Intro To SQL

...

DAY 1

WHY SQL ?

We think of SQL is old and outdated.

The old part is true. SQL is 50+ years old !!

But does that make it irrelevant ??

But why should someone who **wants to get a job in data** spend time learning this 'ancient' language in 2021?

Why not spend all your time mastering Python/R, or **focusing on 'sexier' data skills**, like Deep Learning, Scala, and Spark?

While knowing the fundamentals of a more general-purpose language like Python or R is critical, ignoring SQL will make it much harder to get a job in data. Here are **three key reasons why**:

1. SQL is everywhere

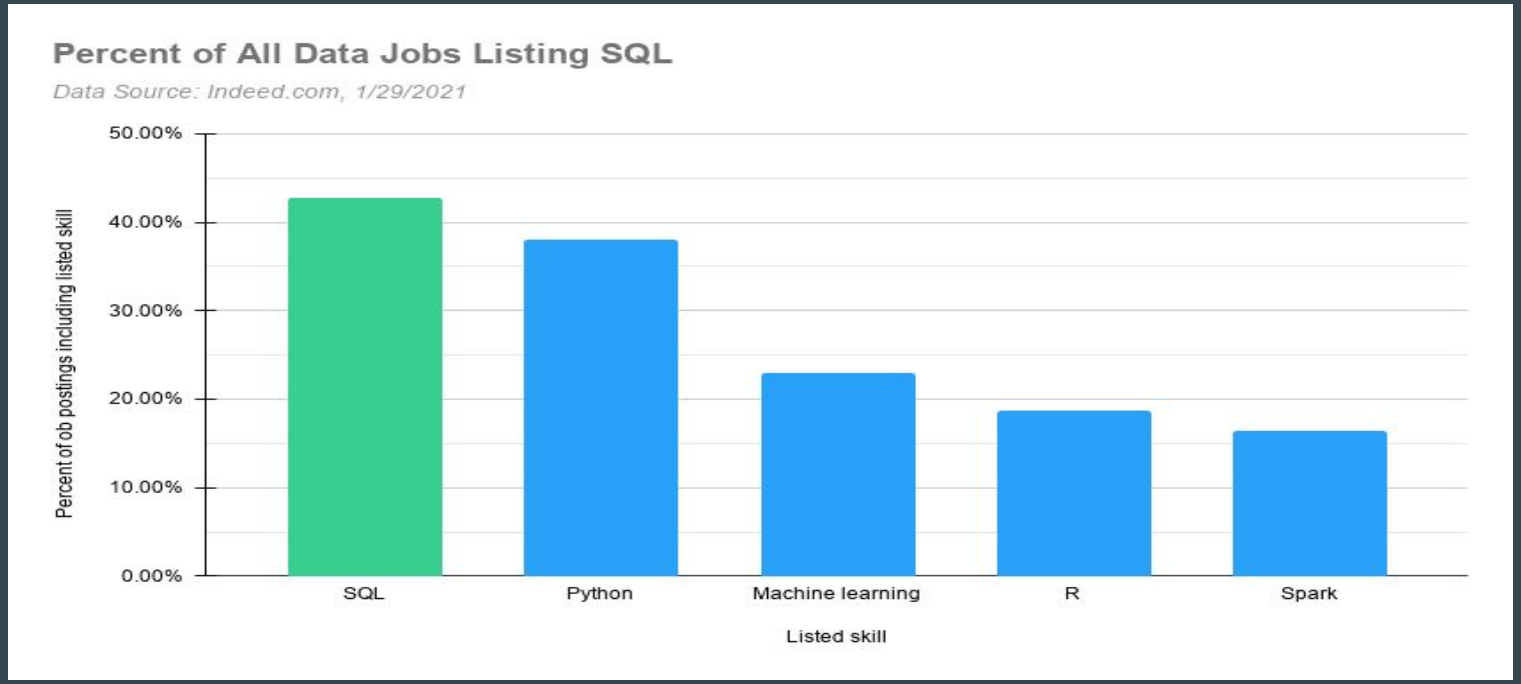
- Almost all of the biggest names in tech use SQL. Uber, Netflix, Airbnb — the list goes on.

- Even within companies like Facebook, Google, and Amazon, which have built their own high-performance database systems, data teams use SQL to query data and perform analysis.

- A quick job search on LinkedIn, for example, will show you that more companies are looking for SQL skills than are looking for Python or R skills.

