

SQL: DDL and DML

Anthony Kleerekoper

Database Systems 6G4Z0016



vevox
Audience Engagement

Join at vevox.app

Or search **Vevox** in the app store

ID: 123-732-226



What is SQL?

SQL is the language used to interact with relational databases

“Structured Query Language”

Pronounced “ess-cue-ell” (though lots of people say “sequel”)



Hawker Hurricane single-seat fighter aircraft
responsible for more than half of Luftwaffe losses during
Battle of Britain

By Photo: Cpl Phil Major ABIPP/MOD, OGL v1.0,
<https://commons.wikimedia.org/w/index.php?curid=110478807>

Standard and Implementations

There is an SQL standard and there are vendor implementations

e.g. MariaDB, Oracle, MySQL, PostgreSQL, Microsoft SQL Server

Often vendors have implemented parts before they are in the standard

In this unit we will teach to the standard as much as possible

where not possible – we will teach MariaDB's implementation

sometimes point out where others (e.g. Oracle) are different

SQL is Declarative

Java is an imperative language

We tell the computer what actions to take

SQL is a declarative language

We tell the computer what result we want

SQL statements are somewhat independent of each other

could think of them as messages sent to the server and acted on

once executed any changes are made permanently

SQL has three parts

DDL – Data Definition Language

Deals with changes to structure

CREATE, ALTER, DROP

DML – Data Manipulation Language

Deals with changes to data

INSERT, SELECT, UPDATE, DELETE

DCL – Data Control Language

changes to permission, access etc.

Data Definition Language

Creating tables and constraints

Creating New Tables

CREATE TABLE statement creates the table structure

General syntax:

```
CREATE TABLE tbl_name (  
    col1 data_type,  
    col2 data_type,  
    col3 data_type  
);
```


Database Constraints

A big advantage of databases is the ability to add **constraints**

Constraints are rules that limit what data can be put in a table

Examples:

- datatypes – limit what kind of data can be put in a column

- primary and foreign keys – limit what values can be added

MariaDB Datatypes

Type of Data	Syntax	Notes
String	CHAR [(size)]	Size is optional - default is 1 character Maximum size is 255 characters
String	VARCHAR (size)	Size is required Maximum size is 65,532 characters
String	LONGTEXT	Up to 4,294,967,295 characters or 4GB
Datetime	DATE	Uses “YYYY-MM-DD” format e.g. “2022-01-28” for 28 th January 2022
Datetime	TIME	Uses “HH:MM:SS” format e.g. “12:05:16” for 12 hours, 5 minutes and 16 seconds
Datetime	DATETIME	Date and Time combined, “YYYY-MM-DD HH:MM:SS”
Numeric	INT	Integers between -2,147,483,648 and 2,147,483,647
Numeric	DECIMAL [(M [, D])]	M is the total number of digits. D is the number of digits after the decimal point.

CHAR vs VARCHAR

Property	CHAR	VARCHAR
Syntax	CHAR [(size)]	VARCHAR (size)
Size parameter	Optional – default is 1	Required
Length	Fixed Length	Variable Length
Padding	Pads to the right	End-of-string delimiter
Maximum length	255 characters	65,535 characters
Best for	Fixed or low-variation fields	Very variable fields

Precision and Scale

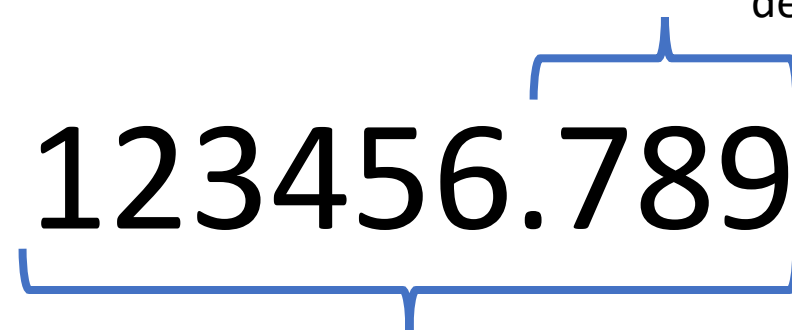
DECIMAL [(M[, D])]

Decimal has two parameters:

M is **precision** = total number of digits (default is 10)

D is **scale** = number of digits after the decimal point (default is 0)

Scale = 3 (number of digits after decimal point)



123456.789

The diagram shows the number 123456.789. A blue bracket underneath the entire number (123456.789) indicates the precision is 9. Another blue bracket above the digits 789 indicates the scale is 3.

