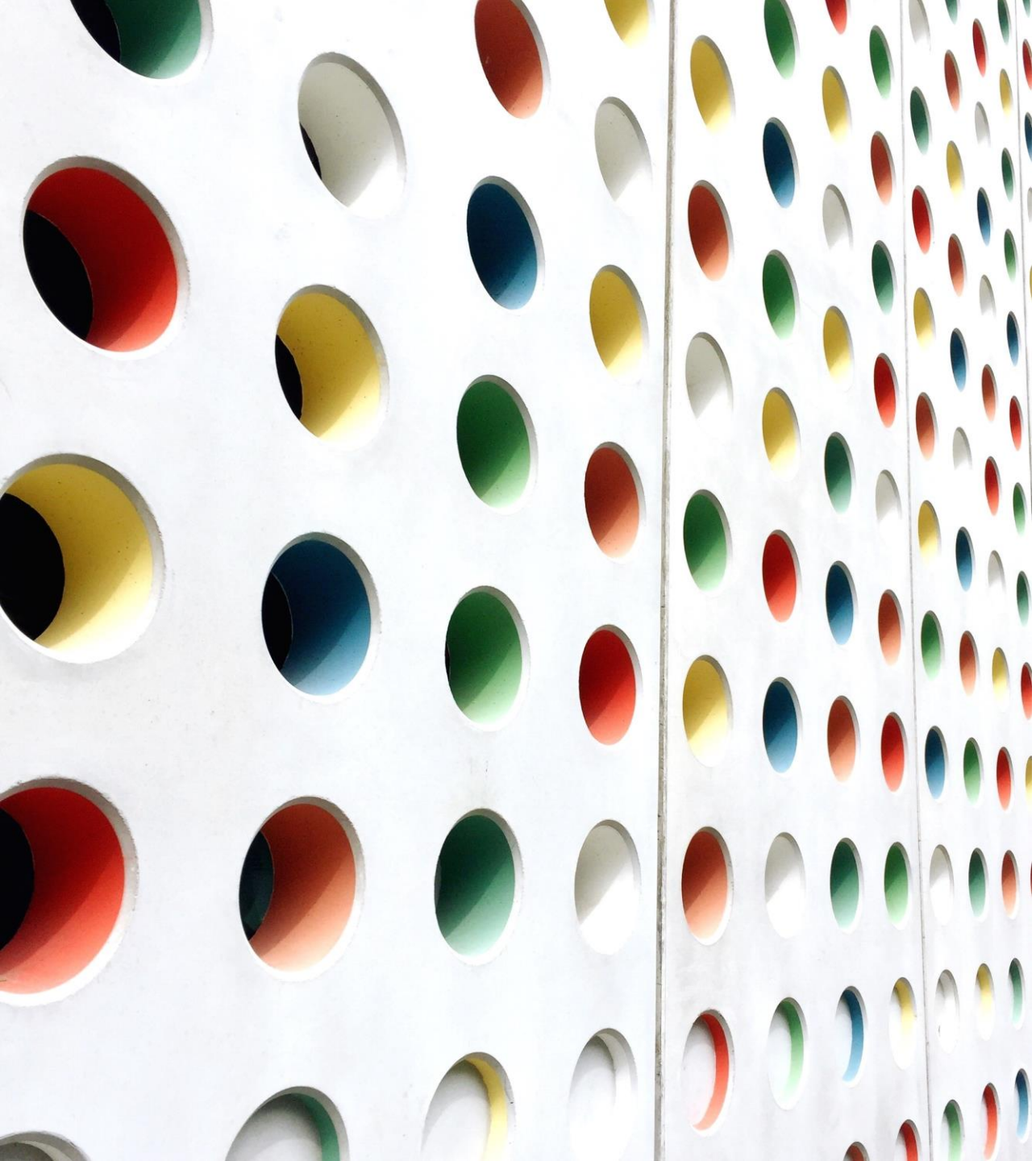


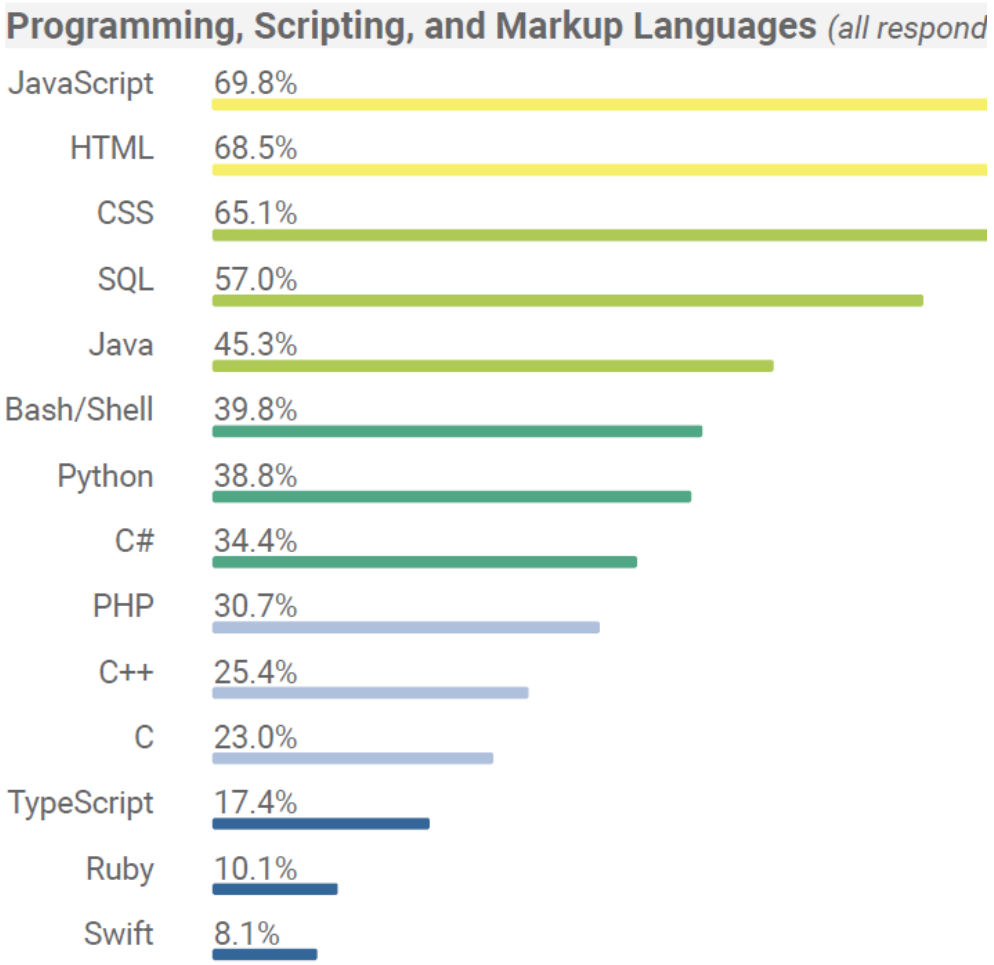
Introduction to SQL

GRACE SUN

DIRECTOR OF INSTITUTIONAL RESEARCH

SQL

- Structured Query Language (SQL) is one of the main query languages used to access data within relational databases.
- SQL is designed to efficiently handle large amounts of data, which is a highly valued capability for organizations.
- Experienced SQL programmers are in high demand.