2. SQL is in demand

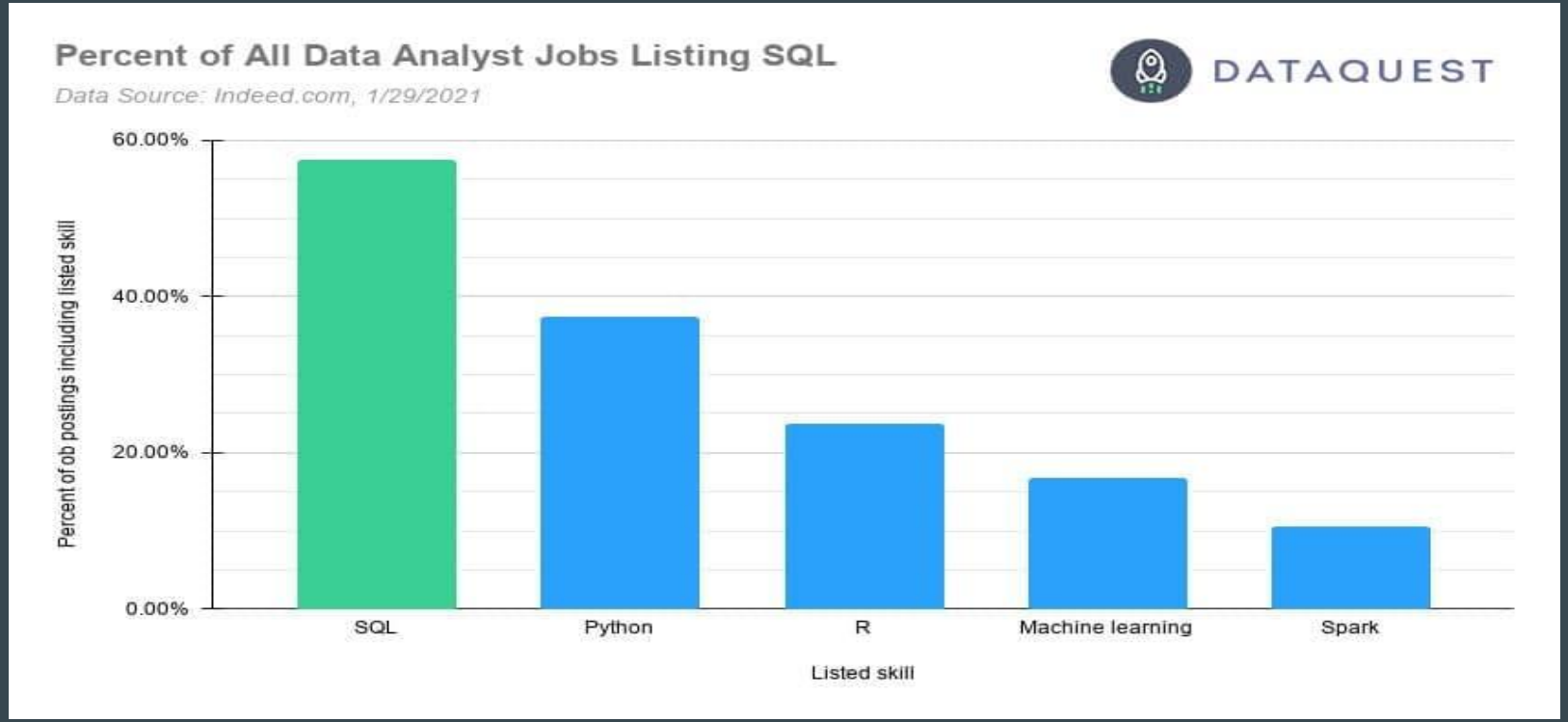
- To demonstrate the importance of SQL specifically in data-related jobs, in early 2021 an analysis was done of more than 32,000 data jobs advertised on Indeed, looking at key skills mentioned in job ads with 'data' in the title.



- **This demand for SQL is increasing**
    - In **2017** :- SQL appeared in 35.7% of all job postings
    - In **2021** :- SQL appeared in 42.7% of all job postings
    - SQL was listed as a key skill in these job ads.

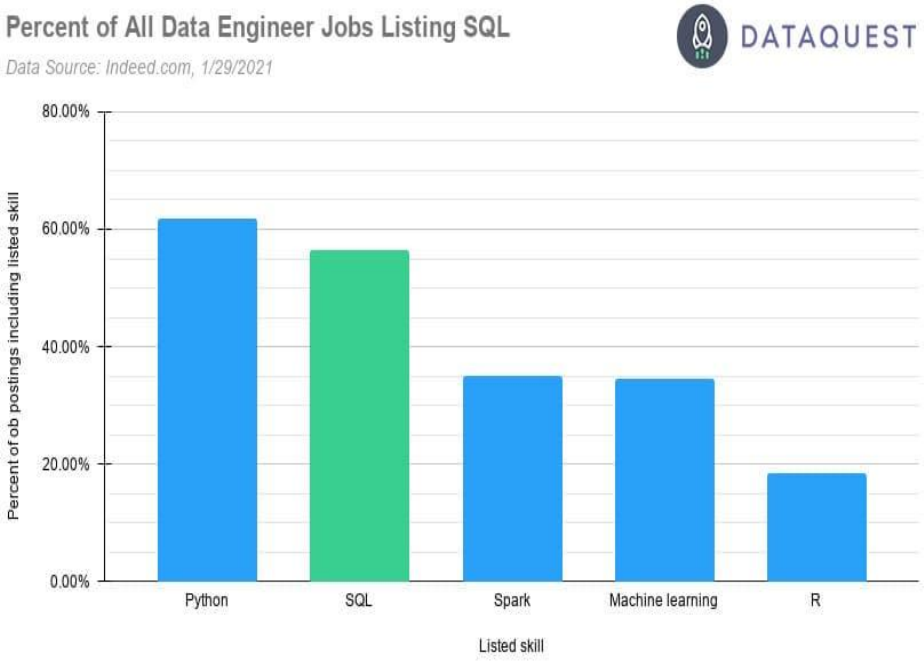
If you're looking for your first job in data....

Most entry-level jobs in data are Data Analyst roles. Take a look at jobs ads with 'data analyst' in the title, and those numbers are even more conclusive:



