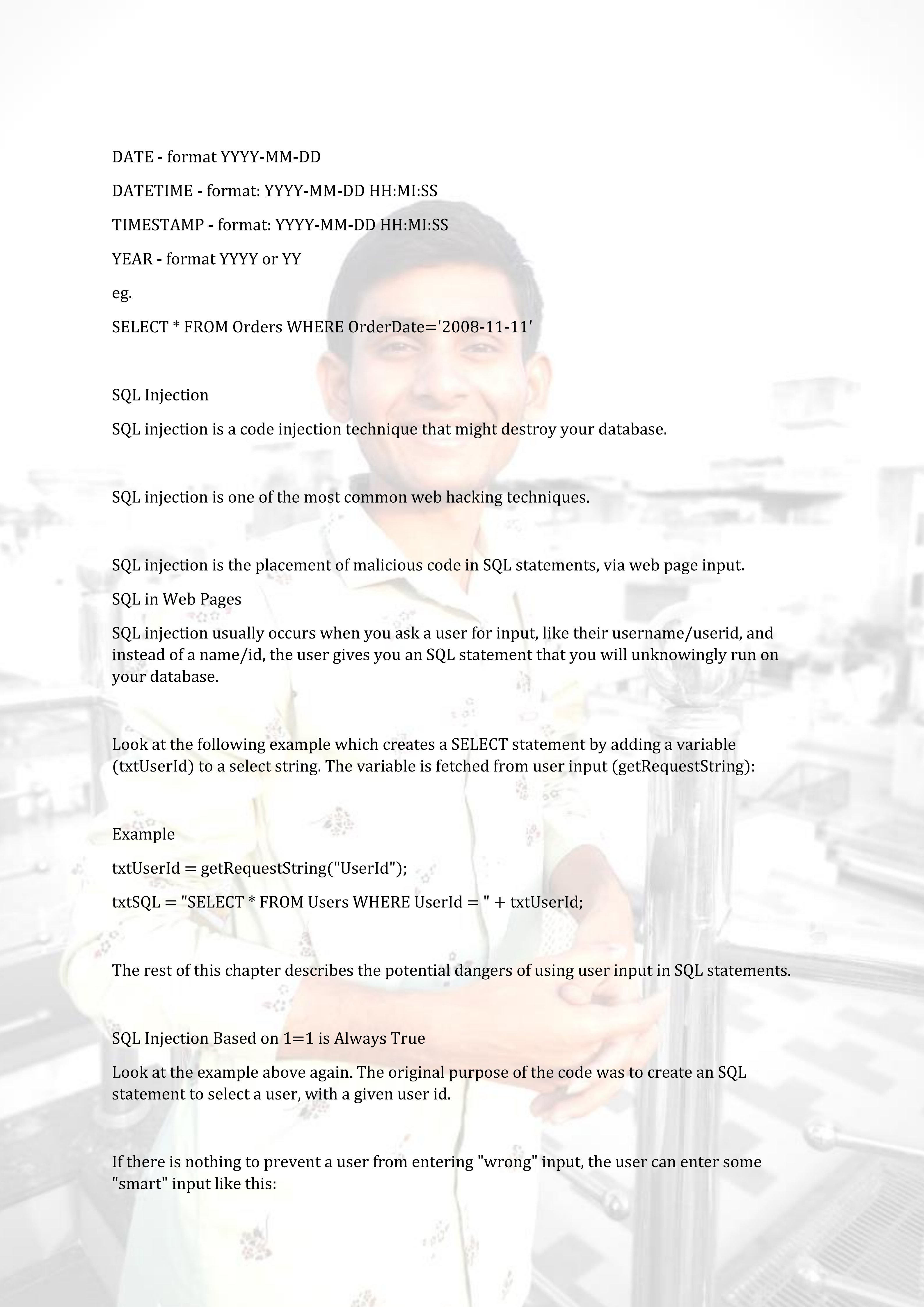
Creates a unique index on a table. Duplicate values are not allowed:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
eg. CREATE INDEX idx_lastname
ON Persons (LastName);
```

SQL Dates

SQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:



DATE - format YYYY-MM-DD

DATETIME - format: YYYY-MM-DD HH:MI:SS