Class Objectives

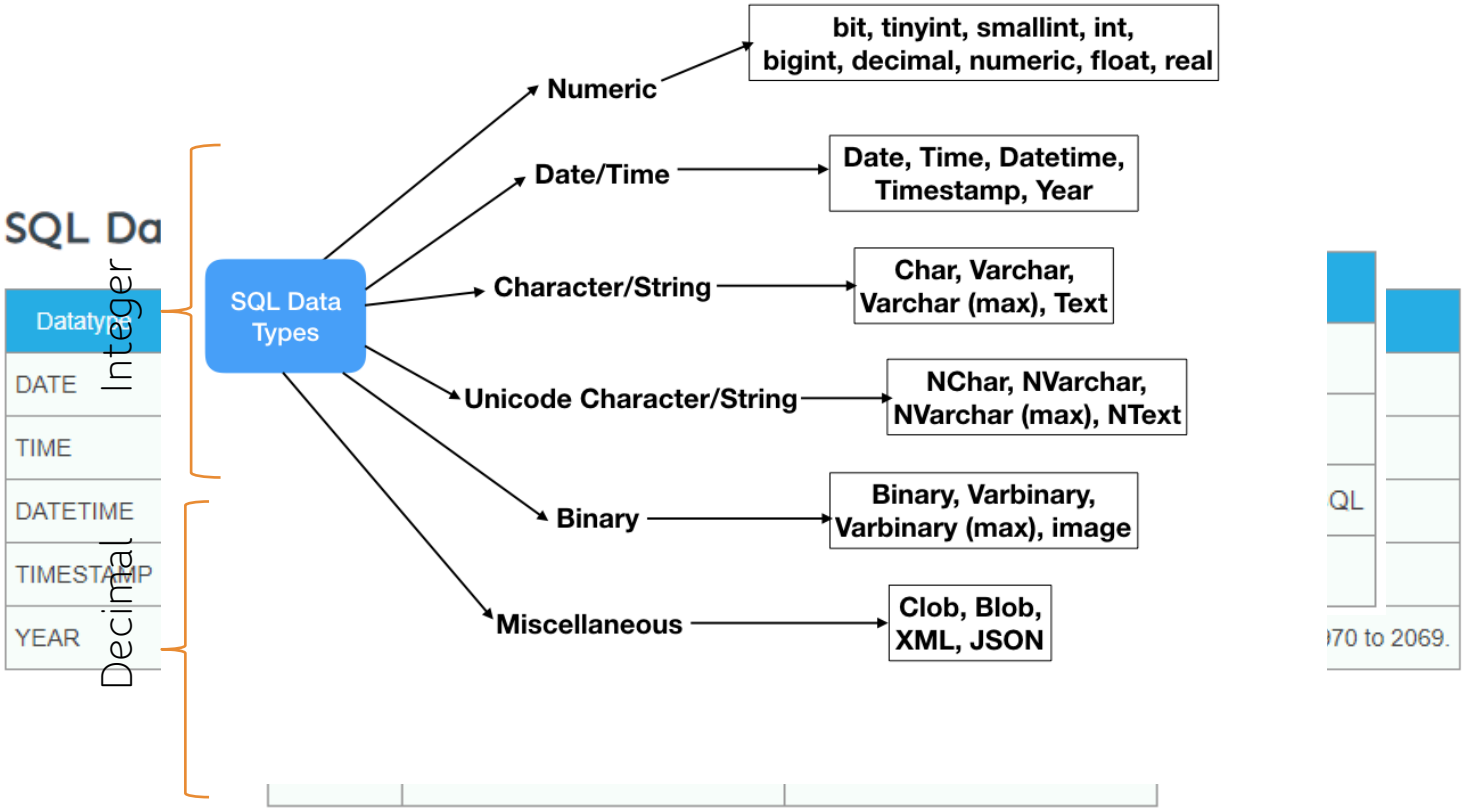
DEFINE SQL DATA TYPES, PRIMARY KEYS, AND UNIQUE VALUES

QUERY DATA FROM A DATABASE

ARTICULATE THE FOUR BASIC FUNCTIONS OF CRUD AND APPLY THEM TO A DATABASE

COMBINE DATA FROM MULTIPLE TABLES USING JOINS.

Data Type


SQL Da
Datatype
DATE
TIME
DATETIME
TIMESTAMP
YEAR
Integer
Decimal
SQL Data Types
Numeric
bit, tinyint, smallint, int, bigint, decimal, numeric, float, real