In more advanced roles, SQL skills are critical

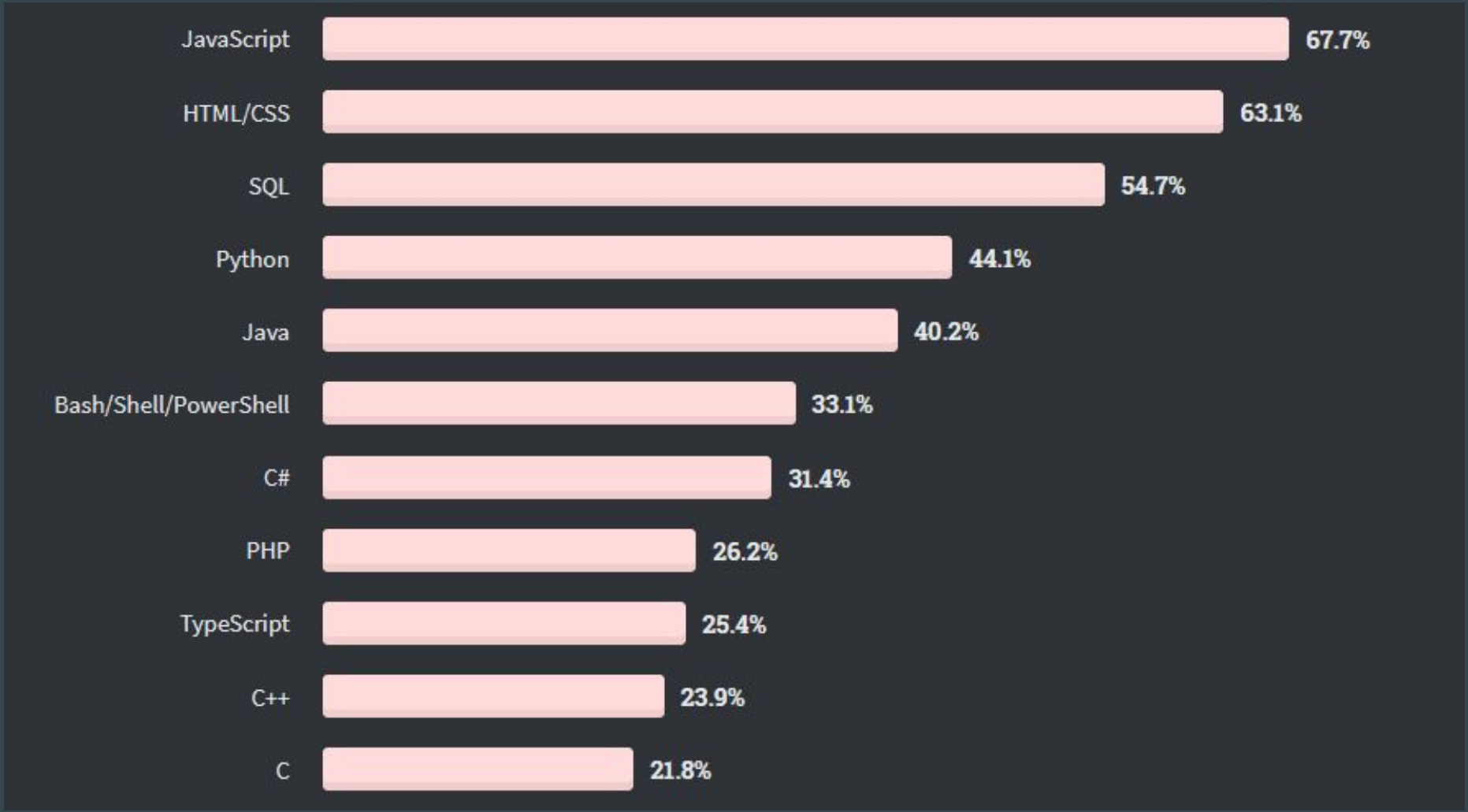
- Here is an analysis on "Data Scientist" and "Data Engineer" job postings.
- SQL listed in 58.2% of data scientist job postings, and 56.4% of data engineer job postings



3. SQL is still the top language for data work

SQL is more popular among data scientists and data engineers than even Python or R. In fact, it's one of the most-used languages in the entire tech industry!

The "most used" technologies **from StackOverflow's 2020 developer survey,** SQL eclipses even Python in terms of popularity. In fact, it's the third-most-popular language among all developers.