TIMESTAMP - format: YYYY-MM-DD HH:MI:SS

YEAR - format YYYY or YY

eg.

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

SQL Injection

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL in Web Pages

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

Example

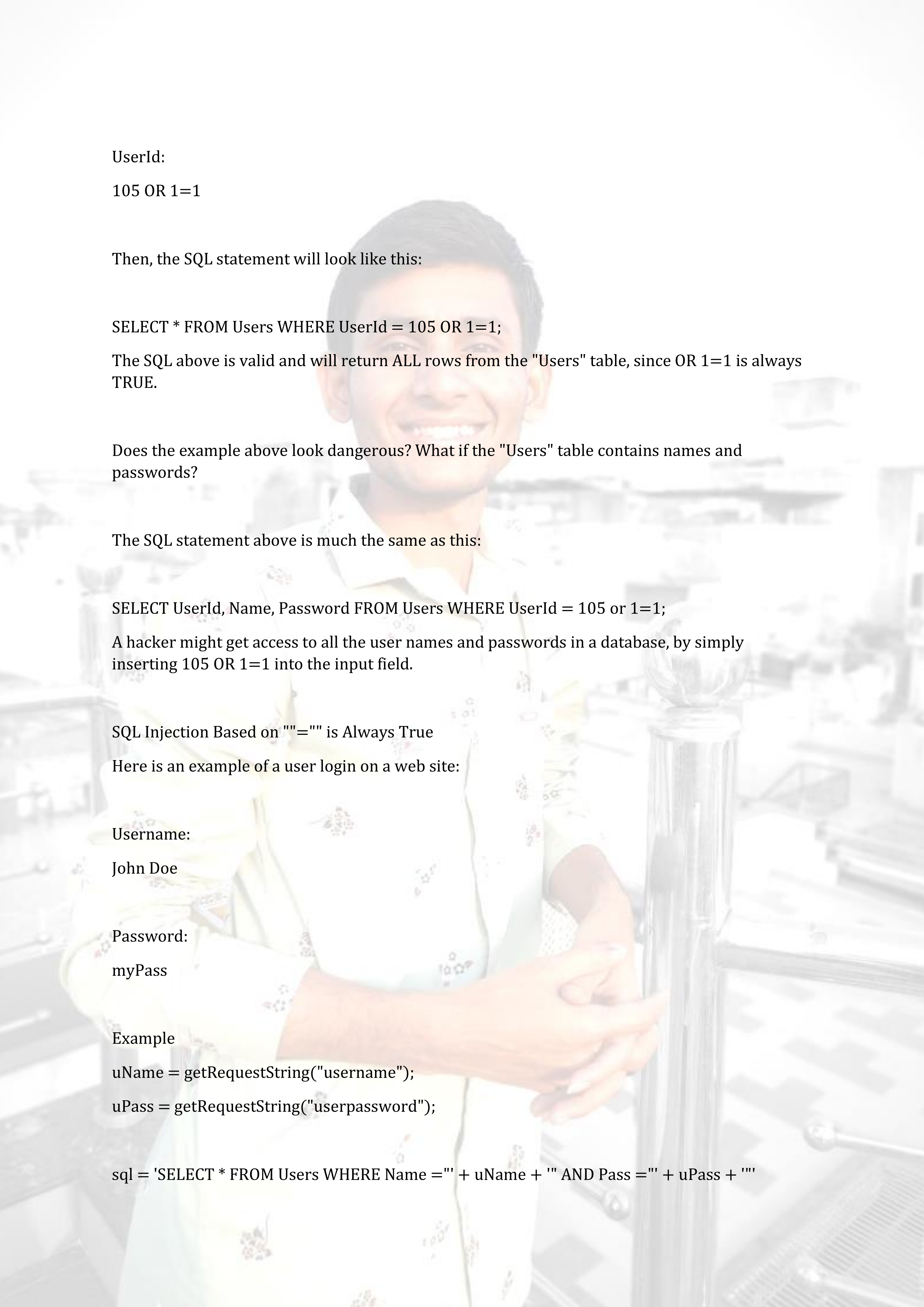
```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