Date/Time
Date, Time, Datetime, Timestamp, Year
Character/String
Char, Varchar, Varchar (max), Text
Unicode Character/String
NChar, NVarchar, NVarchar (max), NText
Binary
Binary, Varbinary, Varbinary (max), image
Miscellaneous
Clob, Blob, XML, JSON
QL
070 to 2069.

Primary Keys

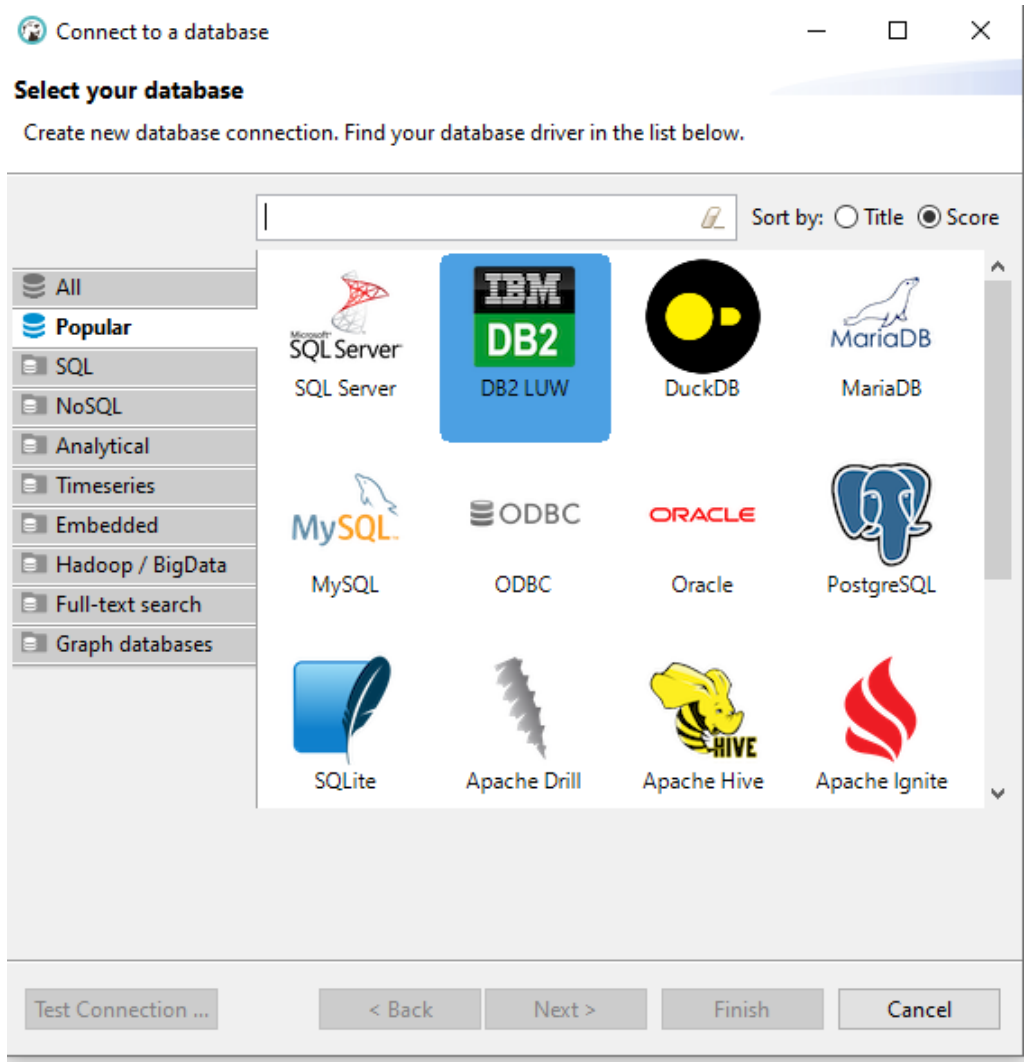
StudentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

