Introduction to Databases

What is a database?

- A structured collection of data organized and stored in a computer system.
- Composed of tables with rows and columns representing records and attributes.
- Efficiently manages and retrieves information for various purposes.
- Supports complex relationships and data integrity constraints.
- Provides a query language (e.g., SQL) to interact with and analyze data.

Who are the users of a database?

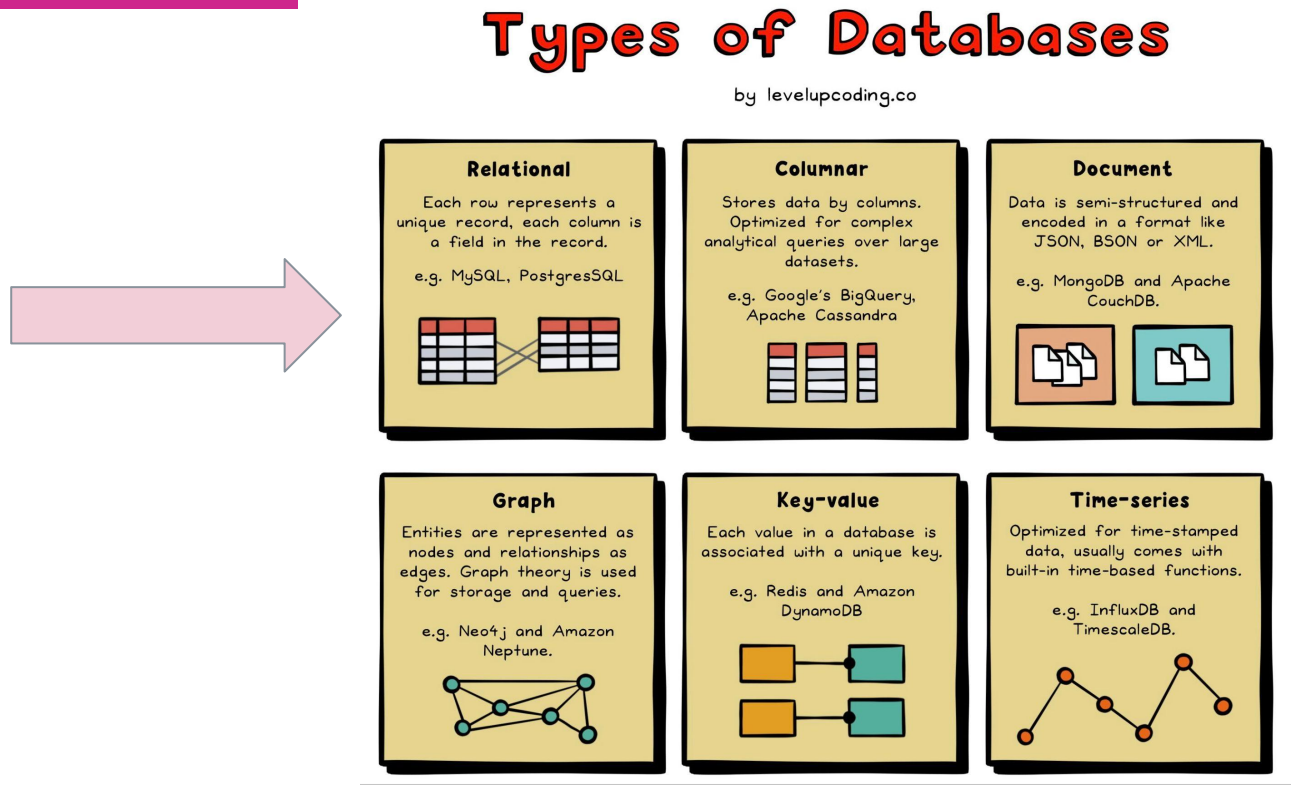
- Analysts
  - Marketing
  - Sales
  - Business
- Technical
  - Data Scientists & Engineers
  - Software Developers
  - Web Developers

In short, anyone who has to deal with data!

Why not just use google sheet / excel?

- Limited data handling capabilities for large volumes of data.
- Insufficient data integrity constraints and security features.
- Challenges with simultaneous collaboration on a larger scale.
- Lack of scalability and optimized performance for growing datasets.
- Limited capabilities for complex querying and advanced analysis.

Different databases!



Popular relational databases!

- **MySQL:**
  - Wide Adoption: MySQL has a large user community and is widely adopted for applications of all sizes.
- **PostgreSQL:**
  - Advanced Features: PostgreSQL provides advanced features like complex queries, full-text search, and support for custom data types.
- **Oracle Database:**
  - Robustness and Reliability: Oracle Database is known for its reliability, durability, and fault-tolerance.
- **Microsoft SQL Server:**
  - Integration with Microsoft Ecosystem: SQL Server seamlessly integrates with other Microsoft products and technologies.

How data is stored in a relational database?

Columns

Rows

food_delivery_dataset

Order_ID	Customer_Name	Restaurant_Name	Delivery_Address	Food_Item	Quantity	Price	Order_Date	Delivery_Date	Payment_Method
1001	Customer 1	McDonald's	1 Main St, City A	Big Mac	3	10.19	2023-05-01	2023-05-01	Cash
1002	Customer 2	McDonald's	2 Main St, City B	Fried Chicken	1	5.64	2023-05-02	2023-05-02	Cash
1003	Customer 3	Burger King	3 Main St, City C	Pepperoni Pizza	1	10.37	2023-05-03	2023-05-03	Cash
1004	Customer 4	Burger King	4 Main St, City D	Turkey Sub	3	24.32	2023-05-04	2023-05-04	Cash
1005	Customer 5	McDonald's	5 Main St, City E	Cheeseburger	1	19.93	2023-05-05	2023-05-05	Cash
1006	Customer 6	KFC	6 Main St, City A	Cheeseburger	3	8.88	2023-05-06	2023-05-06	Credit Card
1007	Customer 7	Pizza Hut	7 Main St, City B	Big Mac	1	5.98	2023-05-07	2023-05-07	Cash
1008	Customer 8	McDonald's	8 Main St, City C	Fried Chicken	2	15.08	2023-05-08	2023-05-08	Cash

Let’s continue to SQL!

Intro to SQL & Environment Setup

What is SQL?

- SQL, or **Structured Query Language,** is a powerful programming language designed for managing, querying, and manipulating relational databases.
- Common Use Cases:
  - Data Retrieval
  - Data Manipulation
  - Database Creation
  - Data Maintenance
  - Data Analysis
  - Reporting

Installation Procedure

- Install Mysql Database: https://dev.mysql.com/downloads/mysql/
- Install Workbench: https://dev.mysql.com/downloads/workbench/

Creating Database & Tables

Focus for this section

Schema

Data Types

Primary & Foreign Key

Constraints

DDL & DML

Database Fundamentals

What is a schema?

A schema is a logical representation of the database structure, defining how the data will be organized, stored, and accessed. It acts as a roadmap for the database, outlining the tables, fields, relationships, and constraints.

- A database can have multiple schemas.
- Multiple schema in one database can connect between each other given permission is granted

Data Types

Different Data types details

- **Boolean**
  - True, False
- **Character**
  - Char ⟶ Fixed-length strings with a known length, like country codes or phone number.
  - Varchar ⟶ Variable-length strings with varying lengths, like names. Saves storage space for shorter entries.
  - Text ⟶ Large textual data like articles, descriptions, or comments, without a predefined maximum length.