The rest of this chapter describes the potential dangers of using user input in SQL statements.

SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:



UserId:

105 OR 1=1

Then, the SQL statement will look like this:

SELECT * FROM Users WHERE UserId = 105 OR 1=1;

The SQL above is valid and will return ALL rows from the "Users" table, since OR 1=1 is always TRUE.

Does the example above look dangerous? What if the "Users" table contains names and passwords?

The SQL statement above is much the same as this:

SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;

A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

SQL Injection Based on ""="" is Always True

Here is an example of a user login on a web site:

Username:

John Doe

Password:

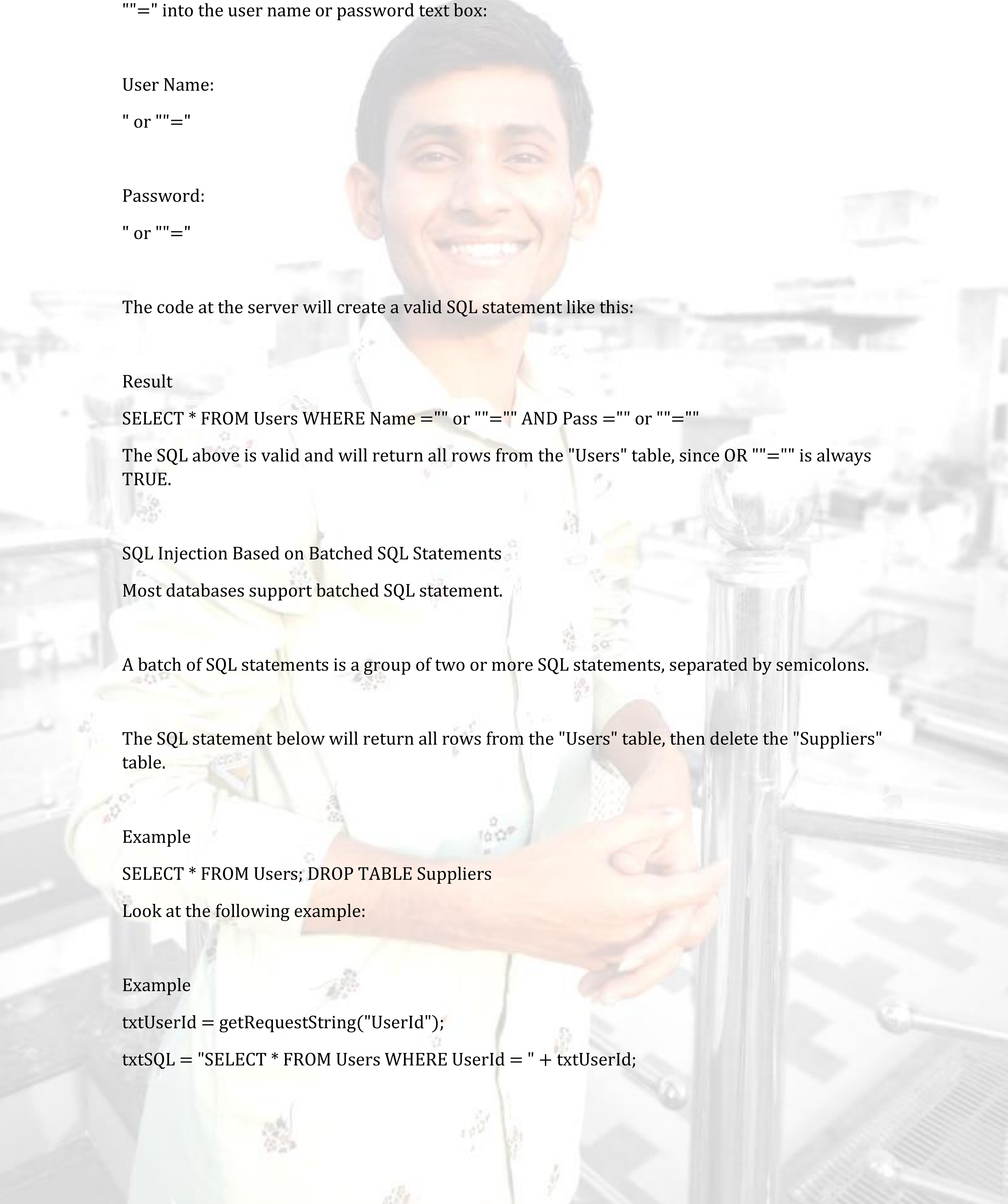
myPass

Example

uName = getRequestString("username");

uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'



Result

```
SELECT * FROM Users WHERE Name ="John Doe" AND Pass ="myPass"
```

A hacker might get access to user names and passwords in a database by simply inserting " OR ""=" into the user name or password text box:

User Name:

" or ""="

Password:

" or ""="

The code at the server will create a valid SQL statement like this:

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

The SQL above is valid and will return all rows from the "Users" table, since OR ""="" is always TRUE.

SQL Injection Based on Batched SQL Statements

Most databases support batched SQL statement.