Primary Key

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.

If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).


Connect to a database
Select your database
Create new database connection. Find your database driver in the list below.
Sort by: Title Score
All
Popular
SQL
NoSQL
Analytical
Timeseries
Embedded
Hadoop / BigData
Full-text search
Graph databases
SQL Server
DB2 LUW
DuckDB
MariaDB
MySQL
ODBC
Oracle
PostgreSQL
SQLite
Apache Drill
Apache Hive
Apache Ignite
Test Connection ...
< Back
Next >
Finish
Cancel

Tool and Connection

Link: https://dbeaver.io/download/

Connection

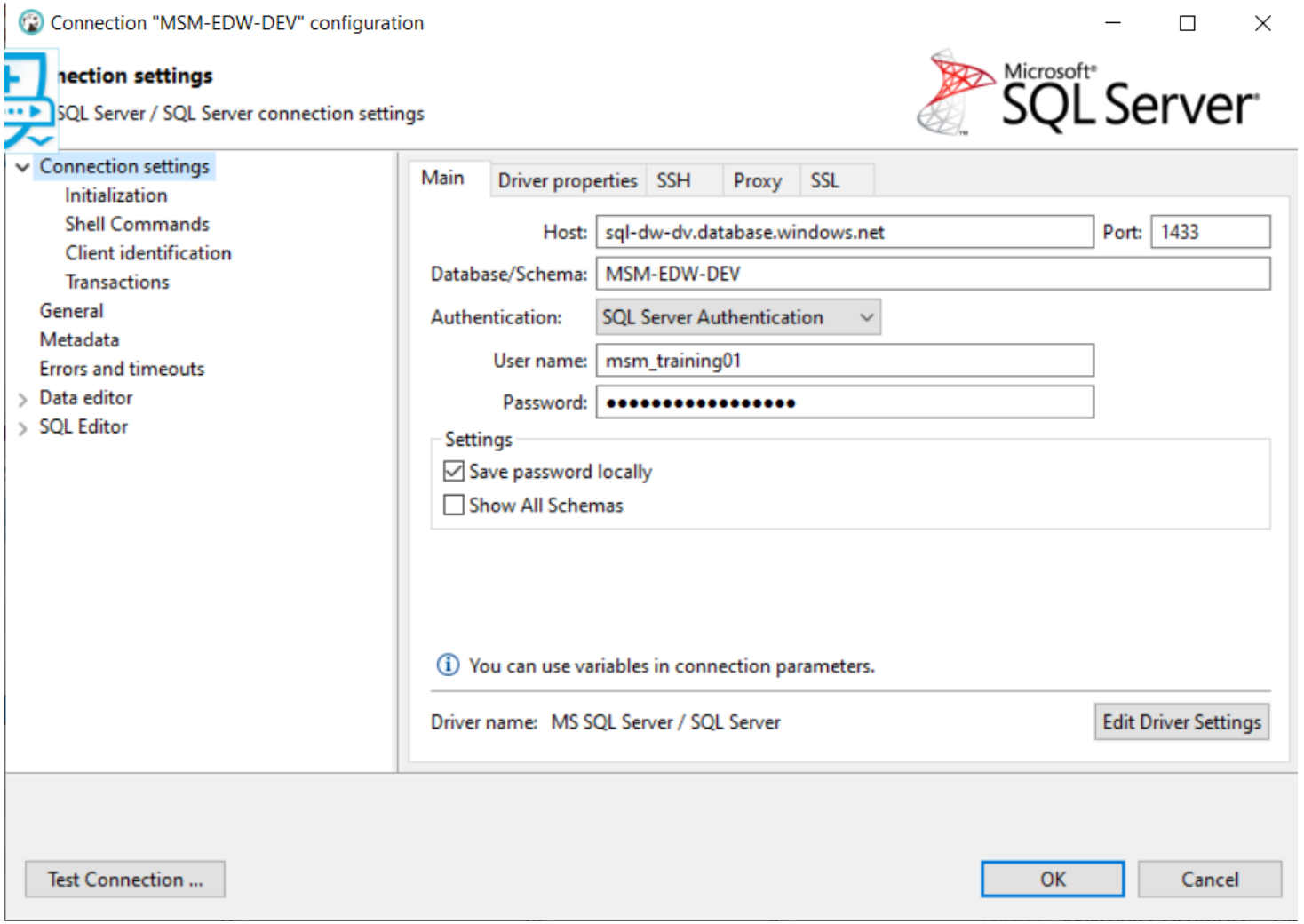
Server Host Name: sql-dw-dv.database.windows.net

