**STRUCTURED QUERY LANGUAGE**

<https://www.microsoft.com/en-us/sql-server/sql-server-editions-express>

Structured Query Language (SQL) is a scripting language that consists of specialized characters, symbols, and syntax to provide instructions for the DBMS. These instructions are termed "statements". There are different categories of SQL statements that command the DBMS to affect, organize, and retrieve the data within the database:

Data Definition Language (DDL)

Data Manipulation Language (DML)

Data Query Language (DQL)

Data Control Language (DCL)

Data Administration Commands (DAC)

Transactional Control Commands (TCC)

The scope for this Basic Database and SQL Course will be to discuss the common goal and usage of SQL in general.

**MYSQL**

MySQL is an open source Relational Database Management System (RDBMS) publicly released in 1998, and eventually acquired by Oracle.

MySQL provides Create, Retrieve, Update, and Delete (CRUD) capabilities and supports features to further configure and denote data relationships to appropriately manage a robust database However, unlike many competing technologies, MySQL is offered free under the GNU General Public License (GPL).

MySQL's success is largely due to an extremely large following of developers and a like-minded community.

**THE MOST COMMONLY USED DATA TYPES FOR SQL ARE:**

**Character(n):**The character string with a specified fixed-value of bits

**Varchar(n):**A character string with a varying number of bits

**Binary:**A binary string with a specified fixed-value of bits

**Boolean:**A descriptive value consisting of either a yes or a no value

**Integer (INT):**A numeric value without a decimal that can take up no more than 4 bytes of data storage

**Smallint:**A numeric value without a decimal that can take up no more than 2 bytes of data storage

**Bigint:**A numeric value without a decimal that can take up no more than 8 bytes of data storage

**Tinyint:**A numeric value without a decimal that can take up no more than 1 byte of data storage.

**Decimal:**Numeric value with a fixed-decimal precision

**Float, Real, or Double:**Numeric value with an approximate decimal precision

**Date:**Stores year, month, and day values

**Time:**Stores hour, minute, and second values

**Timestamp:**Stores year, month, day, hour, minute, and second values

**Array:**A collection of values with a specified fixed-length

There are generally three commonly used keys in a relational database: Unique key, Primary key, and Foreign key.

**UNIQUE KEYS**

Unique keys are used to ensure that a table will maintain unique entries within a specific column. Consider a bank account number. Two customers with the very same bank account number would certainly become a problem, so a database may have the Bank Account field set to "unique" in order to prevent more than one record set (customer) from being assigned a bank account number that's already been taken. It is important to note that the Unique key may be assigned a Null value. A table can be assigned more than one Unique key, and the value contained within the unique field may change freely as long as it still complies with the previous criteria.

**PRIMARY KEYS**

Primary keys are also used to ensure that the data within a column remains unique. However, there are more restrictive characteristics associated with this key. A table may be created containing only a single Primary key. This Primary key field would have the constraint to never accept Null values. The data type assigned for a Primary key may be either an Integer or String value. The most restrictive thing about the Primary key is that once this field has been assigned a value, that value ought not be changed throughout the entire lifetime of the database.

Oftentimes, the Primary key is assigned a special configuration property: an Auto Incrementing setting. Using this feature, the Primary key would have to be assigned an integer data type as the Auto Increment setting would add 1 to the previous record set number and assign the sum to the new record when the new record is being generated, thus automatically ensuring a unique primary value is being maintained.

**FOREIGN KEYS**

Foreign keys inform the database that the particular data contained within this column may be associated with data from another table’s field. This association is termed a Relationship. A foreign key is essentially a marker informing the database that there will be a link made from this field that will eventually be linked together with another table’s marker. Abstractly, you can imagine the foreign key as a thumb tack with a piece of string attached to it that trails off of the table to another tack pinned to a different field within another table. With this, the two tables can now pass information over their connected strings more quickly than without, should specific configurations be provided. These additional instructions are commonly referred to as a Foreign key “Constraint”. These special specifications are rules that the database must abide by in order to ensure that the data assigned within the particular field is always related with the values assigned within the field that has referenced it.

This process is known as Referential Integrity. Referential Integrity is a process where the database is designed to ensure that all table relationships remain consistent.

If a software attempts to provide instructions to the database that would break any constraint or violate the referential integrity, the database will provide an error message, cease to function until the matter has been corrected, or just ignore the command altogether. It is the developer's responsibility to ensure that his or her software interfaces and provides sound instructions to the database engine.

