Day16: SQL

- SQL stands for Structured Query Language

- SQL lets you access and manipulate databases

- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

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

SQL is a Standard - BUT....

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as `SELECT` , `UPDATE` , `DELETE` , `INSERT` , `WHERE`) in a similar manner.

Note: Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

Using 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

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:

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.

A record, also called a row, is each individual entry that exists in a table. For example, there are 91 records in the above Customers table. A record is a horizontal entity in a table.

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

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

Syntax

CREATE DATABASE *databasename*;

Example

CREATE DATABASE testDB;

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*, ...);

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

The SQL INSERT INTO Statement

The `INSERT INTO` statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the `INSERT INTO` statement in two ways:

1. Specify both the column names and the values to be inserted:

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

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the `INSERT INTO` syntax would be as follows:

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

The SQL SELECT Statement

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

The data returned is stored in a result table, called the result-set.

SELECT Syntax

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

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

SELECT * FROM *table_name*;

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 columnsof all the recordsin the Customers table:*/ SELECT * FROM Customers;

The SQL SELECT DISTINCT Statement

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

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

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

SELECT DISTINCT Examples

The following SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table:

Example

`SELECT DISTINCT Country FROM Customers;`

The following SQL statement lists the number of different (distinct) customer countries:

Example

`SELECT COUNT(DISTINCT Country) FROM Customers;`

The SQL WHERE Clause

The `WHERE` clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax

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

SELECT * FROM Customers WHERE Country='Mexico';

SELECT * FROM Customers

WHERE Country='Mexico';

The SQL AND, OR and NOT Operators

The `WHERE` clause can be combined with `AND`, `OR`, and `NOT` operators.

The `AND` and `OR` operators are used to filter records based on more than one condition:

- The `AND` operator displays a record if all the conditions separated by `AND` are TRUE.
- The `OR` operator displays a record if any of the conditions separated by `OR` is TRUE.

The `NOT` operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

SELECT *column1*, *column2, ...*FROM *table_name* WHERE *condition1* AND *condition2* AND *condition3 ...*;

OR Syntax

SELECT *column1*, *column2, ...*FROM *table_name* WHERE *condition1* OR *condition2* OR *condition3 ...*;

NOT Syntax

SELECT *column1*, *column2, ...*FROM *table_name* WHERE NOT *condition*;

AND Example

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is "Berlin":

Example

SELECT * FROM Customers WHERE Country='Germany' AND City='Berlin';

OR Example

The following SQL statement selects all fields from "Customers" where city is "Berlin" OR "München":

Example

SELECT * FROM Customers WHERE City='Berlin' OR City='München';

The following SQL statement selects all fields from "Customers" where country is "Germany" OR "Spain":

Example

SELECT * FROM Customers WHERE Country='Germany' OR Country='Spain';

NOT Example

The following SQL statement selects all fields from "Customers" where country is NOT "Germany":

Example

SELECT * FROM Customers WHERE NOT Country='Germany';

Combining AND, OR and NOT

You can also combine the `AND` , `OR` and `NOT` operators.

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions):

Example

SELECT * FROM Customers WHERE Country='Germany' AND (City='Berlin' OR City='München');

The following SQL statement selects all fields from "Customers" where country is NOT "Germany" and NOT "USA":

Example

SELECT * FROM Customers WHERE NOT Country='Germany' AND NOT Country='USA';

The SQL ORDER BY Keyword

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

The `ORDER BY` keyword sorts the records in ascending order by default. To sort the records in descending order, use the `DESC` keyword.

ORDER BY Syntax

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

ORDER BY Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

Example

SELECT * FROM CustomersORDER BY Country;

ORDER BY DESC Example

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

Example

SELECT * FROM CustomersORDER BY Country DESC;

ORDER BY Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

Example

SELECT * FROM CustomersORDER BY Country, CustomerName;

ORDER BY Several Columns Example 2

The following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column:

Example

SELECT * FROM CustomersORDER BY Country ASC, CustomerName DESC;

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);*

SQL GROUP BY Examples

The following SQL statement lists the number of customers in each country:

Example

SELECT COUNT(CustomerID), CountryFROM CustomersGROUP BY Country;

Try it Yourself »

The following SQL statement lists the number of customers in each country, sorted high to low:

Example

SELECT COUNT(CustomerID), CountryFROM CustomersGROUP BY CountryORDER BY COUNT(CustomerID) DESC;

What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: 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!

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

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;

The IS NULL Operator

The `IS NULL` operator is used to test for empty values (NULL values).

The following SQL lists all customers with a NULL value in the "Address" field:

Example

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

Tip: Always use IS NULL to look for NULL values.

The IS NOT NULL Operator

The `IS NOT NULL` operator is used to test for non-empty values (NOT NULL values).

The following SQL lists all customers with a value in the "Address" field:

Example

`SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NOT 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*;

Note: Be careful when updating records in a table! Notice the `WHERE` clause in the `UPDATE` statement. The `WHERE` clause specifies which record(s) that should be updated. If you omit the `WHERE` clause, all records in the table will be updated!

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

Example

UPDATE CustomersSET ContactName = 'Alfred Schmidt', City= 'Frankfurt'WHERE CustomerID = 1;

UPDATE Multiple Records

It is the `WHERE` clause that determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where country is "Mexico":

Example

UPDATE CustomersSET ContactName='Juan'WHERE Country='Mexico';

Update Warning!