Database Name: MSM-EDW-DEV

Port Number: 1433

UserID: msm_training01

Password: 8TVL1pGvBVQL!hd7R



CRUD –Create, Read, Update, Delete

While an unusual acronym, is a set of tools that are persistently used throughout programming. CRUD stands for Create, Read, Update, and Delete.

Create INSERT table info (column1, column2, column3)

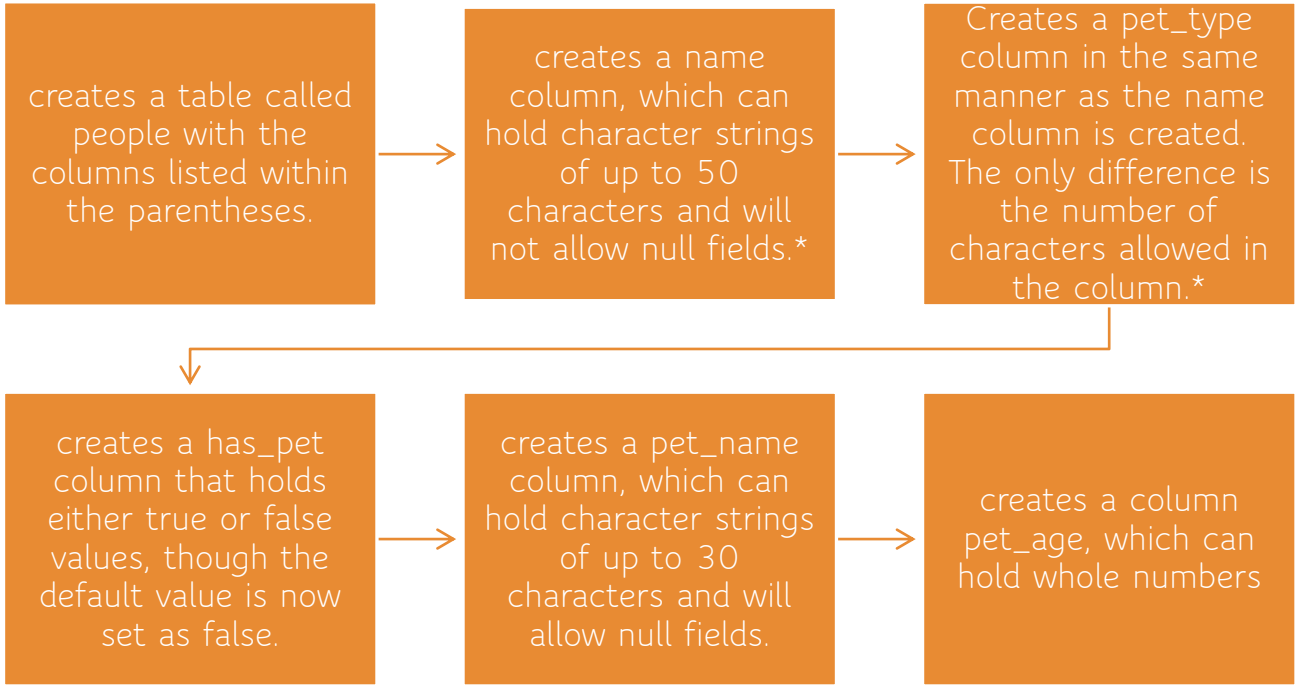
Read SELECT * FROM table

Update UPDATE table SET column1 = VALUE WHERE id = 1

Delete DELETE FROM table WHERE id = 5

These tools are fundamental to all programming languages–not just SQL.

Creating Tables - CREAT




```
create table [MSM-EDW-DEV].TRAINING.people_"yourinitial"(
name varchar(50) not null,
pet_type varchar(10) not null,
has_pet bit default 'FALSE',
pet_name varchar(30),
pet_age int);
```

*The constraint requires the name field to have a value specified.