A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.

The SQL statement below will return all rows from the "Users" table, then delete the "Suppliers" table.

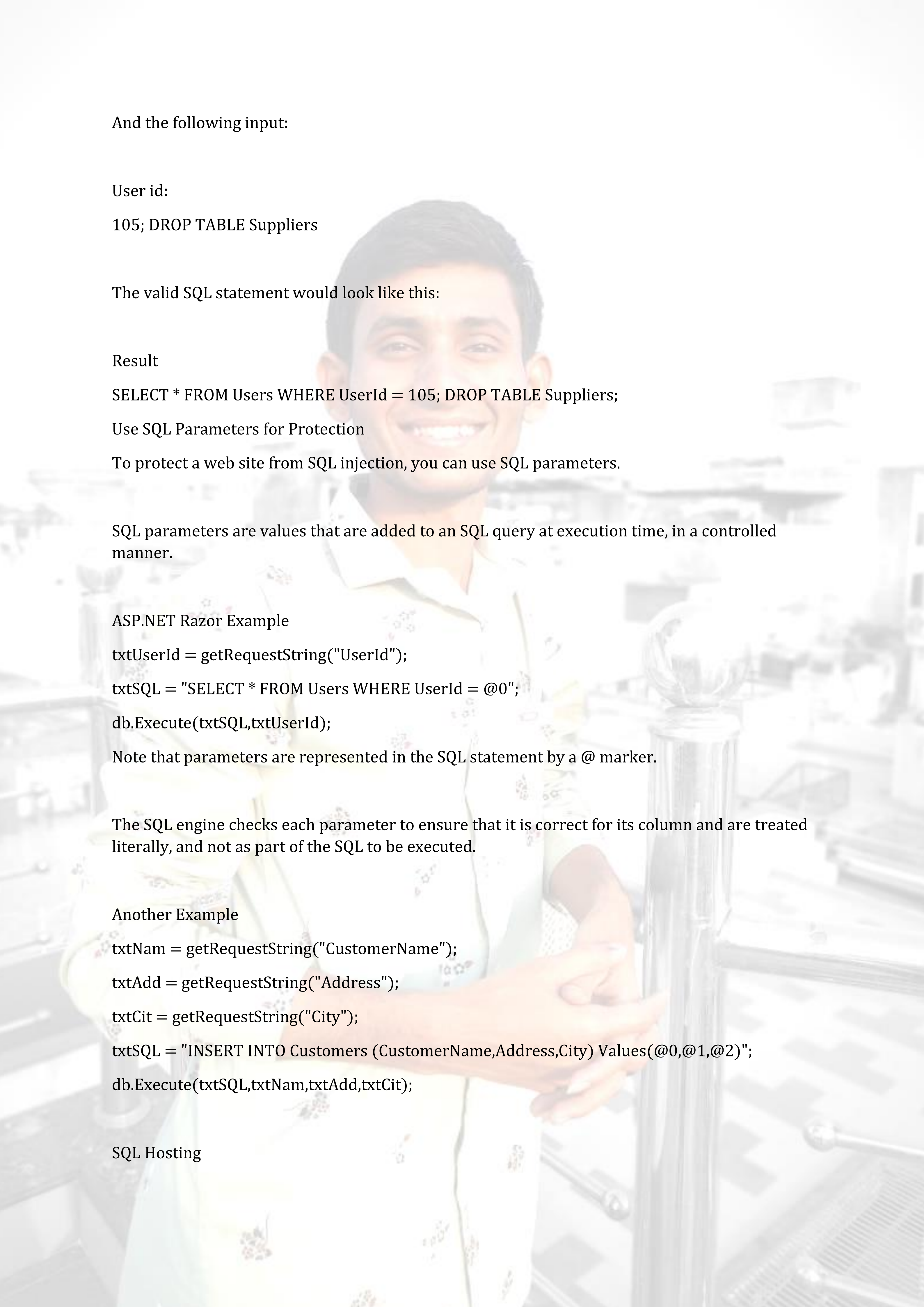
Example

```
SELECT * FROM Users; DROP TABLE Suppliers
```

Look at the following example:

Example

```
txtUserId = getRequestString("UserId");

txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```



And the following input:

User id:

```
105; DROP TABLE Suppliers
```

The valid SQL statement would look like this:

Result

```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;
```

Use SQL Parameters for Protection

To protect a web site from SQL injection, you can use SQL parameters.

SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

ASP.NET Razor Example

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = @0";
db.Execute(txtSQL,txtUserId);
```