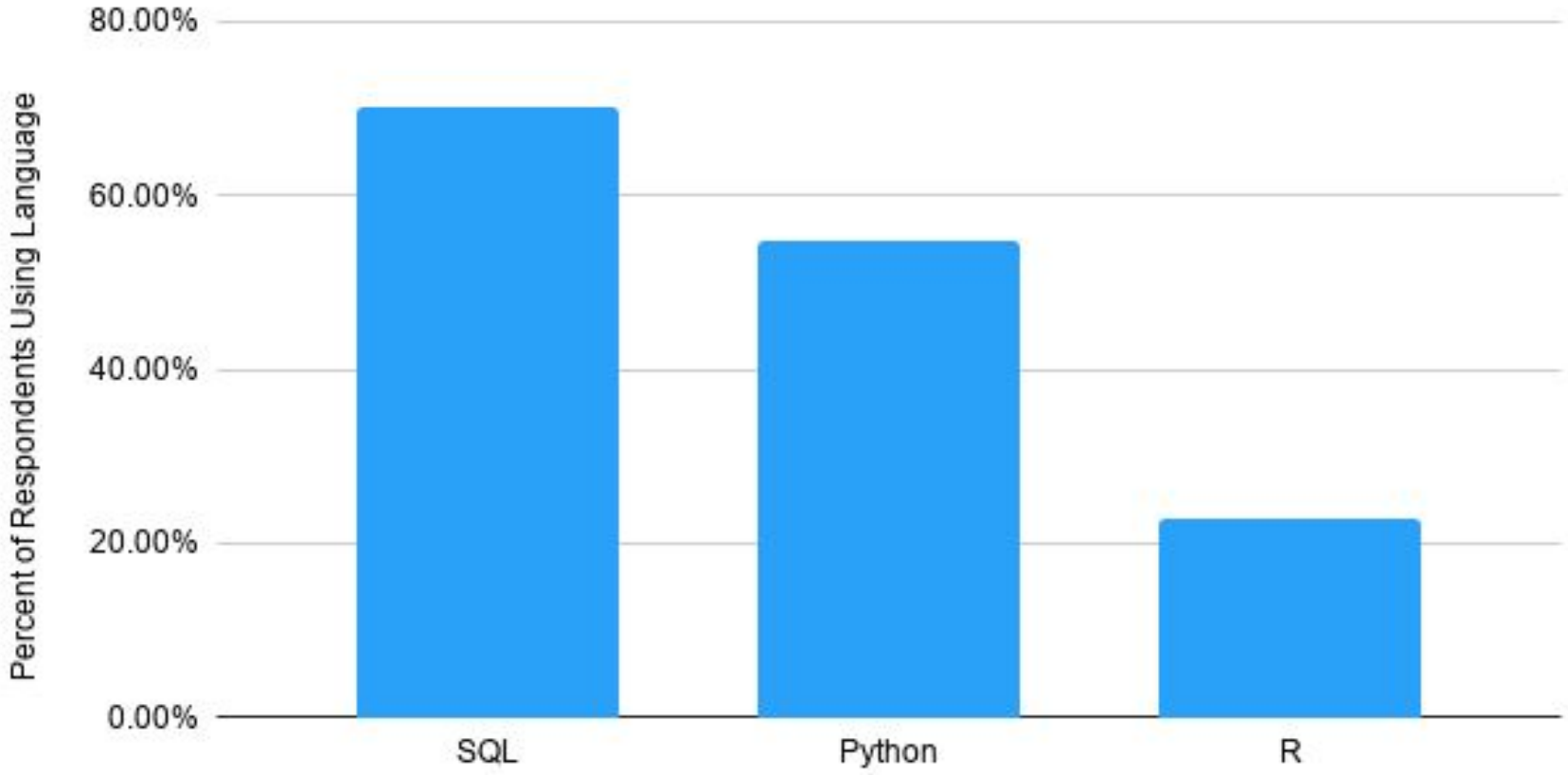

JavaScript
67.7%
HTML/CSS
63.1%
SQL
54.7%
Python
44.1%
Java
40.2%
Bash/Shell/PowerShell
33.1%
C#
31.4%
PHP
26.2%
TypeScript
25.4%
C++
23.9%
C
21.8%

But that was a survey on all jobs....

Let us filter things down a bit and look into specifically with jobs **within the field of data science.**

Among developers who work with data (including **data scientists, data analysts, database administrators, data engineers,** etc.), **more than 70% use SQL** — more than any other language.

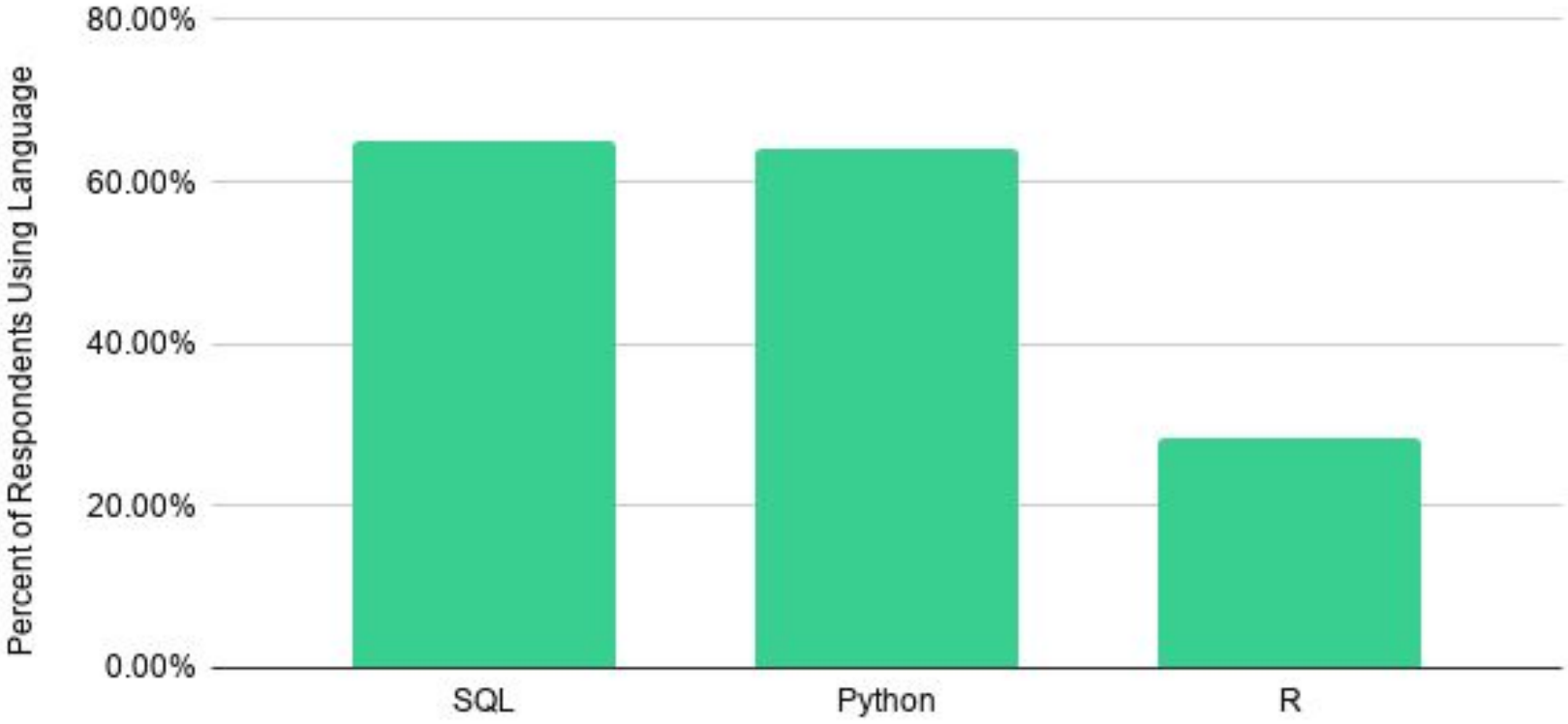
What Languages Do People with Jobs in Data Use?



Still not convinced?

If we filter down still further, into just data scientists and analysts, we can see that SQL is still the most popular technology. **65%** of data scientists and data analysts said they used SQL, compared to 64% for Python, and 28% for R.

What Languages Do People with Data Scientist/Analyst Jobs Use?