3 Relationship Types:

**ONE-TO-MANY**

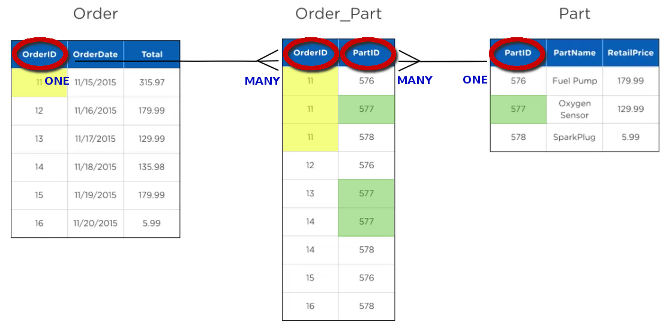
The most common relational type is One-to-Many. One-to-Many denotes how a particular row on one table will relate with multiple rows of data from another table. As the name implies, this is strictly a scenario where a single record of data leads to numerous possible rows from another table.

**ONE-TO-ONE**

A One-to-One relationship ensures that just one record from each side of the relationship will ever point to a single corresponding record from another table. This kind of relationship is not as common in a relational database as generally whenever this type of constraint is needed, the two tables could simply be combined into one larger table instead, however there can be specific occasions where combining into a larger table may be prohibited

**MANY-TO-MANY**

The Many-to-Many relational type means that one particular row in a table can relate with many records from another table or vice-versa. The dilemma with the nature of a many-to-many design is that in a relational database, these tables cannot truly point to one another directly and must be indirectly associated via an additional table called a “Junction table”. With the assistance of this special intermediary table, the Many-to-Many relationship is now essentially split up into two manageable One-to-Many relationships. The two indirectly associated tables may now have single rows of data point to multiple rows within the special Junction table safely.

****

**STEP: 41**

With the advantage of this modeling software, engineers can use symbols to represent a particular relationship type so that they can quickly discover and correct the functionality failures of their design. These special symbols are commonly referred to as Crow’s notation.

use db\_zoo;

create table tbl\_animalia (

animalia\_id INT primary key not null identity (1,1),

animalia\_type Varchar(50) not null

);

Insert into tbl\_animalia

(animalia\_type)

Values

('vertabrate'),

('invertebrate')

;

select \* from tbl\_animalia;

create table tbl\_class (

class\_id int primary key not null identity (100,1),

class\_type varchar(50) not null

);

insert into tbl\_class

(class\_type)

values

('bird'),

('reptilian'),

('mammal'),

('arthrod'),

('fish'),

('worm'),

('cnidaria'),

('echinoderm')

;

select \* from tbl\_class;

select replace(class\_type, 'birds', 'bird') from tbl\_class;

select class\_type from tbl\_class where class\_type = 'bird';

select upper(class\_type) from tbl\_class where class\_type = 'bird';

create table tbl\_persons (

persons\_id int primary key not null identity (1,1),

persons\_fname varchar(50) not null,

persons\_lname varchar(50) not null,

persons\_contact varchar(50) not null

);

insert into tbl\_persons

(persons\_fname, persons\_lname, persons\_contact)

values

('bob', 'smith', '232-345-5768'),

('mary', 'ann', '232-345-5768'),

('tex', 'burns', '232-345-5768'),

('gerry', 'kerns', '232-345-5768'),

('sally', 'fields', '232-345-5768')

;

select \* from tbl\_persons;

select persons\_fname, persons\_lname, persons\_contact from

tbl\_persons where persons\_lname like 'ke%';

update tbl\_persons set persons\_fname = 'mars' where persons\_fname='bob';

select persons\_fname, persons\_lname, persons\_contact from

tbl\_persons where persons\_lname like '\_u%s';

select persons\_fname, persons\_lname, persons\_contact from

tbl\_persons where persons\_lname like '\_u%s';

select persons\_fname as 'First';

select \* from tbl\_persons

create table tbl\_order (

order\_id int primary key not null identity (1,1),

order\_type varchar(50) not null

);

select \* from tbl\_order

Create table tbl\_care (

care\_id varchar(50) primary key not null,

care\_type varchar(50) not null,

care\_specialist int not null

);

select \* from tbl\_nutrition

Create table tbl\_nutrition (

nutrition\_id int primary key not null identity(2200,1),

nutrition\_type varchar(50) not null,

nutrition\_cost money not null

);

create table tbl\_habitat (

habitat\_id int primary key not null identity(5000,1),

habitat\_type varchar (50) not null,

habitat\_cost money not null

);

select \* from tbl\_habitat

create table tbl\_specialist (

specialist\_id int primary key not null identity (1,1),

specialist\_fname varchar(50) not null,

specialist\_lname varchar(50) not null,

specialist\_contact varchar(50) not null

);

select \* from tbl\_specialist

Insert into tbl\_order

(order\_type)