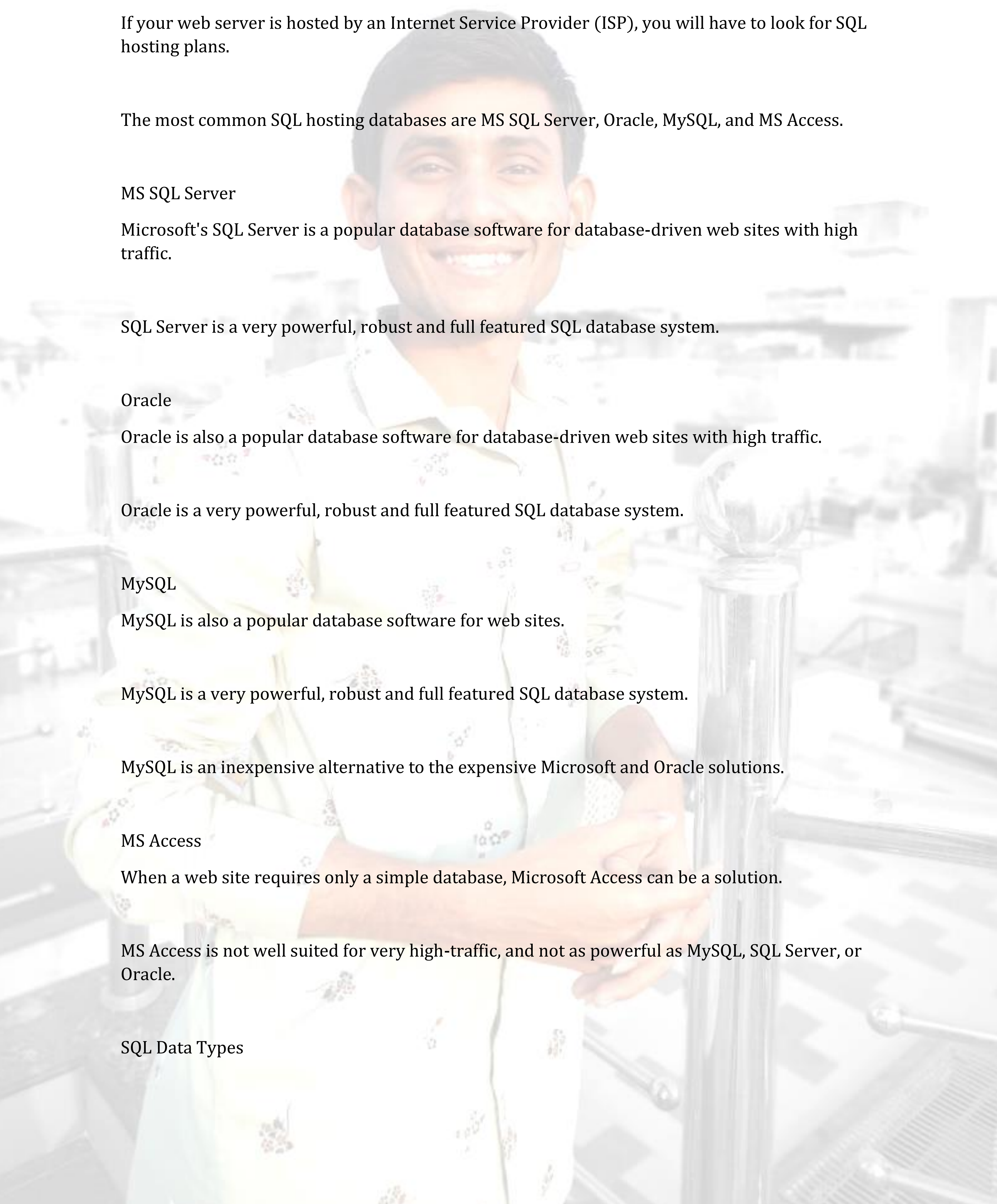
Note that parameters are represented in the SQL statement by a @ marker.

The SQL engine checks each parameter to ensure that it is correct for its column and are treated literally, and not as part of the SQL to be executed.

Another Example

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```

SQL Hosting



If you want your web site to be able to store and retrieve data from a database, your web server should have access to a database-system that uses the SQL language.

If your web server is hosted by an Internet Service Provider (ISP), you will have to look for SQL hosting plans.