In real world, SQL queries look like this :-

dataquest

```sql
WITH invoice_first_track AS
    (
     SELECT
         il.invoice_id invoice_id,
         i.customer_id,
         MIN(il.track_id) first_track_id
     FROM invoice_line il
     INNER JOIN invoice i ON i.invoice_id = il.invoice_id
     GROUP BY 1, 2
    )

SELECT
    album_purchase,
    COUNT(invoice_id) number_of_invoices,
    CAST(COUNT(invoice_id) AS FLOAT) / (
                                        SELECT COUNT(*) FROM invoice
                                       ) percent
FROM
    (
    SELECT
        ifs.*,
        t.album_id,
        CASE
            WHEN
                (
                 SELECT t3.track_id FROM track t3
                 WHERE t3.album_id = t.album_id

                 EXCEPT

                 SELECT il2.track_id FROM invoice_line il2
                 INNER JOIN invoice i2 ON i2.invoice_id = il2.invoice_id
                 INNER JOIN track t2 ON t2.track_id = il2.track_id
                 WHERE
                         il2.invoice_id <= ifs.invoice_id
                     AND i2.customer_id = ifs.customer_id
                     AND t2.album_id = t.album_id

                ) IS NULL THEN "yes"
            ELSE "no"
        END AS "album_purchase"
    FROM invoice_first_track ifs
    INNER JOIN track t ON ifs.first_track_id = t.track_id
    )
GROUP BY 1
```

- Even the most complicated of SQL queries end up using basic commands like **SELECT, MIN, JOIN, WHERE, GROUP BY.**
- It is not about knowing the most complicated concepts, but focusing on a firm basic.
- Then, it is just a matter of how well you understand the use cases.

SQL is actually a cool Grandpa.

In spite of seeming archaic and old, SQL and its holistic understanding will help you master how data in huge databases are stored, structured, and queried.

Let us start learning !

(... finally.)

An Overview

- SQL dates back almost 50 years to 1970 when Edgar Codd, a computer scientist working for IBM, wrote a paper describing a new system for organizing data in databases.

- By the end of the decade, several prototypes of Codd's system had been built, and a query language — the Structured Query Language (SQL) — was born to interact with these databases.

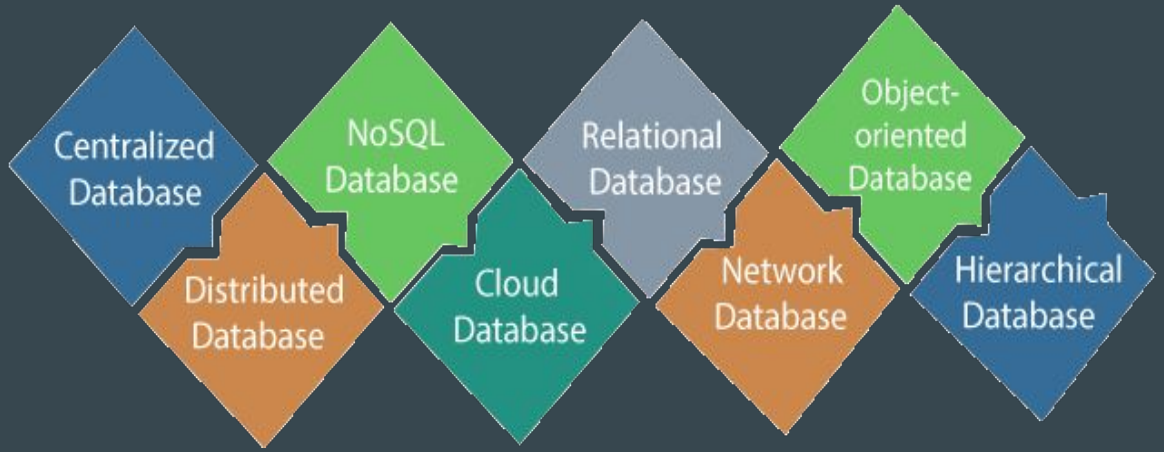
Features of SQL

- Relational Database
  - There exists a relation between tables in the database
  - Unlike Non - Relational Databases, where data is present in a key value pair (like in a json file)
- Well defined standards, no room for ambiguity
- Easy to learn (descriptive queries)
- Multiple Views
  - A virtual/temporary table for use
  - Protects data integrity

Database and Data

- **<u>Database</u> :-** An organized collection, stored and accessed in a certain format.
- Eg :- Imagine a library.

Types of Database



mongoDB

SQL

MySQL

Table (Eg)

Table Name → Person

ID	FIRST_NAME	LAST_NAME	D.O.B
1	OMAIZ	AHMED	13TH MARCH
2	RACHIT	KUMAR	16TH OCTOBER
3	SAMHITHA	BHAT	12TH NOVEMBER
4	RAKSHITH	KAKTHKAR	22ND OCTOBER
5	PURVI	RAVAL	8TH AUGUST
6	VISHNU	RAO	15TH DECEMBER

Tuples

Attributes

Table Constraints

1. **NOT NULL**

If we specify a field in a table to be NOT NULL. Then the field will never accept null value.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);
```

This query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

Table Constraints

2 . UNIQUE

- To uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values
- can have more than one UNIQUE columns in a table.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);
```

This query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID

Table Constraints

3 . PRIMARY KEY

- uniquely identifies each row in the table
- the field will not be able to contain NULL values as well as all the rows should have unique values for this field
- combination of NOT NULL and UNIQUE constraints.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID)
);
```

create a table named Student and specifies the field ID as primary key.

Table Constraints

4 . FOREIGN KEY

- a field in a table which uniquely identifies each row of a another table.
- points to primary key of another table
- creates a kind of link between the tables.

Orders Table

O_ID	ORDER_NO	C_ID
1	2253	3
2	3325	3
3	4521	2
4	8532	1

C_ID in Orders table

is the primary key in Customers table

Customers Table

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

C_ID in Customers Table uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.

Syntax

```
CREATE TABLE Orders
(
O_ID int NOT NULL ,
ORDER_NO int NOT NULL ,
C_ID int ,
PRIMARY KEY (O_ID) ,
FOREIGN KEY (C_ID) REFERENCES
Customers(C_ID)
) ;
```

Table Constraints

5 . CHECK :-

- specify a condition for a field, which should be satisfied at the time of entering values for this field.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);
```

This query creates a table Student and specifies the condition for the field AGE as (AGE >= 18). The user will not be allowed to enter any record in the table with AGE < 18