Values

('carnivore'),

('herbivore'),

('omnivore')

;

select \* from tbl\_order;

insert into tbl\_care

(care\_id, care\_type, care\_specialist)

values

('care\_0', 'replace the straw', 1),

('care\_1', 'repair or replace broken toys',4),

('care\_2', 'bottle feed vitamins', 1),

('care\_3', 'human contact\_pet subject', 2),

('care\_4', 'clean up animal waste',1),

('care\_5', 'move subject to excercise pen' ,3),

('care\_6', 'drain and refill aquarium', 1),

('care\_7', 'extensive dental work', 3)

;

select \* from tbl\_care;

insert into tbl\_nutrition

(nutrition\_type, nutrition\_cost)

values

('raw fish', 1500),

('living rodents', 600),

('mixture of fruit and rice', 800),

('warm bottle of milk', 600),

('syringe feed broth', 600),

('lard and seed mix', 300),

('aphids', 150),

('vitamins and marrow', 3500)

;

select \* from tbl\_nutrition;

insert into tbl\_habitat

(habitat\_type, habitat\_cost)

values

('tundra', 40000),

('grassy gnoll with trees', 12000),

('10 ft pond and rocks', 30000),

('icy acquarium with snowy facade', 50000),

('short grass, shade, and moat', 50000),

('netted forest atrium', 10000),

('jungle vines and winding branches', 15000),

('cliff with shaded cave', 15000)

;

select \* from tbl\_habitat;

insert into tbl\_specialist

(specialist\_fname, specialist\_lname, specialist\_contact)

values

('margaret', 'nyguen', '384-576-2899'),

('mary', 'fischer', '384-784-4856'),

('arnold', 'holden', '385-475-3944'),

('kem', 'byesan', '384-485-4855'),

('delmonte', 'fyedo', '768-288-3749')

;

select \* from tbl\_specialist;

create table tbl\_species (

species\_id int primary key not null identity (1,1),

species\_name varchar(50) not null,

species\_animalia int not null constraint fk\_animalia\_id foreign key references tbl\_animalia(animalia\_id) on update cascade on delete cascade,

species\_class int not null constraint fk\_class\_id foreign key references tbl\_class(class\_id) on update cascade on delete cascade,

species\_order int not null constraint fk\_order\_id foreign key references tbl\_order(order\_id) on update cascade on delete cascade,

species\_habitat int not null constraint fk\_habitat\_id foreign key references tbl\_habitat(habitat\_id) on update cascade on delete cascade,

species\_nutrition int not null constraint fk\_nutrition\_id foreign key references tbl\_nutrition(nutrition\_id) on update cascade on delete cascade,

species\_care varchar(50) not null constraint fk\_care\_id foreign key references tbl\_care(care\_id) on update cascade on delete cascade

);

Insert into tbl\_species

(species\_name, species\_animalia, species\_class, species\_order, species\_habitat, species\_nutrition, species\_care)

values

('brown bear', 1, 102, 3, 5007, 2200, 'care\_1'),

('jaguar', 1, 102, 1, 5007, 2200, 'care\_4'),

('penguin', 1, 100, 1, 5003, 2200, 'care\_6'),

('ghost bat', 1, 102, 1, 5007, 2204, 'care\_2'),

('chicken', 1, 100, 3, 5001, 2205, 'care\_4'),

('panda', 1, 102, 3, 5006, 2202, 'care\_5'),

('bobcat', 1, 102, 1, 5000, 2201, 'care\_4'),

('grey wolf', 1, 102, 1, 5000, 2201, 'care\_4')

;

select \* from tbl\_species

select

a1.species\_name, a2.animalia\_type,

a3.class\_type, a4.order\_type, a5.habitat\_type,

a6.nutrition\_type, a7.care\_type

From tbl\_species as a1

inner join tbl\_animalia a2 on a2.animalia\_id = a1.species\_animalia

inner join tbl\_class a3 on a3.class\_id = a1.species\_class

inner join tbl\_order a4 on a4.order\_id = a1.species\_order

inner join tbl\_habitat a5 on a5.habitat\_id = a1.species\_habitat

inner join tbl\_nutrition a6 on a6.nutrition\_id = a1.species\_nutrition

inner join tbl\_care a7 on a7.care\_id = a1.species\_care

where species\_name = 'brown bear'