Precision = 9 (total number of digits)

vevox.app ID: 123-732-226

Precision and Scale Examples

If no precision or scale is specified, the decimal places so rounds off default to DECIMAL(10,0)

Input Data	Specified As	Stored As
7,456,123.89	DECIMAL	7456124
7,456,123.89	DECIMAL(10,0)	7456124
7,456,123.89	DECIMAL(9,2)	7456123.89
7,456,123.89	DECIMAL(9,1)	7456123.9
7,456,123.89	DECIMAL(6)	(not accepted, exceeds precision)

Can store this number exactly

Rounds off to 1 decimal place

Can only store 6 digits and input data has 7

Integer Types

MariaDB has a number of integer types, all have an optional “length”

e.g. `INT (4)` , `BIGINT (10)`

The length does nothing unless we specify `ZEROFILL` and then it pads

The range is fixed by the type, to properly limit use `DECIMAL`

Datatype	Number of Bytes	Minimum Value	Maximum Value
TINYINT	1	-128	127
SMALLINT	2	-32,768	32,767
MEDIUMINT	3	-8,388,608	8,388,607
INT or INTEGER	4	-2,147,483,648	2,147,483,647
BIGINT	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

Example Database

Universities

uni_name	city	enrolment	app_deadline
Man Met	Manchester	25,810	2022-09-15
Uni of Manchester	Manchester	26,725	2022-09-20
Salford Uni	Salford	14,895	2022-09-18
John Moores	Liverpool	17,835	2022-09-22

Students

snum	stu_name	dob	points	size_hs
003	Jack Fines	2001-09-12	110	60
009	Michelle Jones	2000-12-22	114	50
017	Nazia Hassan	2001-05-05	101	50
022	Shane Jordan	2002-10-10	121	35
035	Peter Watson	2001-06-29	117	45

Applications

snum	uni_name	course	decision
003	Man Met	Computing	Accept
003	Man Met	Computer Science	Accept
009	Uni of Manchester	Computer Science	Reject
017	Man Met	Computing	Reject
017	Salford Uni	Computing	Accept
022	Man Met	Computing	Accept

Example: Creating the Universities Table

```
CREATE TABLE Universities (  
    uni_name          VARCHAR(50),  
    city              VARCHAR(50),  
    enrolment         INT,  
    app_deadline      DATE  
);
```

Universities (uni_name, city, enrolment, app_deadline)

uni_name	city	enrolment	app_deadline
Man Met	Manchester	25,810	2022-09-15
Uni of Manchester	Manchester	26,725	2022-09-20
Salford Uni	Salford	14,895	2022-09-18
John Moores	Liverpool	17,835	2022-09-22

Students (snum, stu_name, dob, points, size_hs)

snum	stu_name	dob	points	size_hs
003	Jack Fines	2001-09-12	110	60
009	Michelle Jones	2000-12-22	114	50
017	Nazia Hassan	2001-05-05	101	50
022	Shane Jordan	2002-10-10	121	35
035	Peter Watson	2001-06-29	117	45

Applications (snum, uni_name, course, decision)

snum	uni_name	course	decision
003	Man Met	Computing	Accept
003	Man Met	Computer Science	Accept
009	Uni of Manchester	Computer Science	Reject
017	Man Met	Computing	Reject
017	Salford Uni	Computing	Accept
022	Man Met	Computing	Accept

Primary Keys

A primary key uniquely identifies a row in a table
can be one column or multiple columns

Examples:

snum in Students table

(snum, uni_name, course) in Applications table

Universities (uni_name, city, enrolment, app_deadline)

uni_name	city	enrolment	app_deadline
Man Met	Manchester	25,810	2022-09-15
Uni of Manchester	Manchester	26,725	2022-09-20
Salford Uni	Salford	14,895	2022-09-18
John Moores	Liverpool	17,835	2022-09-22

Students (snum, stu_name, dob, points, size_hs)

snum	stu_name	dob	points	size_hs
003	Jack Fines	2001-09-12	110	60
009	Michelle Jones	2000-12-22	114	50
017	Nazia Hassan	2001-05-05	101	50
022	Shane Jordan	2002-10-10	121	35
035	Peter Watson	2001-06-29	117	45