Table Constraints

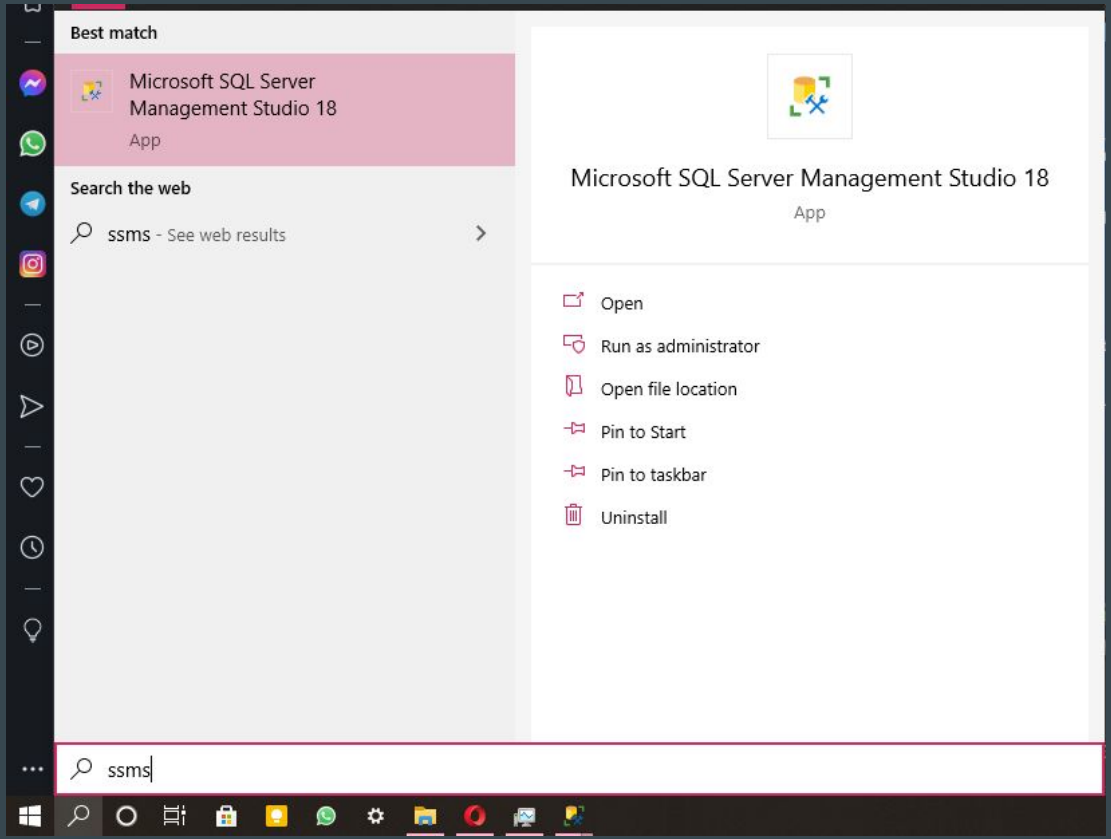
6 . DEFAULT :-

- to provide a default value for the fields. If at the time of entering new records in the table if the user does not specify any value then the default value will be assigned to them.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int DEFAULT 18
);
```

This query will create a table named Student and specify the default value for the field AGE as 18.

Connecting to SQL server using SSMS



Microsoft SQL Server Management Studio

File Edit View Tools Window Help

New Query

Execute

Quick Launch (Ctrl+Q)

Object Explorer

Connect

Connect to Server

SQL Server

Server type: Database Engine

Server name: DESKTOP-6RLPFOO

Authentication: Windows Authentication

User name: DESKTOP-6RLPFOO\tewar

Password:

Remember password

Connect | Cancel | Help | Options >>

Ready

27°C AQI 41 ENG 07:49

Creating and altering Database

- **CREATE** :- Create Database Sample1;
- **ALTER** :-Alter Database Sample1 Modify Name=Sample2;
- **DROP** :- Drop Database Sample2;
- If the db is currently being used by another user, it can't be deleted.

  Set the Db to single user mode, then delete.

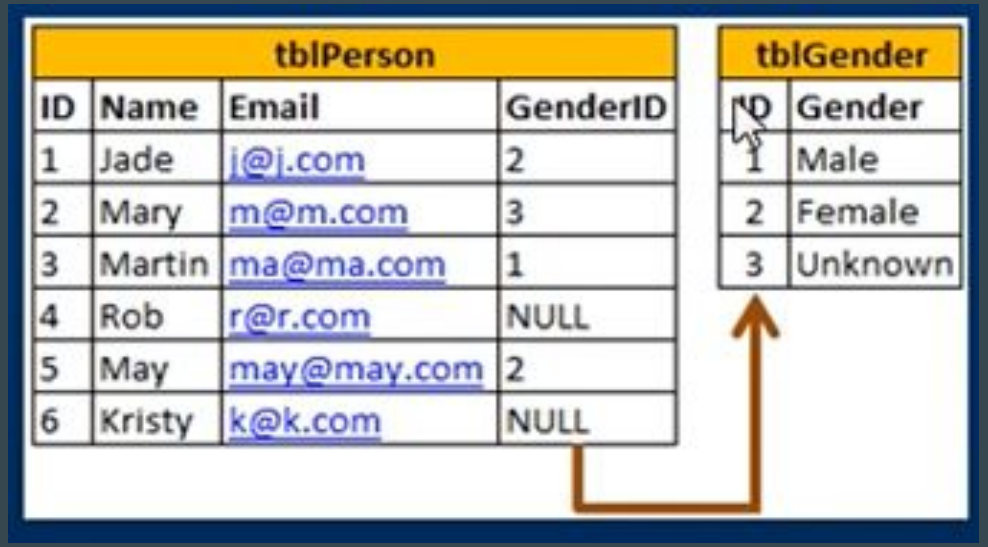
  Alter Database Sample2 Set SINGLE_USER With Rollback Immediate;

Welcome

DAY 2

Creating Table

- Create table **tblPerson** and **tblGender**
- Establish primary key and foreign key constraints



tblPerson

ID	Name	Email	GenderID
1	Jade	j@j.com	2
2	Mary	m@m.com	3
3	Martin	ma@ma.com	1
4	Rob	r@r.com	NULL
5	May	may@may.com	2
6	Kristy	k@k.com	NULL

tblGender

ID	Gender
1	Male
2	Female
3	Unknown

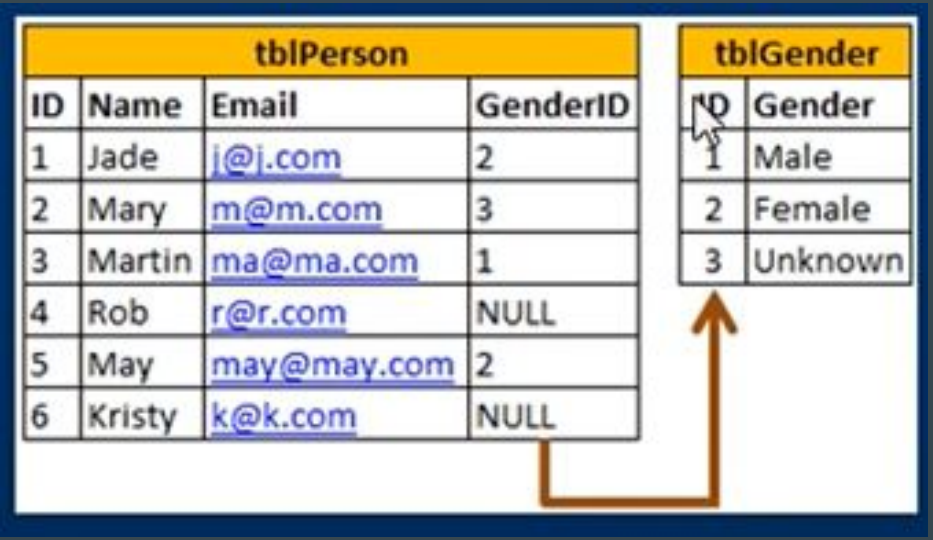
To Create tblPerson

We will use GUI

To Create tblGender