;

select

a1.species\_name, a2.animalia\_type,

a3.class\_type, a4.order\_type, a5.habitat\_type,

a6.nutrition\_type, a7.care\_type

From tbl\_species as a1

inner join tbl\_animalia a2 on a2.animalia\_id = a1.species\_animalia

inner join tbl\_class a3 on a3.class\_id = a1.species\_class

inner join tbl\_order a4 on a4.order\_id = a1.species\_order

inner join tbl\_habitat a5 on a5.habitat\_id = a1.species\_habitat

inner join tbl\_nutrition a6 on a6.nutrition\_id = a1.species\_nutrition

inner join tbl\_care a7 on a7.care\_id = a1.species\_care

where species\_name = @animalName

;

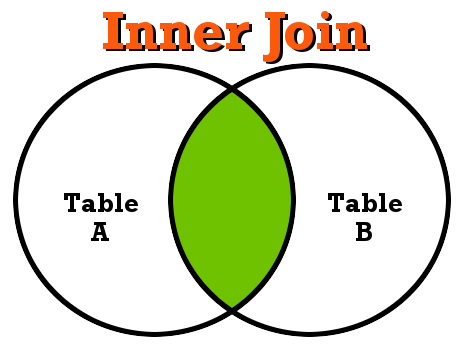
**DATABASE & SQL COURSE**

11 days of access to the LMS remaining.  You are 49 days behind completing the boot camp in time—we recommend putting in extra study time!

Course Progress: 71%

Boot Camp Progress: 46%

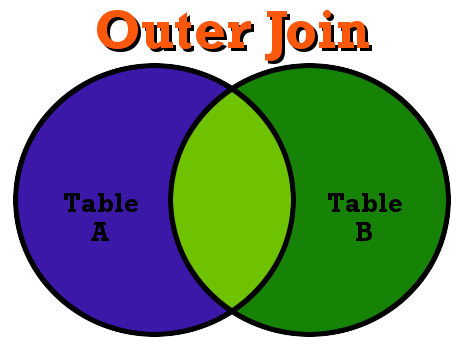
**SQL JOIN STATEMENTS**

****

The Inner Join statement is used to return data from all rows where there is matching data shared between the two table columns.

**SELECT \* FROM table\_A**

**SELECT \* FROM table\_A INNER JOIN table\_B ON table\_B.id = table\_A.id;**

****

The OUTER JOIN statement is used to return all records of data from both tables despite whether there is any actual relational link between the two tables.

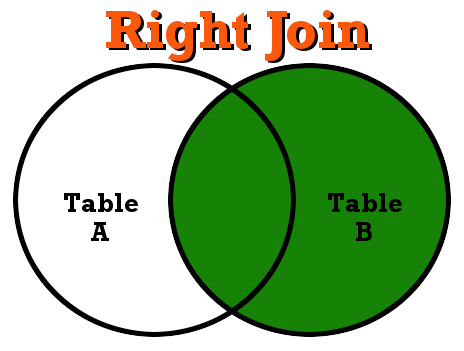
**SELECT table\_A.username, table\_A.fullname, table\_B.zipcodes**

**FROM table\_A**

**FULL OUTER JOIN table\_B**

**WHERE table\_B.id = table\_A.id;**

**STEP: 62**

****

The RIGHT JOIN statement is used to return data from from the right table and any related data from the left table and in the event there is data that does not match it will be reflected by a Null value.

**SELECT table\_A.wholesale, table\_A.products, table\_B.vendor, table\_B.contact**

**FROM table\_A**

**RIGHT JOIN table\_B ON table\_B.id = table\_A.id;**

**WHERE table\_A.id LIKE 'de%';**

CREATE TABLE Emp\_Cont (

Emp\_ID INT PRIMARY KEY not null IDENTITY(1,1),

First\_Name VARCHAR(50) not null,

Last\_Name VARCHAR(50) not null,

Address\_1 VARCHAR(50) not null,

Phone VARCHAR(50) not null

);

INSERT INTO Emp\_Cont

(First\_Name, Last\_Name, Address\_1, Phone)

VALUES

('David', 'Dee', '321 Birch St.', '303 867-5309'),

('Jay', 'Smith', '2052 Dallas St.', '303 555-5309'),

('Beth', 'Crabby', '5040 Vonn St.', '303 555-5309'),

('Don', 'Spongey', '6672 Robe St.', '303 555-5309'),

('David', 'Wallace', '554 Garp St.', '303 555-5309')

;