The most common SQL hosting databases are MS SQL Server, Oracle, MySQL, and MS Access.

MS SQL Server

Microsoft's SQL Server is a popular database software for database-driven web sites with high traffic.

SQL Server is a very powerful, robust and full featured SQL database system.

Oracle

Oracle is also a popular database software for database-driven web sites with high traffic.

Oracle is a very powerful, robust and full featured SQL database system.

MySQL

MySQL is also a popular database software for web sites.

MySQL is a very powerful, robust and full featured SQL database system.

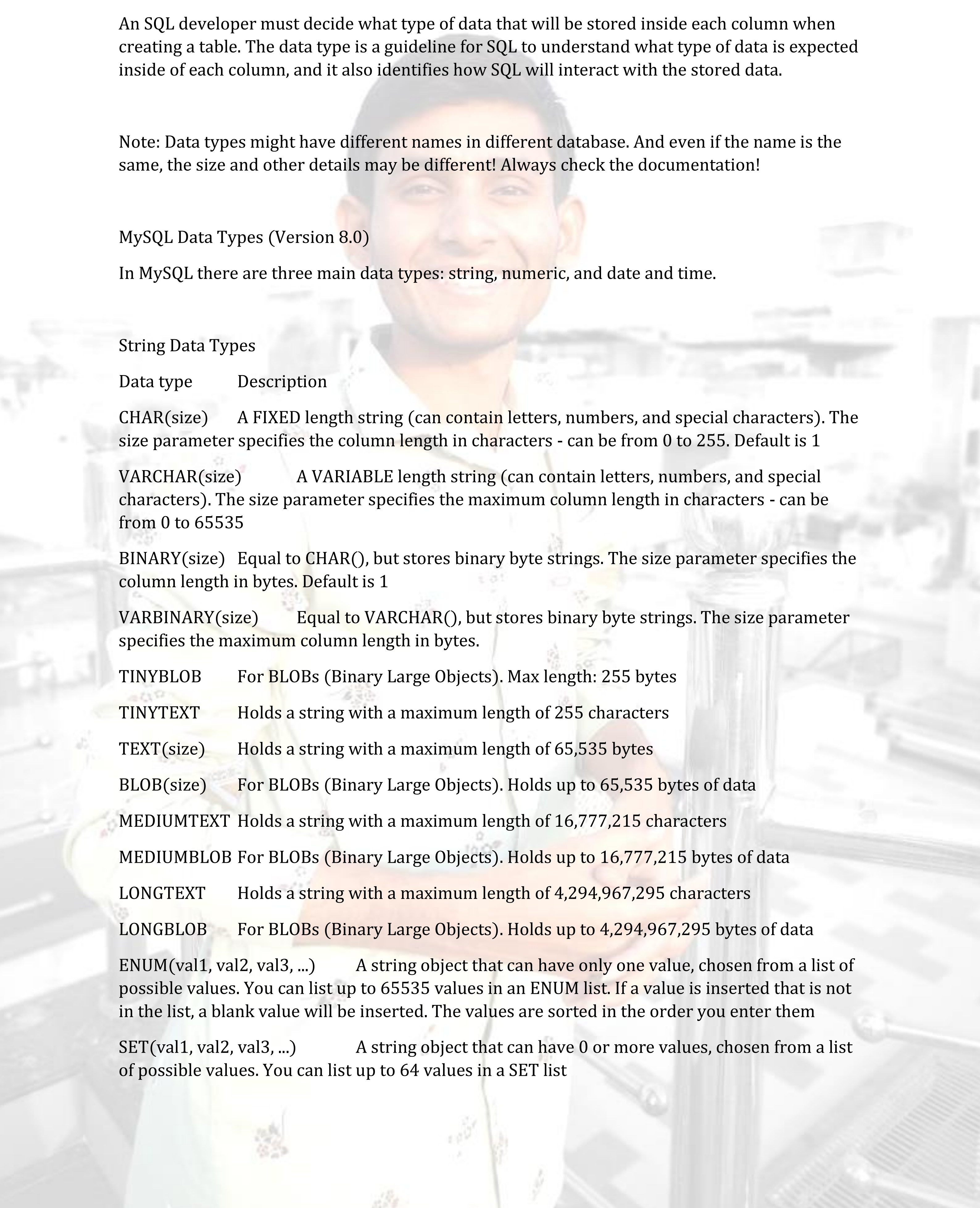
MySQL is an inexpensive alternative to the expensive Microsoft and Oracle solutions.