```
Use [Sample1]
Create Table tblGender
(
ID int NOT NULL Primary Key,
Gender nvarchar(50) NOT NULL
)
```

Foreign Key



tblPerson

ID	Name	Email	GenderID
1	Jade	j@j.com	2
2	Mary	m@m.com	3
3	Martin	ma@ma.com	1
4	Rob	r@r.com	NULL
5	May	may@may.com	2
6	Kristy	k@k.com	NULL

tblGender

ID	Gender
1	Male
2	Female
3	Unknown

- **tblPerson** contains the Foreign key **GenderID**
- Map the Column **GenderID** in **tblPerson** as a **Foreign key constraint**
- The foreign key GenderID is looking up for values in the tblGender table
- The Primary key of tblGender , ID is acting as a Foreign key in tblPerson table

- **SYNTAX:-**

  Alter Table ForeignKeyTable add constraint ForeignKeyTable_ForeignKeyColumn_FK

  Foreign Key (ForeignKeyColumn) references PrimaryKeyTable(PrimaryKeyColumn);

EG:-

Alter Table tblPerson add constraint tblPerson_GenderID_FK

Foreign Key (GenderID) references tblGender(ID);

Insert Values

- **For tblPerson :**

  Insert into tblPerson(ID,Name,Email,GenderID)

  Values (3 , ‘Martin’ , ‘ma@ma.com’ , 1);

- **For tblGender:**

  Insert into tblGender(ID, GenderID)

  values(3 , ‘Unknown’);

Results | Messages

	ID	Name	Email	GenderID
1	1	John	j@j.com	3
2	2	Mary	m@m.com	2
3	3	Martin	ma@ma.com	1
4	4	Rob	r@r.com	1
5	5	May	may@may.com	2
6	6	Kristy	k@k.com	3

	ID	GenderID
1	1	Male
2	2	Female
3	3	Unknown

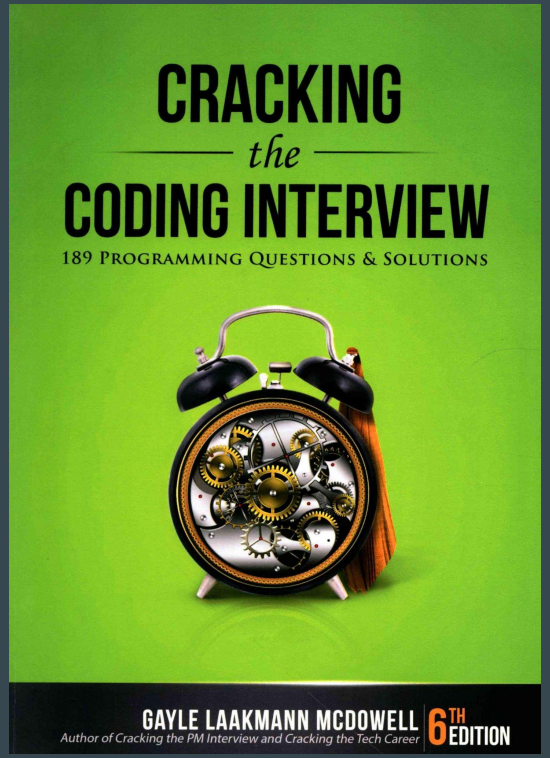
A little break, maybe ?

Too much SQL ?

Let us take a few minutes to look at some of the questions that you guys had in yesterday’s session

Feel free to drop them in the live chat if I have missed your question here

- How much DSA is required for interviews?



Difference between WHERE and HAVING clause

WHERE Clause is used to filter the records from the table based on the specified condition. HAVING Clause is used to filter record from the groups based on the specified condition.

Let us build a different table to understand this

Sales Table

```
Use [Sample2]
Create table Sales
(
    Product nvarchar(50),
    SaleAmount int
)
```

```
Insert into Sales values ('iPhone', 500)
Insert into Sales values ('Laptop', 800)
Insert into Sales values ('iPhone', 1000)
Insert into Sales values ('Speakers', 400)
Insert into Sales values ('Laptop', 600)
```

	Product	SaleAmount
1	iPhone	500
2	Laptop	800
3	iPhone	1000
4	Speakers	400
5	Laptop	600

Now → Where,Having, and Group By

- **WHERE :-**
  - This clause is used to filter records.
  - extract only those records that fulfill a specified condition

```
Select SaleAmount from Sales where Product='Laptop' ;
```

- **GROUP BY :-**
  - The GROUP BY statement groups rows that have the same values into summary rows.