Applications (snum, uni_name, course, decision)

snum	uni_name	course	decision
003	Man Met	Computing	Accept
003	Man Met	Computer Science	Accept
009	Uni of Manchester	Computer Science	Reject
017	Man Met	Computing	Reject
017	Salford Uni	Computing	Accept
022	Man Met	Computing	Accept

Primary Keys

Primary key is a fundamental constraint

Limits what data can be inserted into the table

Example: (snum, uni_name, course) is Primary Key of Applications table
each student can only apply once to each course at each university

Universities (uni_name, city, enrolment, app_deadline)

uni_name	city	enrolment	app_deadline
Man Met	Manchester	25,810	2022-09-15
Uni of Manchester	Manchester	26,725	2022-09-20
Salford Uni	Salford	14,895	2022-09-18
John Moores	Liverpool	17,835	2022-09-22

Students (snum, stu_name, dob, points, size_hs)

snum	stu_name	dob	points	size_hs
003	Jack Fines	2001-09-12	110	60
009	Michelle Jones	2000-12-22	114	50
017	Nazia Hassan	2001-05-05	101	50
022	Shane Jordan	2002-10-10	121	35
035	Peter Watson	2001-06-29	117	45

Applications (snum, uni_name, course, decision)

snum	uni_name	course	decision
003	Man Met	Computing	Accept
003	Man Met	Computer Science	Accept
009	Uni of Manchester	Computer Science	Reject
017	Man Met	Computing	Reject
017	Salford Uni	Computing	Accept
022	Man Met	Computing	Accept

Adding Primary Key Constraint

You can add key constraints **inline** as follows:

```
CREATE TABLE Universities (  
    uni_name          VARCHAR(50) PRIMARY KEY,  
    city              VARCHAR(50) ,  
    enrolment         INT,  
    app_deadline      DATE  
);
```

Universities (uni_name, city, enrolment, app_deadline)

uni_name	city	enrolment	app_deadline
Man Met	Manchester	25,810	2022-09-15
Uni of Manchester	Manchester	26,725	2022-09-20
Salford Uni	Salford	14,895	2022-09-18
John Moores	Liverpool	17,835	2022-09-22

Students (snum, stu_name, dob, points, size_hs)

snum	stu_name	dob	points	size_hs
003	Jack Fines	2001-09-12	110	60
009	Michelle Jones	2000-12-22	114	50
017	Nazia Hassan	2001-05-05	101	50
022	Shane Jordan	2002-10-10	121	35
035	Peter Watson	2001-06-29	117	45

Applications (snum, uni_name, course, decision)

snum	uni_name	course	decision
003	Man Met	Computing	Accept
003	Man Met	Computer Science	Accept
009	Uni of Manchester	Computer Science	Reject
017	Man Met	Computing	Reject
017	Salford Uni	Computing	Accept
022	Man Met	Computing	Accept

Composite Key Constraints

Inline declaration impossible for composite keys
add constraint after declaring the columns

```
CREATE TABLE Applications(  
    snum          INT,  
    uni_name      VARCHAR(50),  
    course        VARCHAR(20),  
    decision      CHAR(6),  
  
    CONSTRAINT PRIMARY KEY (snum, uni_name, course)  
);
```

Universities (uni_name, city, enrolment, app_deadline)

uni_name	city	enrolment	app_deadline
Man Met	Manchester	25,810	2022-09-15
Uni of Manchester	Manchester	26,725	2022-09-20
Salford Uni	Salford	14,895	2022-09-18
John Moores	Liverpool	17,835	2022-09-22

Students (snum, stu_name, dob, points, size_hs)

snum	stu_name	dob	points	size_hs
003	Jack Fines	2001-09-12	110	60
009	Michelle Jones	2000-12-22	114	50
017	Nazia Hassan	2001-05-05	101	50
022	Shane Jordan	2002-10-10	121	35
035	Peter Watson	2001-06-29	117	45

vevox.app ID: 123-732-226

Applications (snum, uni_name, course, decision)

snum	uni_name	course	decision
003	Man Met	Computing	Accept
003	Man Met	Computer Science	Accept
009	Uni of Manchester	Computer Science	Reject
017	Man Met	Computing	Reject
017	Salford Uni	Computing	Accept
022	Man Met	Computing	Accept

Types of Primary Keys

A **natural key** is a key with meaning to the user

exists outside of the database

e.g. National Insurance Numbers, UCAS numbers, number plates etc

Also called a *business key*

A **surrogate key** is a key with no meaning to the user

does not exist outside of the database (not a real key)

e.g. internally assigned ID numbers, auto-increment numbers

Also called a *technical key*

Advantages of Natural Keys

A well-designed table should have a natural key anyway

Every row should contain information about a different entity

Guarantee non-duplicated data

Cannot accidentally enter same data twice with different ID number

May improve performance

No need to find next available key

No need for extra ID column

Advantages of Surrogate Keys

No need to change the keys

A natural key may change, e.g. an employee provided incorrect information

No need to wait before creating a new record

e.g. if employee NI number is not yet known

May improve performance

Smaller keys, often numeric

Always available

Sometimes natural keys are not obvious

What to do in practice?