Creating Tables - INSERT

The query below operates as it reads.

- Data is inserted to the table in the following columns.
- Data is the following, in the same order columns were stipulated.

```
insert into [MSM-EDW-DEV].TRAINING.people_"yourinitial"
(name, pet_type, has_pet, pet_name, pet_age)
values('Jacob','dog',0,'Misty',1),
('Ahmed','rock',0,'Rockington',200),
('Peter','cat',0,'Franklin',2),
('Dave','dog',0,'Queso',1);
```

Extract/Read data

The query below only extracts the data from the column in the table:

```
select *
from [MSM-EDW-DEV].TRAINING.people_"your initial";
```

```
select pet_name
from [MSM-EDW-DEV].TRAINING.people_"your initial";
```

The query below will return only dog(s), with their names that are younger than five years old.

```
select *
from [MSM-EDW-DEV].TRAINING.people_"your initial"
where pet_type='dog'
and pet_age <5;
```

Add /Change Columns and Update

The query below will add a gender and automatic generated ID to the table.

```
Alter table [MSM-EDW-DEV].TRAINING.people_"your initial"
add ID2 int not null identity (1,1) Primary Key,
    pet_gender varchar(30);
```

The query below will add a gender and automatic generated ID to the table.

```
update [MSM-EDW-DEV].TRAINING.people_"your initial"
set pet_gender ='M'
where ID2 =1;

update [MSM-EDW-DEV].TRAINING.people_"your initial"
set pet_gender ='O'
where ID2 =2;

update [MSM-EDW-DEV].TRAINING.people_"your initial"
set pet_gender ='M'
where ID2 =3;

update [MSM-EDW-DEV].TRAINING.people_"your initial"
set pet_gender ='F'
where ID2 =4;
```

Activity I: Creating table

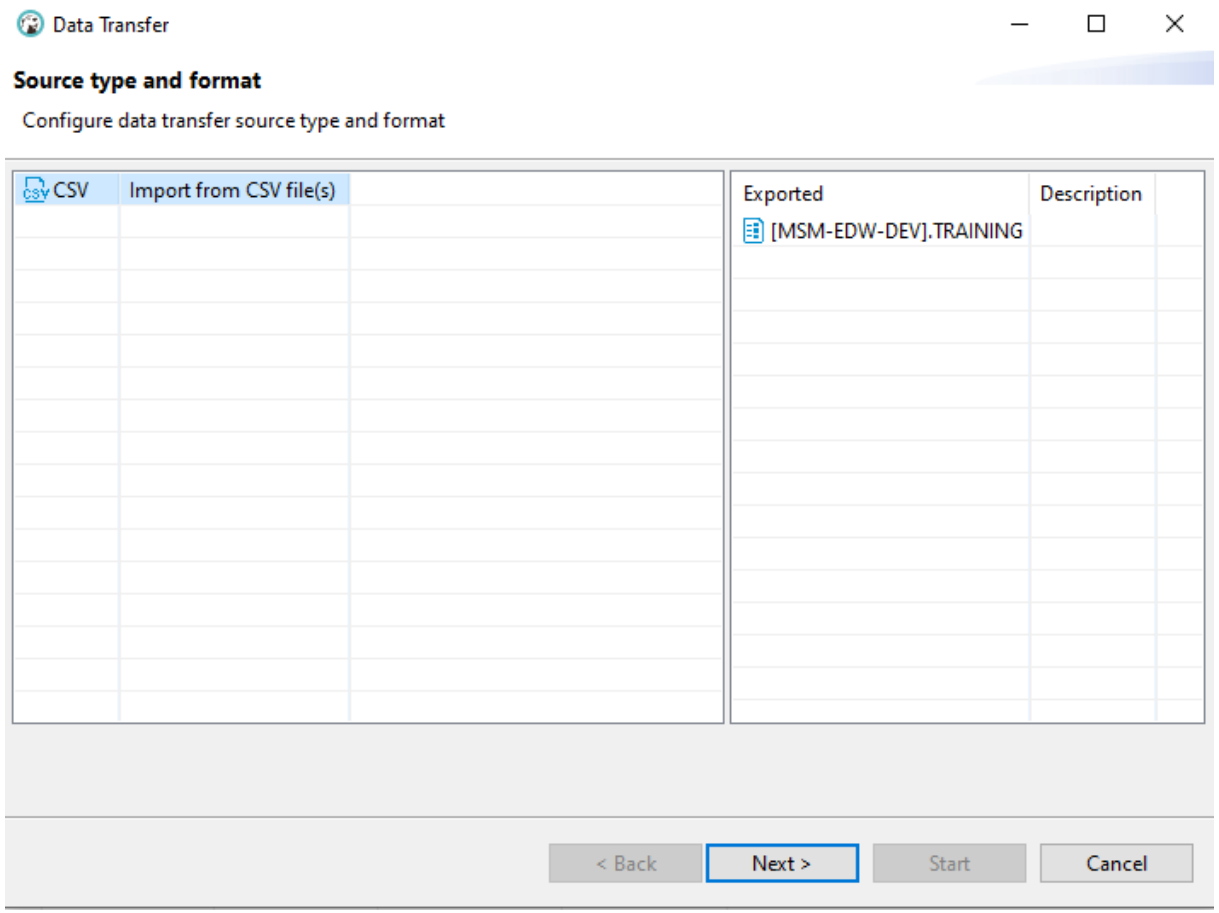
- Using the query tool, create an empty table named cities_"your initial" . Be sure to match the data types!
- Insert data into the new table. The result should match table **on the right**
  - Filter the table to view only cities in Arizona.
  - Filter the table to view only cities with a population of less than 100,000.
  - Filter the table to view California cities with a population of less than 100,000.