```
Select count(Product) from Sales

group by Product
```

- **HAVING :-**

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

```
Select Product from Sales

Having Sum(SaleAmount)<=1500;
```

What does this ERROR mean???

```
Select Product from Sales

group by Product

Having Sum(SaleAmount)<=1500;
```

Try using WHERE instead of HAVING ?

```
Select Product from Sales

Where Sum(SaleAmount)<=1500

group by Product
```

Difference between WHERE and HAVING clause

- WHERE comes before GROUP BY. This means WHERE clause filters rows before aggregate calculations are performed. HAVING comes after GROUP BY.
- This means HAVING clause filters rows after aggregate calculations are performed. So from a performance standpoint, HAVING is slower than WHERE and should be avoided when possible.

Look at the previous queries...

Another example to clear your confusion ?

GROUP BY

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
WHERE Product in ('iPhone', 'Speakers')
GROUP BY Product
```

HAVING

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
HAVING Product in ('iPhone', 'Speakers')
```

NOTICE BOTH OUTPUTS

Back to the boring stuff.....

Update a Column Value

- EG:-

```
Update tblPerson

Set GenderID = 1

Where ID=1 ;
```

```
Update tblPerson
Set Name = 'Kenny'
Where ID = 6;
```

Delete a Row

- **SYNTAX :-**

DELETE FROM table_name WHERE condition;

Eg:-

DELETE FROM tblPerson WHERE ID=6;

	ID	Name	Email	GenderID
1	1	John	j@j.com	3
2	2	Mary	m@m.com	2
3	3	Martin	ma@ma.com	1
4	4	Rob	r@r.com	1
5	5	May	may@may.com	2

Default Constraint

- Insert values in tblPerson in such a way that if GenderID is not entered, it is taken as 3 by default and not null.
- Let us try to insert a row without entering the GenderID

```
Insert into tblPerson(ID,Name,Email)

Values(6,'Kenny','k@k.com');
```

Results | Messages

	ID	Name	Email	GenderID
1	1	John	j@j.com	3
2	2	Mary	m@m.com	2
3	3	Martin	ma@ma.com	1
4	4	Rob	r@r.com	1
5	5	May	may@may.com	2
6	6	Kenny	k@k.com	NULL

	ID	GenderID
1	1	Male
2	2	Female
3	3	Unknown

- Now add default constraint :

```
Alter table tblPerson

ADD Constraint DF_tblPerson_GenderID

Default 3 for GenderID;
```

- Now, insert rows without giving GenderID

```
Insert into tblPerson(ID,Name,Email)

Values(7,'Rick','r@r.com');

Insert into tblPerson(ID,Name,Email)

Values(8,'Mike','mike@r.com');
```

	ID	Name	Email	GenderID
1	1	John	j@j.com	3
2	2	Mary	m@m.com	2
3	3	Martin	ma@ma.com	1
4	4	Rob	r@r.com	1
5	5	May	may@may.com	2
6	6	Kenny	k@k.com	NULL
7	7	Rick	r@r.com	3
8	8	Mike	mike@r.com	3

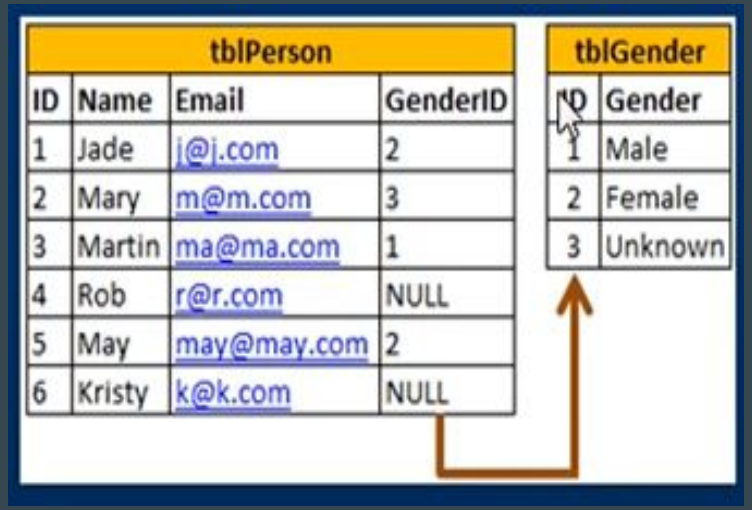
	ID	GenderID
1	1	Male
2	2	Female
3	3	Unknown

Drop Constraint

```
Alter table tblPerson

Drop constraint DF_tblPerson_GenderID;
```

Referential integrity constraint



tblPerson

ID	Name	Email	GenderID
1	Jade	j@j.com	2
2	Mary	m@m.com	3
3	Martin	ma@ma.com	1
4	Rob	r@r.com	NULL
5	May	may@may.com	2
6	Kristy	k@k.com	NULL

tblGender

ID	Gender
1	Male
2	Female
3	Unknown

- What if a user deletes the 2nd row in tblGender ???

  Delete from tblGender
  where ID=2

- Orphan records

This constraint allows us to define the actions that the Microsoft SQL server should take when user attempts to delete or update a key to which a foreign key points.

4 Options

1. **No Action:** This is the default behaviour. No Action specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE or UPDATE is rolled back.

2. **Set Default:** Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to default values.

3. **Set NULL:** Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to NULL.

4. **Cascade:** Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted or updated.

Onto Hackerrank...

Questions