Data Types (Continued)

- Numeric
  - int ⟶ Whole numbers without decimal places, like unique IDs.
  - float ⟶ Approximate real numbers with floating-point precision, like temperature.
  - double ⟶ Stored using more bytes than float, providing higher precision for critical calculations. Eg. 3.1416
- Temporal
  - date / datetime ⟶ Store a specific date and time in a fixed format.
  - timestamp ⟶ Logging transaction timestamps generated by the system.

Primary vs Foreign Key

- A **primary key** is a column or group of columns used to identify a row uniquely in a table
  - One table can have a singular primary key or a multi column primary key
- A **foreign key** is a field or group of fields used in a table to identify a unique row in another table
  - Generally defined in a table that points to another table which has the primary key
  - Table containing the foreign key is called the child table, and the table that it points to is called the parent table
  - One table can have multiple foreign keys

Constraints

- Constraints are the rules enforced on data columns or tables
- Used as a proactive measure to prevent invalid data
- Ensures the accuracy & reliability of data in a database
- Have two categories :
  - **Column constraints**
    - Constraints the data in a column to follow some rules
    - Mostly used for simple table definitions
  - **Table Constraints**
    - Applied to entire table instead of a single column
    - appropriate for more complex rules that involve interactions between multiple columns

Column Constraints

- Most common Column constraints:
  - **UNIQUE**
    - Ensures that all values in a column is different
  - **NOT NULL**
    - Ensures the column can not have a null value
  - **Primary key & Foreign Key**
    - Uniquely identifies a row in a database
  - **CHECK**
    - Ensures that all values in a column satisfies certain criteria

Table Constraints

- Most common Column constraints:
  - **UNIQUE (column list)**
    - Values of the column listed here should be unique
  - **NOT NULL (column list)**
  - **Primary key (column list)**
    - Allows to define primary key that consists multiple columns
  - **CHECK (condition)**
    - Check a condition when inserting or updating data

DDL & DML

DDL - Data Definition Language

- Defines and manages the structure and schema of the database by allowing analysts to create, modify, and delete database objects such as tables, indexes, views, and constraints.
- DDL statements enable analysts to define user access, permissions, and other security measures to ensure data integrity and protect sensitive information.
- Executes static operations that bring permanent changes to the database schema. DDL statements, such as CREATE, ALTER, and DROP, are executed less frequently compared to DML statements.

Most common DDL Statements

- **CREATE**: Used to create new database objects
- **ALTER**: Used to modify existing database objects.
- **DROP**: Used to delete database objects.
- **TRUNCATE**: Used to remove all data from a table, but not the table structure itself. It is a more efficient way to delete all records from a table compared to the DELETE statement.
- **RENAME**: Used to change the name of an existing database object.

DML - Data Manipulation Language

- Manipulates data within the database through insertion, update, retrieval, and deletion. DML statements like INSERT, UPDATE, DELETE, and SELECT to add new records, modify existing data, retrieve specific information, or remove data from database tables.
- Unlike DDL statements, which primarily focus on the database structure, DML statements directly affect the data & mostly used for day to day operations.

Most common DML Statements

- **SELECT**: Used to retrieve data from one or more tables in the database.
- **INSERT**: Used to add new records or rows into a table.
- **UPDATE**: Used to modify existing data in a table.
- **DELETE**: Used to remove records or rows from a table.

Creating our first Database table!

Hands on DDL & DML

DDL

- Create Table
- Alter table
- Drop Table
- Truncate Table

DML

- Select
- Insert
- Update
- Delete

Database Normalization

What is Normalization

- Normalization is a technique to arrange data in tables to minimise data redundancy.
- What is redundancy? Repetition of similar data at multiple places.
- What are the issues of data redundancy?
  - Unnecessary space consumption
  - Insertion anomaly
  - Update anomaly
  - Deletion anomaly

Why does it help?

Student_id	Name	Subject	HOD	Phone
1	Andalib	Math	Mr. X	123
2	Nabila	Math	Mr. X	123
3	Sakib	Business	Ms. Y	233
4	Rahman	Business	Ms. Y	233

Student_id	Name	Subject
1	Andalib	Math
2	Nabila	Math
3	Sakib	Business
4	Rahman	Business

Subject	HOD	Phone
Math	Mr. X	123
Business	Ms. Y	233

Common types of Normalization

1NF

2NF

3NF

1st Normalization Form (1NF)

- Each column should have a single data type
- Each column should have a unique & clear name
- Each row (record) in a table should be unique.
- Each column should hold a single value for each record
- Must have a primary key

1NF Example

Student

Student ID	Name	Subjects
1	Andalib	SQL, C++
2	Nabila	Java
3	Sakib	C, C++

Student

Student ID	Name	Subjects
1	Andalib	SQL
2	Nabila	Java
3	Sakib	C
1	Andalib	C++
3	Sakib	C++

1NF Example

Student

Student ID	Name
1	Andalib
2	Nabila
3	Sakib

Subject

Subject_ID	Subject_name
1	SQL
2	Java
3	C
4	C++

student_subject

id	student_id	subject_id
1	1	1
2	2	2
3	3	3
4	1	4
5	3	4

2nd Normalization Form (2NF)

- Table must be in 1NF
- Table should not have any partial dependency

What is partial dependency?

Student

Student_ID (PK)	Name	Reg_No	Branch
1	Andalib	CSE-08	Dhaka
2	Andalib	BBA-09	Dhaka
3	Nabila	BBA-03	Chittagong
4	Sakib	EEE-09	Khulna

2NF Example

Partial Dependency

Subject

Subject_ID	Subject_name
1	Math
2	Science
3	Business
4	Stats

Subject

Subject_ID	Subject_name	teacher
1	Math	Mr. Math
2	Science	Ms. Science
3	Business	Ms. Business
4	Stats	Mr. Stats

Result

Score_id	Student_id	Subject_Id	mark
1	1	1	80
2	1	2	70
3	2	4	86
4	3	3	90

3rd Normalization Form (3NF)

- Must be in 2nd normal form.
- Should not have Transitive dependency

Transitive Dependency: Transitive dependency is a concept where there is an indirect relationship between attributes (columns) within a table. It occurs within a table with a composite primary key, where an attribute depends on another attribute through a third attribute.

3NF Example

Result

Score_id	Student_id	Subject_Id	mark	Exam Name	Total Marks
1	1	1	80	First_Sem	170
2	1	2	70	First_Sem	170
3	2	4	86	Second_Sem	160
4	3	3	90	Second_Sem	145

3NF Example

Exam_Table

Exam_ID	Exam_name
1	First_Sem
2	Second_Sem
3	Third_Sem
4	Fourth_sem

Total_mark

Exam_id	Student_id	Total_mark
1	1	170
2	2	160
2	3	145

Result

Score_id	Student_id	Subject_Id	mark	Exam_id
1	1	1	80	1
2	1	2	70	1
3	2	4	86	2
4	3	3	90	2

CONGRATS!