	city character varying (30)	state character varying (30)	population integer
1	Alameda	California	79177
2	Mesa	Arizona	496401
3	Boerne	Texas	16056
4	Anaheim	California	352497
5	Tucson	Arizona	535677
6	Garland	Texas	238002

Hints:

- For the fourth question, you will need to use a WHERE clause to filter the original query.
- For the last question, an AND clause will also be necessary

Import Data




```
--Drop table if exists
Drop table TRAINING.bird_song_"your initial";

create table TRAINING.bird_song_"your initial"(
english_name varchar(50),
country varchar(50),
latigude dec(7,4),
longitude dec(7,4));

--View table columns and datatypes
select * from TRAINING.bird_song_"your initial";
```

Wildcard: % and _

Wildcards are used to substitute zero, one, or multiple characters in a string. The keyword **LIKE** indicates the use of a wildcard.

```
SELECT *
FROM actor
WHERE last_name LIKE 'Will%';
```

The % will substitute zero, one, or multiple characters in a query.
For example, all of the following will match: Will, Willa, and Willows.

```
SELECT *
FROM actor
WHERE first_name LIKE '_AN';
```

The _ will substitute one, and only one, character in a query.
_AN returns all actors whose first name contains three letters, the second and third of which are AN.

SQL Functions

Aggregate Functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

Scalar Functions:

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

Activity II

- Create a table called `firepower_"your initial"`. Import the data from `GlobalFirePower.csv` using the Import/Export tool.
- Update the table and add a new primary key column called 'id'
- Find the rows that have a `ReservePersonnel` of 0 and remove these rows from the dataset.
- Let's find which country only has one `FighterAircraft` and take note of it.
- Every country in the world at least deserves one `FighterAircraft`—it only seems fair. Let's add one to each nation that has none.
- Oh no! By updating this column, the values within `TotalAircraftStrength` column are now off for those nations! We need to [add 1] to the original number but not for the country that already had 1 `FighterAircraft`.
- Find the [Averages] for `TotalMilitaryPersonnel`, `TotalAircraftStrength`, `TotalHelicopterStrength`, and `TotalPopulation`, and rename the columns with their designated average.

Bonus

After creating your new nation and some parts of your military strategy, add the average values you calculated to the appropriate columns in the newly created rows. Update their values in any way you wish!

```sql
-- Drop table if exists
DROP TABLE IF EXISTS [MSM-EDW-DEV].TRAINING.firepower;

-- Create new table to import data
CREATE TABLE [MSM-EDW-DEV].TRAINING.firepower (
    country VARCHAR(250),
    ISO3 VARCHAR(30),
    rank INT,
    TotalPopulation BIGINT,
    ManpowerAvailable BIGINT,
    TotalMilitaryPersonnel BIGINT,
    ActivePersonnel BIGINT,
    ReservePersonnel BIGINT,
    TotalAircraftStrength BIGINT,
    FighterAircraft BIGINT,
    AttackAircraft BIGINT,
    TotalHelicopterStrength BIGINT,
    AttackHelicopters BIGINT
);

-- Import data from GlobalFirePower.csv
-- View the table to ensure all data has been imported correctly
SELECT * FROM [MSM-EDW-DEV].TRAINING.firepower;
```

```sql
-- Add primary key
ALTER TABLE [MSM-EDW-DEV].TRAINING.firepower
Add AutoId int NOT NULL IDENTITY (1, 1) Primary key;

-- Delete and update data
DELETE FROM [MSM-EDW-DEV].TRAINING.firepower
WHERE ReservePersonnel = 0;

-- Find countries with one FighterAircraft
SELECT * FROM [MSM-EDW-DEV].TRAINING.firepower
WHERE FighterAirCraft = 1;

-- Make note of them
-- Sir Lanka

-- Give each country with no FighterAircraft one

UPDATE [MSM-EDW-DEV].TRAINING.firepower
SET FighterAircraft = 1
WHERE FighterAircraft = 0;

-- Update TotalAircraftStrength by one to countries that we added FighterAircraft

UPDATE [MSM-EDW-DEV].TRAINING.firepower
SET TotalAircraftStrength = TotalAircraftStrength + 1
WHERE FighterAircraft = 1 AND country != 'Sri Lanka';

-- Select averages and rename columns
SELECT AVG(TotalMilitaryPersonnel) AS AvgTotMilPersonnel,
    AVG(TotalAircraftStrength) AS AvgTotAircraftStrength,
    AVG(TotalHelicopterStrength) AS AvgTotHelicopterStrength,
    AVG(TotalPopulation) AS AvgTotalPopulation
FROM [MSM-EDW-DEV].TRAINING.firepower;

-- Insert new data
INSERT INTO [MSM-EDW-DEV].TRAINING.firepower(Country,
                                             TotalPopulation,
                                             TotalMilitaryPersonnel,
                                             TotalAircraftStrength,
                                             TotalHelicopterStrength)
VALUES ('GlobalLand', 60069024, 524358, 457, 183);

-- View table
SELECT * FROM [MSM-EDW-DEV].TRAINING.firepower;
```