MS Access

When a web site requires only a simple database, Microsoft Access can be a solution.

MS Access is not well suited for very high-traffic, and not as powerful as MySQL, SQL Server, or Oracle.

SQL Data Types



Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

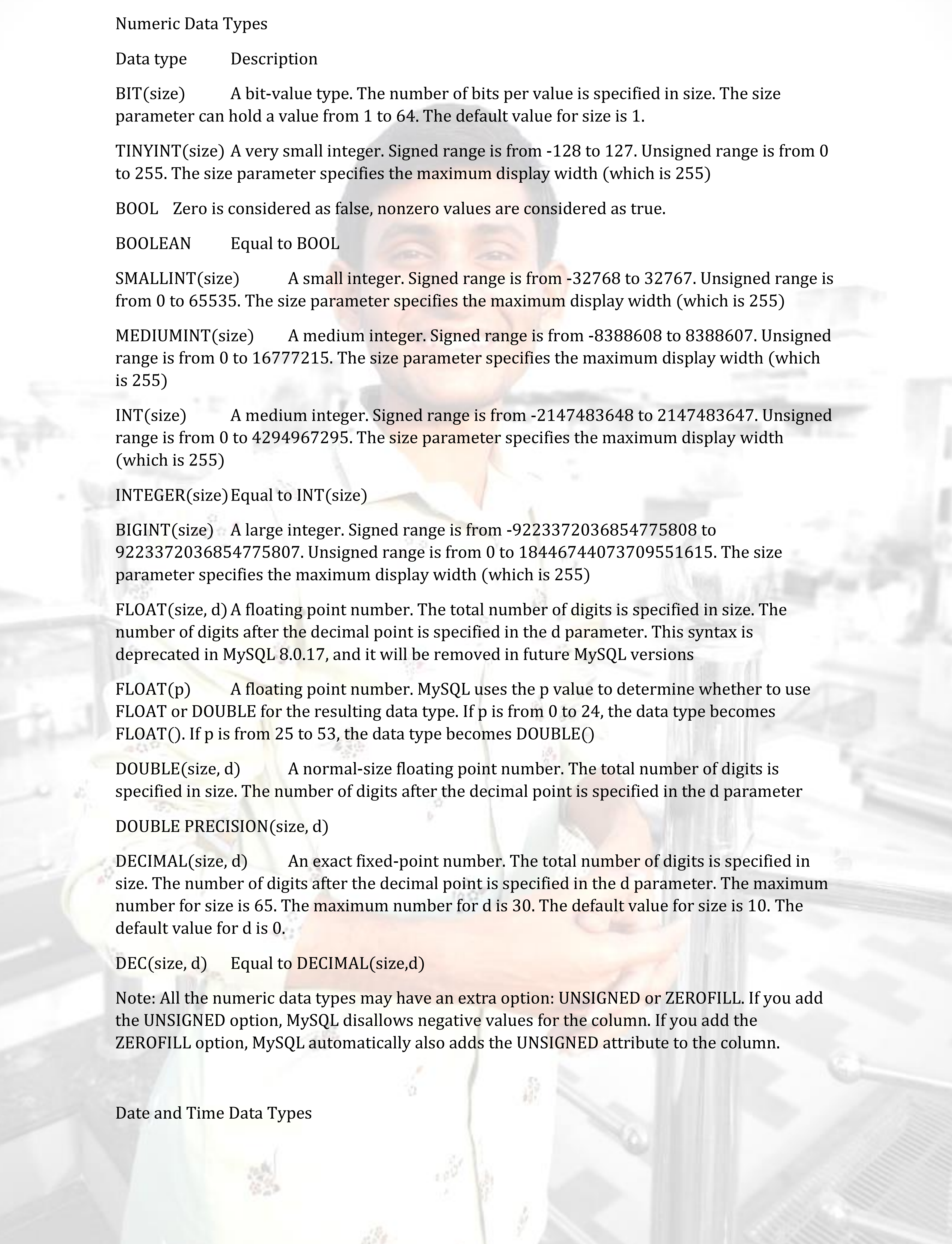
Note: Data types might have different names in different database. And even if the name is the same, the size and other details may be different! Always check the documentation!

MySQL Data Types (Version 8.0)

In MySQL there are three main data types: string, numeric, and date and time.