Data Retrieval

SELECT

- Most common type of SQL Statement
- Works on majority of the database systems

SELECT column_name FROM table_name

Pro tip: Always use CAPITAL letters on your keywords while writing SQL

SELECT

- It is advisable to avoid writing * on select statements if you don't need all the columns
- It will automatically query everything and increase traffic between database server & application and slow down data retrieval process

DISTINCT Statement

DISTINCT

- A column in a database sometimes can have duplicate values and you might want to know what are the unique/distinct values
- DISTINCT keyword enables you to do just that

```
SELECT DISTINCT column_name FROM table_name
```

Example of Distinct

item_id	restaurant_id	name	description	price	availability	has_photo
49	2	Biriyani	Description for Item 49	7	1	1
43	4	Biriyani	Description for Item 43	11	1	0
6	6	Biriyani	Description for Item 6	16	0	0
9	6	Biryiani	Description for Item 9	1	0	0
1	4	Burger	Description for Item 1	1	1	1
48	1	Burger	Description for Item 48	22	1	0
31	5	Burger	Description for Item 31	8	1	0
27	2	Burger	Description for Item 27	12	0	1
5	7	Cheesy Chicken Burger	Description for Item 5	14	1	0
35	5	Cheesy Chicken Burger	Description for Item 35	10	0	0

name
Biriyani
Burger
Cheesy Chicken Burger

```
SELECT DISTINCT name FROM menuitems
```

Filtering Data with WHERE Statement

WHERE Statement

- This statement helps us specify conditions on rows to be returned
- Basic Syntax:
  - SELECT column1,column2 FROM table1 WHERE (conditions)
- Used to filter rows
- Works on the following
  - Comparison operators
  - Logical operators
  - Special operators

Comparison Operators

Operator	Description
=	Equal
>	Greater than
<	Less Than
>=	Greater than or equal to
<=	Less than or equal to
<> or !=	Not equal to

Logical Operators

Allows us to combine multiple comparison operators

- AND
- OR
- NOT

Special Operators

Enhances WHERE statement

- IN
- BETWEEN
- LIKE

Between

Between operator is used to match a value against a range of values.

- value between High and Low
- Essentially same as
  - value>=low and value<=high

NOT BETWEEN operator is the same as:

- Value NOT BETWEEN low AND high
- value<low or value>high

LIKE

- Used for pattern matching.
  - Find all emails ending with @gmail.com
  - Find all names starting with “A”
- **LIKE** operator allows us pattern matching against a string with wildcard characters.
  - % to match any sequence of characters
  - _ to match a single character

LIKE

- All names that begin with A
  - “A%”
- All names that end with a
  - “%a”
- **LIKE** is case sensitive

Sorting and Ordering Results

ORDER BY

- This statement is used for arranging or sorting rows based on a column value in ascending or descending order

  Basic Syntax:
  SELECT column1, column2 FROM table1
  ORDER BY column1 ASC/DESC

ORDER BY

- **ASC** ⟶ Retrieve value in Ascending order
- **DESC** ⟶ Retrieve value in Descending order
- If we leave it blank then by default system sets it as ASC
- Can be used on one column or multiple columns

Show hands on example. Menuitems table order by itemname, restaurant_id

LIMIT Statement

LIMIT

- Allows us to limit # of rows returned by the query
- Very useful while we are trying to explore data
- Sits at the very end of the query

Example:

```
SELECT column1, colum2 FROM table1 LIMIT 10;
```

Data Manipulation & Joins

INSERT

- The **INSERT** statement in SQL is used to insert new data into a table.
- It allows us to add records to a database table, expanding the dataset.

Example:

```
INSERT INTO table1 (column 2, column2)
VALUES (1,"Name")
```

UPDATE

- The **UPDATE** statement in SQL is used to modify existing data in a table.
- The **UPDATE** statement uses the **SET** keyword to specify the columns to be updated and their new values.
- The **WHERE** clause is used to specify the condition for selecting the rows to be updated. Without it, all rows in the table would be affected.
- It is crucial to use the **WHERE** clause carefully to ensure only the intended records are updated.

UPDATE

- Example:

```
UPDATE table1
SET column1 = “new value”
WHERE column1 = “Old value”
```

DELETE

- The **DELETE** statement in SQL is used to remove one or more records from a table.
- The **DELETE** statement affects the entire row (record) specified by the WHERE clause condition.
- Without the WHERE clause, the **DELETE** statement would remove all rows from the table, leading to data loss.
- The **DELETE** statement is a row-by-row operation.

Example:

DELETE FROM table1 WHERE column1 = “Some value”

Introduction to JOINS

What will we learn?

- Joins allow us to combine information from multiple tables
- We will learn how to create alias with AS clause
- Different type of joins:
  - INNER JOIN
  - OUTER JOIN
  - FULL JOIN
  - UNIONS

AS clause

- We can use AS to create aliases in two fronts
  - Column Alias
  - Table Alias
- The AS operator is executed at the very end of the query meaning we can not use it within a where clause

Column Example: SELECT column1 **AS** first_column FROM table1

Table Example: SELECT column1 FROM table1 **AS** A

INNER JOIN

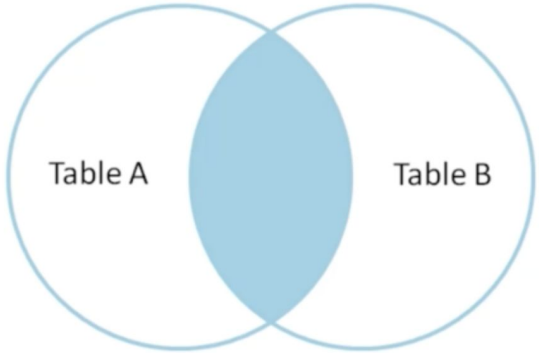
- An **INNER JOIN** is an SQL operation that combines rows from two or more tables based on a specified condition.
- It retrieves only the rows with matching values in both tables, excluding non-matching rows.

INNER JOIN

SYNTAX:

SELECT * FROM tableA **INNER JOIN** tableB

ON table1.column1 = table2.column1



OUTER JOIN

- There are few type of outer joins
  - FULL OUTER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
- An OUTER JOIN is an SQL operation that combines rows from two or more tables based on a specified condition.
- It retrieves matching rows as well as non-matching rows

  from one or both tables, based on the join condition.

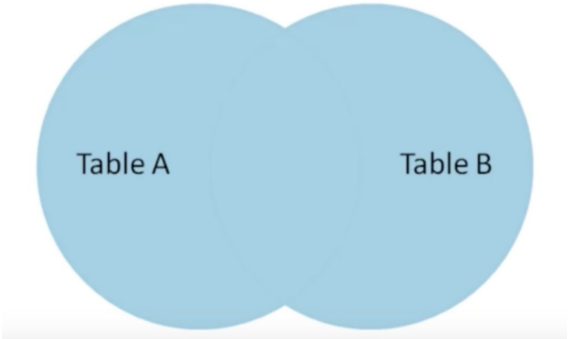
FULL OUTER JOIN

- **A FULL OUTER JOIN** is an SQL operation that combines rows from two or more tables based on a specified condition.
- It retrieves all rows from both tables, including matching rows, as well as non-matching rows from either table.