JOIN –INNER JOIN

- INNER JOIN returns records that have matching values in both tables.

- LEFT JOIN returns all records from the left table and the matched records from the right table.

- RIGHT JOIN returns all records from the right table and the matched records from the left table.

- CROSS JOIN returns records that match every row of the left table with every row of the right table. This type of join has the potential to make very large tables.

- FULL OUTER JOIN places null values within the columns that do not match between the two tables, after an inner join is performed.

```sql
-- Drop table if exists
DROP TABLE [MSM-EDW-DEV].TRAINING.players;

-- Create the players table
CREATE TABLE [MSM-EDW-DEV].TRAINING.players (
    player_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hand VARCHAR(30),
    country_code VARCHAR(30)
);

-- Check data import
SELECT *
FROM [MSM-EDW-DEV].TRAINING.players;

-- Create the matches table
CREATE TABLE [MSM-EDW-DEV].TRAINING.matches (
    loser_age DEC(5,2),
    loser_id INT,
    loser_name VARCHAR(50),
    loser_rank INT,
    winner_age DEC(5,2),
    winner_id INT,
    winner_name VARCHAR(50),
    winner_rank INT
);

-- Check data import
SELECT *
FROM [MSM-EDW-DEV].TRAINING.matches;

-- Perform an INNER JOIN on the two tables
SELECT distinct players.first_name, players.last_name, players.hand, matches.loser_rank
FROM [MSM-EDW-DEV].TRAINING.matches
INNER JOIN [MSM-EDW-DEV].TRAINING.players ON
players.player_id=matches.loser_id;

-- Alternative solution:
-- Perform an INNER JOIN on the two tables
SELECT distinct p.first_name, p.last_name, p.hand, m.loser_rank
FROM [MSM-EDW-DEV].TRAINING.matches AS m
INNER JOIN [MSM-EDW-DEV].TRAINING.players AS p ON
p.player_id=m.loser_id;
```

Inner Join

In our given scenario player_id column of the players table and the loser_id/winner_id columns of the maches table have matching values.

- In that case we can join these tables together utilizing the INNER JOIN:

Activity III

create two new tables named players and seasons_stats

create the tables, and then import the corresponding data from Player.csv and Seasons_Stats.csv

Perform joins that will generate the following outputs.

- Basic Information Table:

id integer	player character varying	height integer	weight integer	college character varying	born integer	position character varying	tm character varying
0	Cliff Barker	188	83	University of Kentucky	1921	SG	INO
0	Cliff Barker	188	83	University of Kentucky	1921	SG	INO
0	Cliff Barker	188	83	University of Kentucky	1921	SG	INO
1	Ralph Beard	178	79	University of Kentucky	1927	G	INO
1	Ralph Beard	178	79	University of Kentucky	1927	G	INO
2	Charlie Black	196	90	University of Kansas	1921	F-C	TOT

Activity III – Cont.

Percents Stats:

player_id integer	college character varying	year numeric	position character varying	two_point_percentage numeric	fg_percentage numeric	ft_percentage numeric	ts_percentage numeric
0	University of Kentucky	1950	SG	0.372	0.372	0.708	0.435
0	University of Kentucky	1951	SG	0.252	0.252	0.649	0.322
0	University of Kentucky	1952	SG	0.298	0.298	0.588	0.343
1	University of Kentucky	1950	G	0.363	0.363	0.762	0.422
1	University of Kentucky	1951	G	0.368	0.368	0.775	0.435
2	University of Kansas	1950	F-C	0.278	0.278	0.651	0.346


KEEP
CALM
AND
CODE
SQL