BIG DEBATE

Guidance:

Use a natural key if one is available that is small and stable

Use a surrogate key in other cases

My advice:

discuss with your colleagues

Auto-Generated Surrogate Keys

If you do choose a surrogate key for a table, can auto-generate

```
CREATE TABLE Students (  
    snum INT PRIMARY KEY AUTO_INCREMENT,  
    stu_name VARCHAR(50),  
    dob DATE,  
    points INT,  
    size_hs INT  
);
```

Universities (uni_name, city, enrolment, app_deadline)

uni_name	city	enrolment	app_deadline
Man Met	Manchester	25,810	2022-09-15
Uni of Manchester	Manchester	26,725	2022-09-20
Salford Uni	Salford	14,895	2022-09-18
John Moores	Liverpool	17,835	2022-09-22

Students (snum, stu_name, dob, points, size_hs)

snum	stu_name	dob	points	size_hs
003	Jack Fines	2001-09-12	110	60
009	Michelle Jones	2000-12-22	114	50
017	Nazia Hassan	2001-05-05	101	50
022	Shane Jordan	2002-10-10	121	35
035	Peter Watson	2001-06-29	117	45

Applications (snum, uni_name, course, decision)

snum	uni_name	course	decision
003	Man Met	Computing	Accept
003	Man Met	Computer Science	Accept
009	Uni of Manchester	Computer Science	Reject
017	Man Met	Computing	Reject
017	Salford Uni	Computing	Accept
022	Man Met	Computing	Accept

Foreign Keys

A Foreign Key is a column in table A whose values **uniquely** identify rows in table B

Table A is called the parent table, table B is the child table

Helps link tables together – but is primarily for data integrity

Example: snum in Applications table

refers to the snum column in the Students table

prevents us having an application for a student who doesn't exist

Foreign Key Constraint

Making a foreign key is a constraint

A column declared as a foreign key:

1. Must have the same data type as the column it links to
2. Can only contain values found in column it links to
Although can contain NULL values

Declaring a Foreign Key

Stand-alone:

```
CONSTRAINT name FOREIGN KEY (col_this_table)  
REFERENCES other_table (other_col)
```

Example:

```
CONSTRAINT fk_nm FOREIGN KEY (snum)  
REFERENCES Students (snum)
```

Declaring a Foreign Key

Inline:

```
col type REFERENCES other_table (other_col)
```

Example:

```
snum NUMBER(3) REFERENCES Students (snum)
```

Most versions of MariaDB ignore a FK declared this way!

Referential Integrity

Foreign Keys uniquely identify rows in another table

Therefore must contain values that exist in the other table (or NULL)

A foreign key value must exist in parent table
called ***referential integrity***

What if we want to change value in parent table?
e.g. need to change snum of a student?

Cascading Changes

Tell the database system to automatically change the foreign key value

Done when specifying the foreign key initially:

```
REFERENCES tbl(col) ON UPDATE CASCADE
```

If we change an snum value in Students table, corresponding snum in Applications table will change automatically

Other Inline Constraints

`NOT NULL`

Column cannot contain NULL values

`UNIQUE`

All values in column must be unique, but not primary key

`DEFAULT val`

If no value specified in INSERT, use this value

Can combine them – spaces but no commas between them

Example: Constraints

`stu_name VARCHAR(50) NOT NULL,`

`enrolment INT UNIQUE,`

`points INT DEFAULT 100,`

`email VARCHAR(50) NOT NULL UNIQUE,`

The CHECK constraint

Limit what values are allowed in a column

Use any condition expression – if value returns TRUE then allowed in

Very powerful constraint

Can be as complex as we want

CHECK Constraint Syntax

Inline:

```
points INT CHECK (points > 0)
```

Longer form:

```
CONSTRAINT chk_name CHECK (points > 0)
```

Anything that can go in a WHERE clause can go in CHECK constraint

We'll cover WHERE clauses next week

Changing CHECK Constraints

Only way to change a CHECK constraint is to delete and recreate

Cannot create constraint on table if existing data breaks constraint

Need to test every existing row

- Can be slow on large tables

- Prevents anyone using the table during testing

Advice on CHECK Constraints

Only use if the constraint is inherent and very unlikely to change

basic sanity check

e.g. points must be positive

Good to not rely on external code to perform inherent checks

Deleting Tables

Removing entire tables is done with the DROP command

```
DROP TABLE Students;
```