FULL OUTER JOIN

SELECT * FROM Table A

FULL OUTER JOIN TableB
ON TableA.column1 = TableB.column1



LEFT OUTER JOIN aka LEFT JOIN

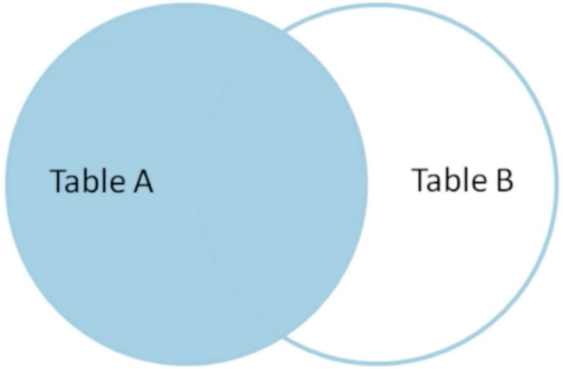
- It retrieves all rows from the left table and matching rows from the right table, including non-matching rows from the left table.
- It allows us to compare data from two tables and identify rows that do not have corresponding matches in the right table.

LEFT OUTER JOIN aka LEFT JOIN

SYNTAX:

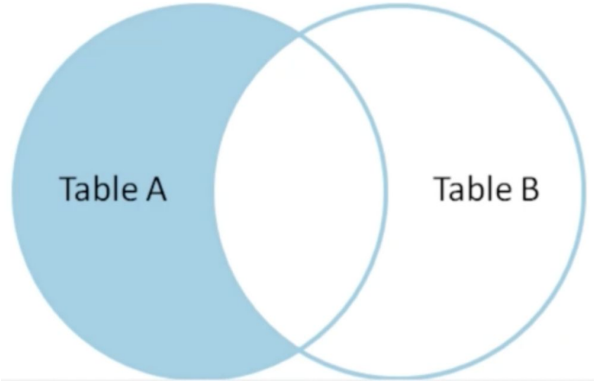
SELECT * FROM tableA **LEFT OUTER JOIN** tableB

ON table1.column1 = table2.column1



LEFT OUTER JOIN

- What if we only wanted rows unique to Table A? Rows that are found in Table A and not in Table B.



SYNTAX:

SELECT * FROM tableA **LEFT JOIN** tableB

ON table1.column1 = table2.column1

WHERE table2.column1 is null

RIGHT OUTER JOIN aka RIGHT JOIN

- It retrieves all rows from the right table and matching rows from the left table, including non-matching rows from the right table.
- RIGHT OUTER JOIN is less commonly used than INNER JOIN or LEFT OUTER JOIN, but it can be invaluable when we want to include all data from the right table, regardless of whether matching data exists in the left table.

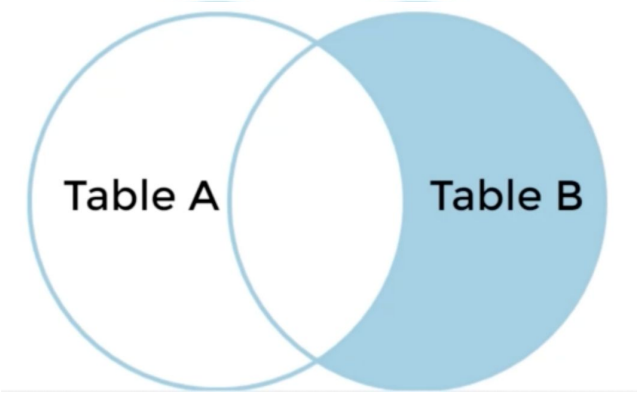
RIGHT OUTER JOIN aka RIGHT JOIN

SYNTAX:

SELECT * FROM tableA **RIGHT OUTER JOIN** tableB

ON tableA.column1 = tableB.column1

WHERE tableA.id is null



UNION

- The **UNION** operator in SQL is used to combine the result sets of two or more **SELECT** queries into a single result set.
- It removes duplicate rows from the final result, producing a distinct set of records.
- On the other hand, **UNION ALL** does not remove duplicates but combines the result of two or more **SELECT** queries.

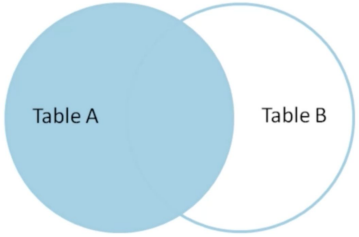
EXAMPLE:

```
SELECT column1, column2 FROM table1

UNION

SELECT column1, column2 FROM table2
```

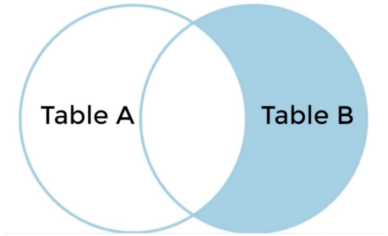
FULL OUTER JOIN in MYSQL

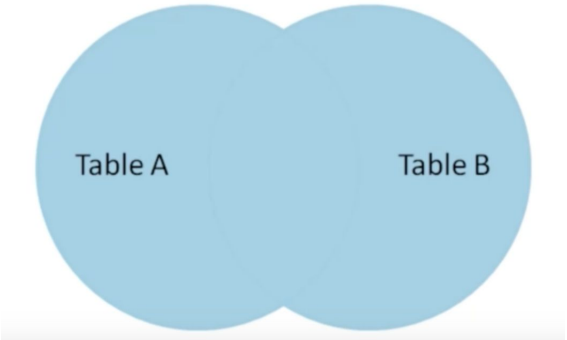
```
SELECT * FROM table1
LEFT JOIN table2 ON table1.column = table2.column
UNION
SELECT * FROM table1
RIGHT JOIN table2 ON table1.column = table2.column
WHERE table1.column IS NULL;
```

FULL OUTER JOIN in MYSQL



UNION





You already know the basics!

SQL Aggregations & Functions

Aggregate Functions

In MySQL, aggregate functions are used to perform calculations on a set of rows and return a single value as the result. These functions summarize data in various ways, such as computing the sum, average, maximum, minimum, count, etc.

- COUNT ⟶ Returns number of values
- SUM ⟶ Returns the sum of all values
- AVG ⟶ Returns average value
- MAX ⟶ Returns the maximum value
- MIN ⟶ Returns the minimum value

Good to know!

- **AVG()** returns values with many decimal places. Eg. 3.141628
  - We can use **ROUND()** function to specify precision after the decimal
- COUNT() simply returns the number of rows.
  - So, we just write **COUNT(*)**

GROUP BY

GROUP BY

GROUP BY allows us to aggregate data & apply functions to better understand how data is distributed on a category level.

GROUP BY

Category	Value
A	5
A	10
B	3
C	4
D	6
D	10
C	8

- We need to select a categorical column to run **GROUP BY**
- Categorical columns are not continuous values
- Keep in mind that, categorical value still can be numerical. Eg. AGE

GROUP BY

Category	Value
A	5
A	10
B	3
C	4
D	6
D	10
C	8

A	5
A	10

B	3

C	4
C	8

D	6
D	10

SUM

A	15
B	3
C	12
D	16

Syntax

```
SELECT column1, AGG(column2) as column_name
FROM table1
GROUP BY column1
```

- GROUP BY must appear after FROM or WHERE statement

Syntax using WHERE

```
SELECT company, division, SUM(sales)
FROM table1
WHERE company in ('A','B')
GROUP BY company, division
```

- WHERE statement should not refer to the aggregated value. We will use **HAVING** statement to handle that part later on

Syntax using ORDER BY

SELECT company, division, SUM(sales)

FROM table1

GROUP BY company, division

ORDER by SUM(SALES)

- For sorting results based on aggregation, make sure to refer to the exact /whole function

HAVING Clause

HAVING

HAVING clause allows us to filter data after the aggregations has taken place.