String Data Types

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

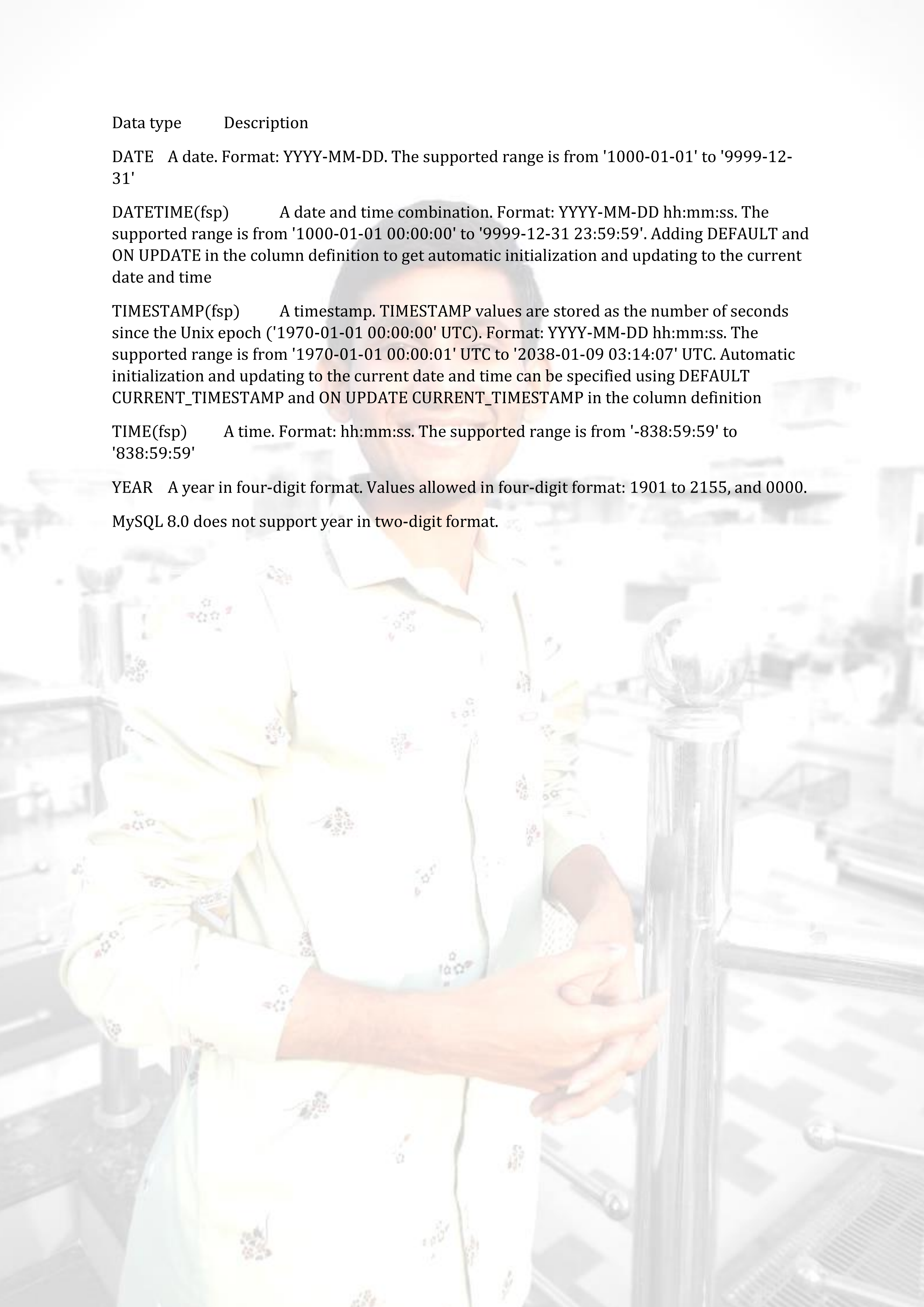


Numeric Data Types

Data type	Description
BIT(size)	A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.
TINYINT(size)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(size)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255)
MEDIUMINT(size)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255)
INT(size)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)
INTEGER(size)	Equal to INT(size)
BIGINT(size)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255)
FLOAT(size, d)	A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(p)	A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(size, d)	A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter
DOUBLE PRECISION(size, d)	
DECIMAL(size, d)	An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0.
DEC(size, d)	Equal to DECIMAL(size,d)

Note: All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

Date and Time Data Types



Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(fsp)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(fsp)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(fsp)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000.

MySQL 8.0 does not support year in two-digit format.