This removes the table and of course all the data in it

Referential Integrity prevents dropping the parent before the child

Because then the child has values not found in the parent

Some versions of SQL have a CASCADE for this

```
DROP TABLE Students CASCADE [CONSTRAINTS];
```

but MariaDB does not support this

Deleting Tables in Scripts

Scripts are collections of SQL statements in one file

Very common for creating and populating tables

Usual to start them with DROP TABLE statements to start fresh

If table doesn't yet exist then will throw an error

`DROP TABLE IF EXISTS tbl_name` to get warning only

Changes to Table Structures

Can add or remove a column:

```
ALTER TABLE Students ADD age INT;
```

```
ALTER TABLE Students DROP COLUMN size_hs;
```

Can change datatype:

```
ALTER TABLE Universities  
MODIFY city VARCHAR(25);
```

Only works if all values in the column are valid with new datatype

Renaming Columns and Tables

Can rename the table:

```
ALTER TABLE Students  
RENAME TO Applicants;
```

Can rename a column: (with this syntax need to respecify the column)

```
ALTER TABLE Students  
CHANGE COLUMN dob date_of_birth DATE;
```

Only in newest versions of MariaDB:

```
ALTER TABLE Students  
RENAME COLUMN dob TO date_of_birth;
```

Data Manipulation Language

Inserting New Rows

Data is inserted into a table with the INSERT statement

General Syntax:

```
INSERT INTO table_name (col1, col2, col3, ...)  
VALUES (val1, val2, val3, ...)
```

Example: Inserting New Rows

```
INSERT INTO Universities  
    (uni_name, city, enrolment, app_deadline)  
VALUES ('UCLAN', 'Preston', 20180, '2022-09-10');
```

uni_name	city	enrolment	app_deadline
Man Met	Manchester	25,810	2022-09-15
Uni of Manchester	Manchester	26,725	2022-09-20
Salford Uni	Salford	14,895	2022-09-18
John Moores	Liverpool	17,835	2022-09-22
UCLAN	Preston	20,180	2022-09-10

Note: strings and dates need to be inside quotation marks

Implicit Columns

Can skip listing columns if inserting into every column

Order of values must match table order

```
INSERT INTO Universities  
VALUES ( 'UCLAN', 'Preston', 20180, '2022-09-10' );
```

Best practice is to list columns explicitly

Don't leave things up to the database system

Inserting Multiple Rows

Can inserting many rows in one go:

```
INSERT INTO Students
      (snum, stu_name, dob, points, size_hs)
VALUES (003, 'Jack Fines',      '2001-09-12', 110, 60),
      (009, 'Michelle Jones',  '2000-12-22', 114, 50),
      (017, 'Nazia Hassan',    '2001-05-05', 101, 50),
      (022, 'Shane Jordan',    '2002-10-10', 121, 35),
      (035, 'Peter Watson',    '2001-06-29', 117, 45);
```

(Oracle does not support this)

Updating Data

The UPDATE statement modifies one or more rows

```
UPDATE Universities  
SET enrolment = 20180  
WHERE uni_name = 'UCLAN';
```

If you skip the WHERE clause – will update every row

Our server uses Safe Mode which prevents UPDATES without WHERE
And requires WHERE to refer to the Primary Key of the table

Updating with Expressions

Can use expressions inside an update:

```
UPDATE Students  
SET points = points * 1.1  
WHERE points < 115;
```

If you skip the WHERE clause – will update every row

Our server uses Safe Mode which prevents UPDATES without WHERE
And requires WHERE to refer to the Primary Key of the table

Deleting Data

DELETE statement deletes one or more rows

Removes the entire row – cannot specify individual columns

```
DELETE FROM Universities  
WHERE uni_name = 'UCLAN';
```

If you skip the WHERE clause – will delete every row

Our server uses Safe Mode which prevents DELETE without WHERE
And requires WHERE to refer to the Primary Key of the table