Example

```
SELECT company, division, SUM(sales)
FROM table1
WHERE company<>'facebook'
GROUP BY company, division
```

- Let's say we want to filter based on SUM(sales), what shall we do?
- We can't use WHERE before the aggregation has taken place

Example

SELECT company, division, **SUM(sales)**

FROM table1

WHERE company<>'facebook'

GROUP BY company, division

HAVING **SUM(sales)**>1000

- HAVING allows us to act as an aggregation filter along with GROUP BY

Advanced SQL topics!

Subquery

What is it?

- Subquery allows us to create complex queries. It's basically writing a query on top of the result of another query.
- Syntax is straightforward and requires two SELECT statements

Good to know!

- Subqueries are run first because they remain inside the parenthesis
- We can also use IN operator along with subquery to check against multiple results returned

CASE conditions!

What is it?

- CASE condition is very similar to IF/ELSE statement in other programing languages
- It's often used to create custom columns or control the flow of data retrieval in queries.

Syntax

```
SELECT column1,
    CASE
        WHEN condition1 THEN 'Result1'
        WHEN condition2 THEN 'Result2'
        ELSE 'Other'
    END
FROM table_name;
```

Example

```
SELECT * FROM TEST;

SELECT column1,
        CASE
                WHEN column1=1 THEN “One”
                WHEN column1 = 2 THEN “Two”
                ELSE “no value”
        END as value_column
FROM TEST;
```

column1
1
2

column1	value_column
1	One
2	Two

COALESCE

What is it?

- The COALESCE function is used to return the first non-NULL value from a list of expressions. It's particularly useful when dealing with NULL values in SQL queries.
- The COALESCE function allows you to replace NULL values with suitable alternatives.
- If all expressions evaluate to NULL, COALESCE returns NULL.

```
SELECT COALESCE(expression1, expression2, expression3, ...)
FROM table_name;
```

Syntax

- Table of Products

ITEM	PRICE	DISCOUNT
A	100	20
B	300	null
C	200	10

Syntax

SELECT product.*, (price-discount) as final_price

FROM product;

ITEM	PRICE	DISCOUNT	Final_Price
A	100	20	80
B	300	null	null
C	200	10	190

Syntax

SELECT product.*, (price-COALESCE(discount,0)) as final_price

FROM product;

ITEM	PRICE	DISCOUNT	Final_Price
A	100	20	80
B	300	null	300
C	200	10	190

STRING Operations

What is it?

- String functions in SQL are powerful tools that manipulate and transform text data within queries
- Different string functions:
  - CONCAT() : Concatenates two or more strings together.
  - UPPER() : Converts a string to uppercase.
  - LOWER() : Converts a string to lowercase.
  - LENGTH() : Returns the length (number of characters) of a string.

Different string functions

- Continued...
  - SUBSTRING() : Extracts a substring from a string based on specified positions.
  - TRIM() : Removes leading and trailing spaces from a string.
  - REPLACE() : Replaces occurrences of a substring with another substring.
  - LEFT() : Returns a specified number of characters from the beginning of a string.
  - RIGHT() : Returns a specified number of characters from the end of a string.

Indentation

What is it?

- Indentation involves adding consistent spacing at the beginning of SQL statements to improve code readability and structure.

Why is it important?

- **Clarity**: Indented SQL queries are easier to read and understand, especially for complex queries or subqueries.
- **Organization**: Indentation helps distinguish SELECT clauses, JOINs, WHERE conditions, and other segments of a query.
- **Debugging**: Clear indentation assists in quickly identifying errors and logic issues in SQL code.
- **Collaboration**: Indented code fosters collaboration as team members can grasp the query's intent more readily.

Example

```
SELECT FIRST_ORDER_DATE, COUNT(USER_ID) NEW_USER
FROM (SELECT USER_ID, MIN(DATE(ORDER_DATE)) FIRST_ORDER_DATE FROM ORDERS GROUP BY USER_ID) AS FIRST_USER_DATE
GROUP BY FIRST_ORDER_DATE
ORDER BY NEW_USER DESC;
```

```
SELECT FIRST_ORDER_DATE, COUNT(USER_ID) NEW_USER
FROM
(
    SELECT USER_ID, MIN(DATE(ORDER_DATE)) FIRST_ORDER_DATE
     FROM ORDERS
    GROUP BY USER_ID
) AS FIRST_USER_DATE
GROUP BY FIRST_ORDER_DATE
ORDER BY NEW_USER DESC;
```

What is analytics?

What is it?

- Every company is generating a lot of data & we are surrounded by it. But data in raw form does not mean anything.
- Data analytics is the systematic process of examining, cleaning, transforming, and interpreting data to discover meaningful insights, patterns, and trends.
- Data analytics is like a detective for data. It takes raw data, often messy and unstructured, and transforms it into valuable information that can drive decisions.

What does an analyst do?

- Examine
- Clean
- Transform
- Interpret
- Discovering Insights

Application of data analytics

- Marketing
- Sales
- Finance
- Fraud
- Operations optimisation
- HR analytics

Give example from a digital marketer's perspective on pizza sales.

Different type of analytics

3 types of analytics

- Descriptive
  - Answers, what happened?
  - Summarizes historical data
- Diagnostic
  - Why is it happening? (example - sales dropped because of increased price)
- Predictive
  - What will happen in the future? (example ⟶ football XG, churn prediction)
- Prescriptive
  - Actionable recommendations (example ⟶ recommendation engine. Amazon, netflix, spotify)

Myths about data analysis!

5 myths about analytics

- It's all about numbers & statistics!
  - It's more about interpreting different data and identify trend.
- You have to be a coding master to be an analyst
  - Not necessarily. SQL is important but Google sheet, Excel, tablea, powerbi are built to make you shine without coding.
- Machine learning way or the highway.
- Only big companies require data analysis
- I'm not a computer science graduate so I can't be an analyst.

Types of Data

Different types of data

- Quantitative Data ⟶ Involves numbers and measurable quantities.
  - Discrete ⟶ Consists of distinct, separate values.
    - # of items sold per day, # of active users
  - Continuous ⟶ Represents measurements that can take any value within a given range.
    - Amount in revenue, distance covered
- Qualitative Data ⟶ Involves non-numeric information or categories.
  - Nominal ⟶ Represents categories with no inherent order. Eg. RED, Green, BLUE, Married/ unmarried
  - Ordinal ⟶ Categories with a specific order or ranking. Eg, Poor, avg, Excellent

Types of Data

- **Structured Data:**
  - Structured data is highly organized and follows a specific format.
  - It is commonly found in relational databases and spreadsheets.
  - Examples include customer information in a CRM database or financial records in an Excel sheet.
- **Semi-Structured Data:**
  - Semi-structured data is less organized than structured data but has some level of structure.
  - It often includes data with flexible schemas, such as JSON or XML files.
  - Semi-structured data is prevalent in web applications and data exchange formats.
- **Unstructured Data:**
  - Unstructured data is chaotic and lacks a specific format or structure.
  - Examples include text documents, social media posts, images, videos, and audio recordings.
  - Natural language processing and machine learning techniques are used to extract insights

Data Collection

What is data collection?