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

The SQL DELETE Statement

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

DELETE Syntax

DELETE FROM *table_name* WHERE *condition*;

MySQL Syntax:

```
SELECT  column_name(s)  FROM  table_name  WHERE  condition  LIMIT  number ;
SELECT * FROM Customers LIMIT 3;
```


The SQL MIN() and MAX() Functions

The `MIN()` function returns the smallest value of the selected column.

The `MAX()` function returns the largest value of the selected column.

MIN() Syntax

SELECT MIN(*column_name*)FROM *table_name*WHERE *condition*;

MAX() Syntax

SELECT MAX(*column_name*)FROM *table_name*WHERE *condition*;

The following SQL statement finds the price of the cheapest product:

Example

SELECT MIN(Price) AS SmallestPrice FROM Products;

MAX() Example

The following SQL statement finds the price of the most expensive product:

Example

SELECT MAX(Price) AS LargestPrice FROM Products;

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

The `COUNT()` function returns the number of rows that matches a specified criterion.

COUNT() 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*;

COUNT() Example

The following SQL statement finds the number of products:

Example

SELECT COUNT(ProductID) FROM Products;

Note: NULL values are not counted.

AVG() Example

The following SQL statement finds the average price of all products:

Example

SELECT AVG(Price)FROM Products;

Note: NULL values are ignored.

SUM() Example

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

Example

SELECT SUM(Quantity)FROM OrderDetails;

Note: NULL values are ignored.

The SQL LIKE Operator

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

There are two wildcards often used in conjunction with the `LIKE` operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

The percent sign and the underscore can also be used in combinations!

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

Aa 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"

IN Operator Examples

The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

Example

SELECT * FROM CustomersWHERE Country IN ('Germany', 'France', 'UK');

BETWEEN Example

The following SQL statement selects all products with a price between 10 and 20:

Example

SELECT * FROM ProductsWHERE Price BETWEEN 10 AND 20;

DROP COLUMN Example

Next, we want to delete the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

ALTER TABLE PersonsDROP COLUMN DateOfBirth;

SQL DROP TABLE Example

The following SQL statement drops the existing table "Shippers":

Example

DROP TABLE Shippers;

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.

Look at the following two tables:

MySQL supports foreign keys, which permit cross-referencing related data across tables, and foreign key constraints, which help keep the related data consistent.

A foreign key relationship involves a parent table that holds the initial column values, and a child table with column values that reference the parent column values. A foreign key constraint is defined on the child table.

This following example relates `parent` and `child` tables through a single-column foreign key and shows how a foreign key constraint enforces referential integrity.

Create the parent and child tables:

```
CREATE TABLE parent (
    id INT NOT NULL,
    PRIMARY KEY (id)
) ;
CREATE TABLE child (
    id INT,
    parent_id INT,
    INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id)
        REFERENCES parent(id)
) ;
```

Insert a row into the parent table:

```
mysql> INSERT INTO parent (id) VALUES (1);
```

Verify that the data was inserted:

```
mysql> SELECT * FROM parent;
+----+| id |+----+|  1 |+----+
```

Insert a row into the child table:

```
mysql> INSERT INTO child (id,parent_id) VALUES (1,1);
```

The insert operation is successful because `parent_id` 1 is present in the parent table.

Insert a row into the child table with a `parent_id` value that is not present in the parent table:

```
mysql> INSERT INTO child (id,parent_id) VALUES(2,2);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
(`test`.`child`, CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
REFERENCES `parent` (`id`))
```

The operation fails because the specified `parent_id` value does not exist in the parent table.

Try to delete the previously inserted row from the parent table:

```
mysql> DELETE FROM parent WHERE id  = 1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
(`test`.`child`, CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
REFERENCES `parent` (`id`))
```

This operation fails because the record in the child table contains the referenced id (`parent_id`) value.

When an operation affects a key value in the parent table that has matching rows in the child table, the result depends on the referential action specified by `ON UPDATE` and `ON DELETE` subclauses of the `FOREIGN KEY` clause. Omitting `ON DELETE` and `ON UPDATE` clauses (as in the current child table definition) is the same as specifying the `RESTRICT` option, which rejects operations that affect a key value in the parent table that has matching rows in the parent table.

DROP a FOREIGN KEY Constraint

To drop a `FOREIGN KEY` constraint, use the following SQL:

MySQL:

ALTER TABLE Orders DROP FOREIGN KEY FK_PersonOrder;

MySQL Data Types (Version 8.0)

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

String Data Types

Aa 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

Aa Data type	☰ Description
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

Aa Title	☰ Data type	☰ Description
Untitled	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.
Untitled	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)
Untitled	BOOL	Zero is considered as false, nonzero values are considered as true.
Untitled	BOOLEAN	Equal to BOOL
Untitled	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)
Untitled	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)
Untitled	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)
Untitled	INTEGER(*size*)	Equal to INT(size)
Untitled	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)
Untitled	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
Untitled	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()
Untitled	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
Untitled	DOUBLE PRECISION(*size*, *d*)	
Untitled	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.
Untitled	DEC(*size*, *d*)	Equal to DECIMAL(size,d)

Date and Time Data Types

☰ Data type	Aa Description

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.

This SQL keywords reference contains the reserved words in SQL.

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

☰ Keyword	Aa Description
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

≡ Keyword	Aa Description
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