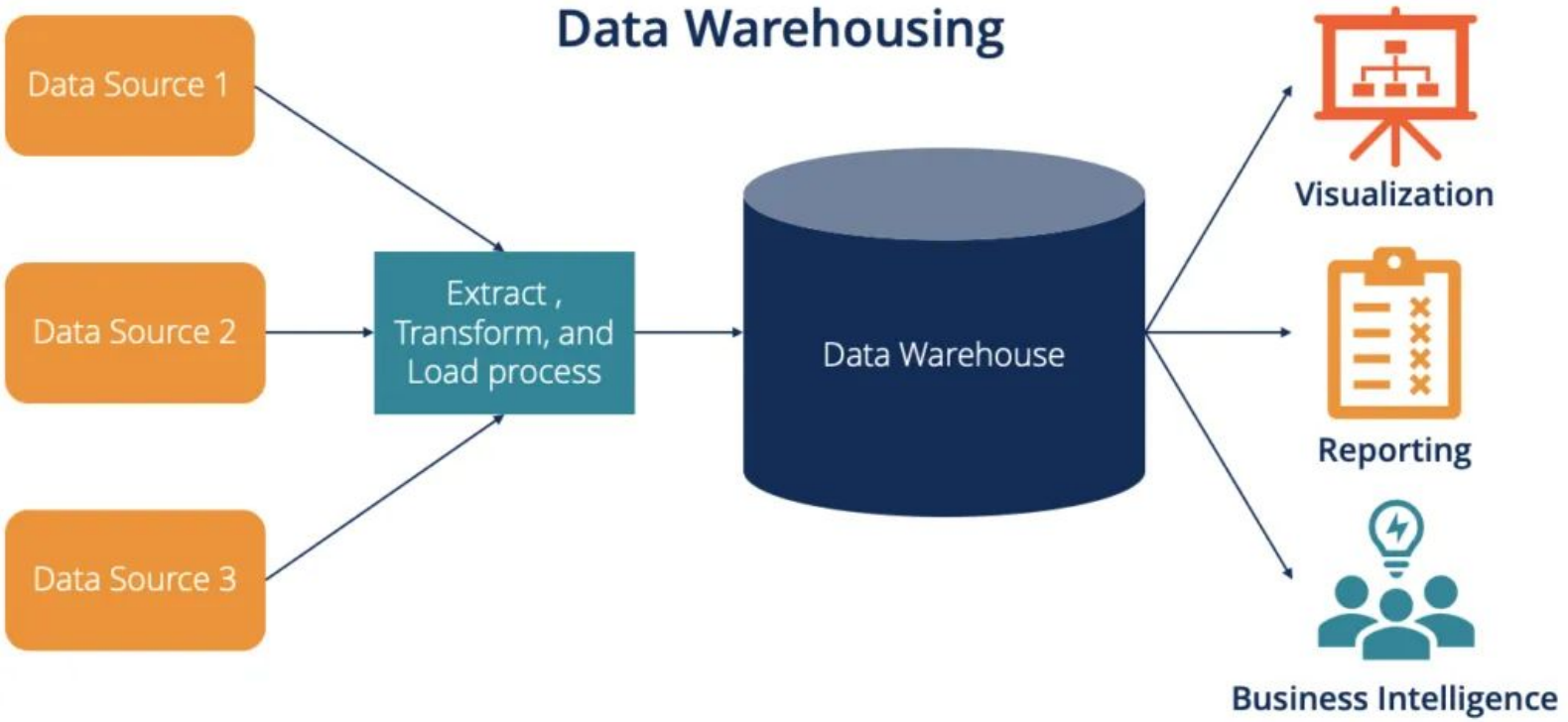
- Data collection is the process of gathering and storing data for analysis.Data collection involves systematically gathering information or data from various sources to answer specific questions, make informed decisions, or analyze trends.
- Data can be collected using various methods, including:
  - Surveys and Questionnaires
  - Interviews
  - Web Scraping
  - Sensors and IoT Devices
  - App Backend (User behavior data, app log data)

Data Warehouse!

What is data warehousing?

- Data warehousing is a structured approach to storing, managing, and analyzing data to support business intelligence and decision-making.
- Components of a Data Warehouse:
  - Data Sources
  - ETL (Extract, Transform, Load) Process
  - Data Storage
  - Data Access Tools

What is data warehousing?



Types of data warehouse

- Types of Data Warehouses:
  - Enterprise Data Warehouse (EDW) ⟶ Centralised data repository
  - Operational Data Store (ODS) ⟶ Temporary warehouse for real-time data, supporting immediate decision-making.
  - Data Mart ⟶ Data marts are smaller, focused data warehouses designed for specific business units or departments.

Benefits of data warehouse

- Benefits of Data Warehousing:
  - Enhanced Decision-Making
  - Improved Data Quality
  - Centralized Data Access
  - Historical Analysis
  - Scalability

Data visualization basics!

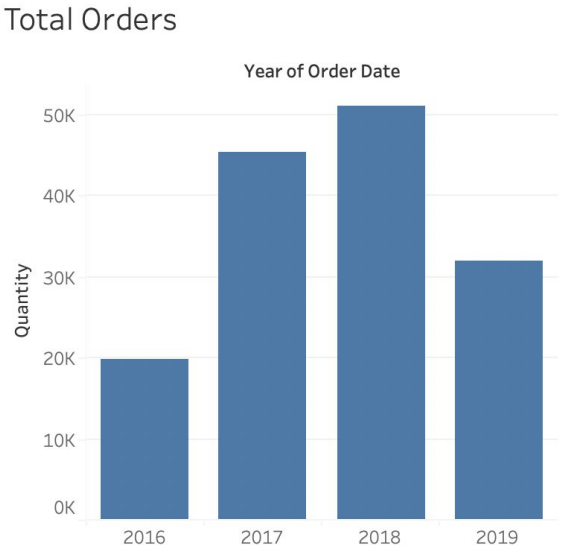
What is this?

- Simply put, the process of giving your numbers & findings a form factor that is appealing to your end user. (plain numbers are boring to most ;))

Total Orders

	Order Date			
	2016	2017	2018	2019
	19,889	45,390	51,030	32,528

VS



Why shall we use this?

- To set a context & help understand trend/pattern
- To make effective decision out of an analysis which can be convincing for everyone involved.

Steps for effective visualization

- Understand the context
- Select appropriate visual
- Focus attention where you want it
- Combine & tell a story

How shall we set the context?

Setting the context helps you and your target group to digest your story. Here is how to set one:

- Who is your audience?
- What do you need them to know?
- So what?
- What action you want the audience to take?
- Your delivery mechanism
- The 3 minutes story vs your Big idea

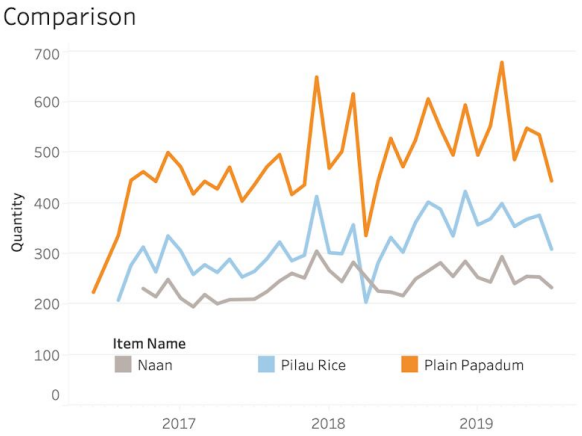
Most common graphs!

Bar Graph

- Simply use for showing trends (long period, large changes) or stacks.

Line Graph

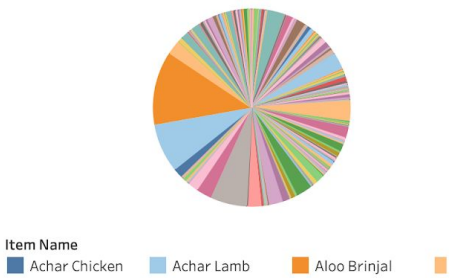
- Short period trend, small changes, multiple line item changes



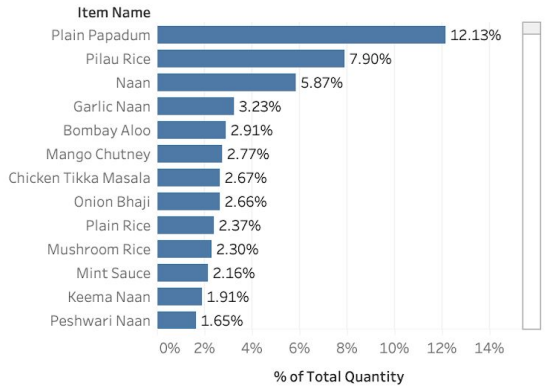
Pie chart

- Pie charts are used for numerical categorisation of data but often hard to read. Instead, use horizontal bar charts.

Pie

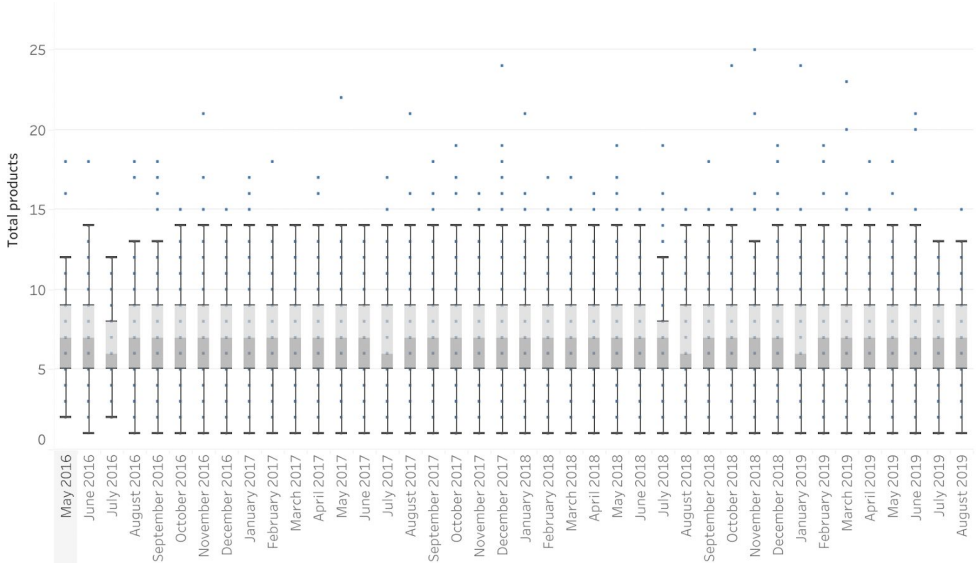


Bar



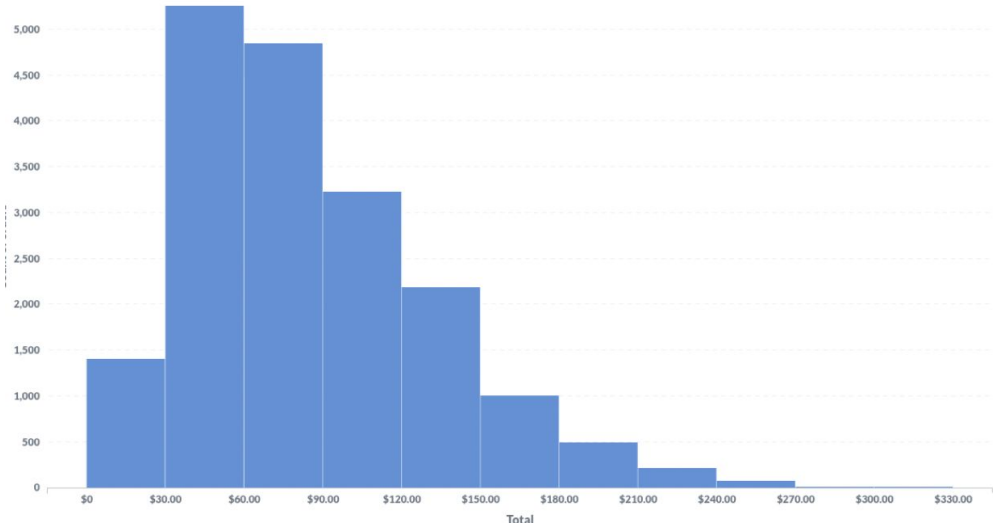
Box plots

- Box plots visually show the distribution of numerical data and skewness through displaying the data quartiles (or percentiles) and averages.



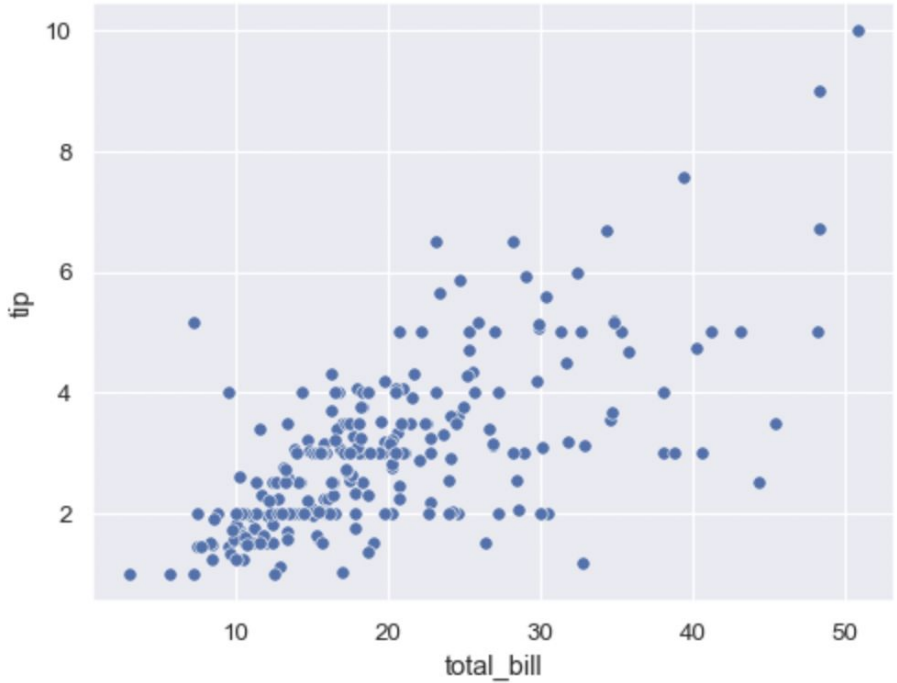
Histogram

- Histogram is helpful to determine distribution based on user specification. Similar to box plot but more detailed.



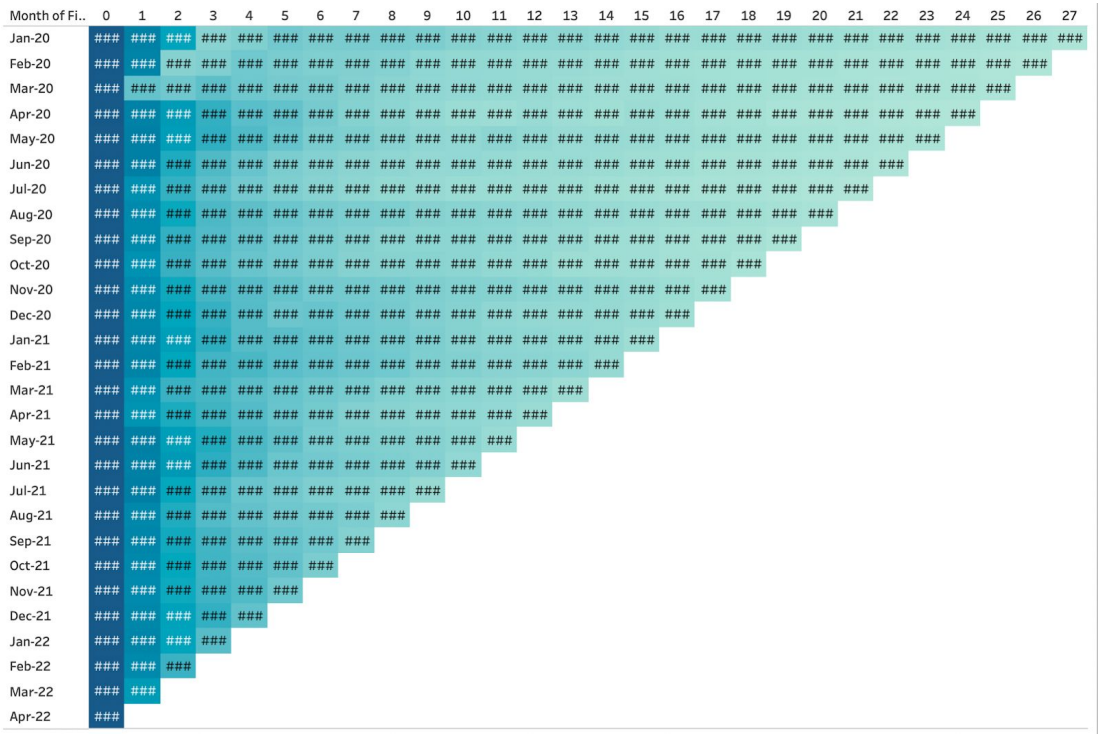
Scatterplot

- Scatterplot is used to find relationship between two numerical variables.



Retention Charts

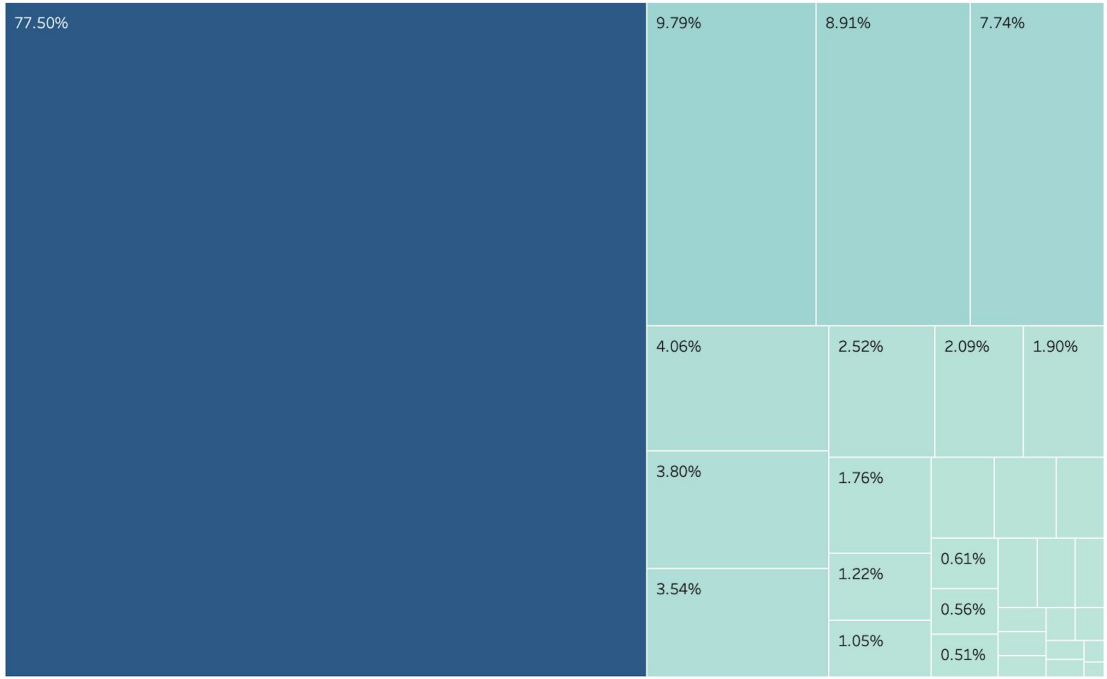
- Used to understand retention & stickiness.



Month of Fi..	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Jan-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###
Feb-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	
Mar-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###		
Apr-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###			
May-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###				
Jun-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###					
Jul-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###						
Aug-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###							
Sep-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###								
Oct-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###									
Nov-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###										
Dec-20	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###											
Jan-21	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###												
Feb-21	###	###	###	###	###	###	###	###	###	###	###	###	###	###	###													
Mar-21	###	###	###	###	###	###	###	###	###	###	###	###	###	###														
Apr-21	###	###	###	###	###	###	###	###	###	###	###	###	###															
May-21	###	###	###	###	###	###	###	###	###	###	###	###																
Jun-21	###	###	###	###	###	###	###	###	###	###	###																	
Jul-21	###	###	###	###	###	###	###	###	###	###																		
Aug-21	###	###	###	###	###	###	###	###	###																			
Sep-21	###	###	###	###	###	###	###	###																				
Oct-21	###	###	###	###	###	###	###																					
Nov-21	###	###	###	###	###	###																						
Dec-21	###	###	###	###	###																							
Jan-22	###	###	###	###																								
Feb-22	###	###	###																									
Mar-22	###	###																										
Apr-22	###																											

Tree Map

- Used to determine volume of different categories



How to tell a story?

- Understand why we are here
- Define & simplify the problem
- Show what you have found
- Describe why this matters
- End with a recommendation