Custom Bootcamp: SQL and Data

30 August 2023 09:57

Custom Bootcamp, Day1: Databases

What is a Database-collection of related data and metadata in structure/optimized
DBMS-software for easy access, creation or modification of databases
Database System-Integrated system of hardware, software, people, procedures, and data that define and regulate the collection, storage, management, and use of data within a dB env

Why?

Database: Optimize data management and transform data into information
DBMS/DBS: Stores and manages data, hides complexity of the relations from the user, providing a clean view, enforces data integrity and security

Database things

Entity: a thing about which data is to be collected and stored
Attribute: A characteristic of an entity
Relationship: Describes an association b/w entities
Constraint: Restrictions placed on the data

Electronic/manual File System

Hierarchical DB

disadv- v complex, no standards for creating the 1:m relationships, doesn't support M:N relations

Network Database

doesn't have an inherent parent child hierarchy, many 1:m sets of related data.
Allows more data types, more efficient and flexible data access, has standards.
tight bound network means any changes are hard to implement

Relational DBMS

Separated the machines view, i.e., the actual physical representation from the user view, which was more logical and could be visualized
used 2D tables that allowed for flexible logical structures for data representation
This allowed for Ad Hoc querying, using SQL

NORMALIZATION

redundancy is not the same as duplicity. Redundant data is data that might be important, but is repeating a lot in the same table AND bears a link to something else in the same table doing the same.
for example, a employee table with employee department ID and department name doesn't need both, since one of them would always be redundant.

The most common normal forms are:

First Normal Form (1NF): Ensures that each column in a table contains only atomic (indivisible) values, and there are no repeating groups or arrays within a single cell.

Second Normal Form (2NF): Builds on 1NF and ensures that the table doesn't have partial dependencies, which means that each non-key column is fully dependent on the primary key.

Third Normal Form (3NF): Builds on 2NF and eliminates transitive dependencies, where non-key attributes depend on other non-key attributes rather than just the primary key.

Boyce-Codd Normal Form (BCNF): An extension of 3NF, BCNF focuses on more complex dependencies and ensures that every determinant (set of attributes that uniquely determines another attribute) is a superkey (able to uniquely identify a row).

Fourth Normal Form (4NF): Addresses multi-valued dependencies, where a single value in one column depends on multiple values in another column.

Fifth Normal Form (5NF): Also known as Project-Join Normal Form (PJNF), 5NF deals with cases where a table contains a join dependency that can't be expressed through simpler normal forms.

DENORMALIZATION

Normalization is good for storing data efficiently without redundancy, but it is very complex to query from.
hence, denormalization combines tables to give a more informative and hence easily queried table

CANDIDATE KEY
is an attribute or a set of attributes that can uniquely identify any tuple, except for PK. A candidate key is as strong as a PK. can be more than one

SUPER KEY
a superkey is an attribute set that can uniquely id a tuple, it is a superset of a candidate key. PK is always a superkey

OLAP
Online Analytical Processing DB. takes data from OLTP, performs ETL and loads into OLAP.
OLAP is read heavy, used for analysing historical data to draw decisions
OLAP uses a dimensional table, which is a more descriptive set of attributes
OLAP is updated in batches, while OPLT is constantly updated in real time

OLTP
Online Transactional Processing DB. captures all transactional details and stores it. these are normally somewhat repetitive. OLTP is write heavy and focuses on concurrency and integrity.
OLTP is highly normalized and usually uses relational Data Structures.
OLTP is focused on easy querying, lots of volume and maintaining low latency doing it.

DIMENSIONAL MODELLING
A dimensional model has two different tables, a dimensional table and a fact table. They allow the model to separate the numeric and quantitative data into the fact table, and the descriptive attributes into the dimension table.

Fact table: Sales facts/transactional data table, with txn_id, date, prod_id, cust_key, sales amt etc. Usually you can perform calculations on these tables. contains FKs to all dimension tables. usually contains a huge amt of data

Dimension Table: Product dimension table with prod_id, Prod_name, category, brand etc.
These are comparatively fewer than the fact table and usually has descriptive tuples and data formats(text).
These tables describe the attributes of the Fact table. these tables are fully denormalized.

2 types of schema: STAR and SNOWFLAKE

STAR SCHEMA
one centralized fact table with FKs to the other dimension tables with PKs, that describe the fact table

SNOWFLAKE SCHEMA
dimension+fact, but some Dimensions are partially normalized. this allows for lower redundancy and more data integrity. It also increases the complexity in querying.
for ex, in a sales facts schema, the main fact table would have an offshoot of product details. rather than having product category and its details in the same table, these two DESCRIPTIVE tables are split to normalize them and reduce redundancy.
Picture on page "Schema Diagrams"

total qty and rev customer name wise and product name wise

```
SELECT customer_name, Product_name
sum(qty), sum(revenue)
FROM (sales_fact As s NATURAL JOIN
customer_dim AS c ON s.c_id=c.c_id
NATURAL JOIN prod_dim AS p on
p.p_id=s.p_id
GROUP BY customer_name, Product_name
```

country, year, product_name, total_qty, total_revenue

```
SELECT county, year, Product_name,
sum(qty), sum(revenue)
FROM sales_fact INNER JOIN date on
sales_fact.date_id=date.date_id INNER JOIN
customer on sales_fact.c_id=customer.c_id
INNER JOIN product p on
p.p_id=sales_fact.p_id
GROUP BY country, year, Product_name
```

DAY2:
SURROGATE KEY
a Surrogate key is a unique identifier that doesn't contain any inherent information. it is usually computer generated and numerical. Since it is numerical, it joins much faster. It also allows to maintain a history of certain transactions, where the unique identifier id(cust_id, or name, or emp_id) is repeated, as per SCD Type 2.

Slowly Changing Dimension
Is a technique to track changes in a dimension table. 3 types.

Over the course of time, data stored in a warehouse would change/get updated to reflect real-world changes.
These can be categorized into 3 types:
1. **Over-write**: Historical data is not preserved. Old entries don't matter and are written over. It is simple and easy to implement
2. **Add New Row**: Type 2 SCD involves creating a new record when changes occur, creating a new version of the dimension. This approach preserves historical data by maintaining multiple records for the same entity over time. Each record is associated with a start and end date to track when it was active. Historic data is preserved, and can be audited and analysed. This is where surrogate keys come in handy.
3. **Add New Attribute**: Type 3 SCD maintains a limited history by adding columns to the dimension table to track specific changes. Instead of creating new rows, only certain attributes are updated.

For Ex, an employee might receive a raise, or a change of position, while still retaining their EMP_ID. Thus, every new record for them will have a different surrogate key, but the same EMP_ID, allowing a historical archive.

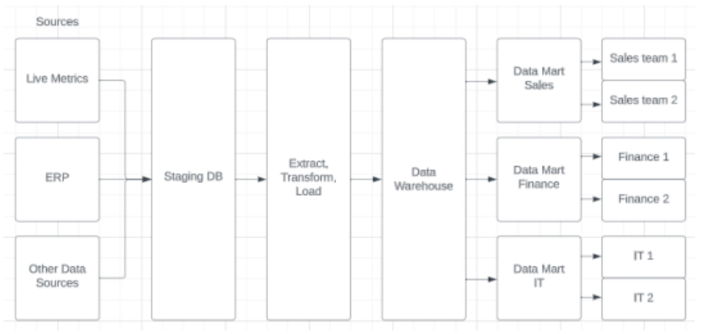
In the example above, this would include adding a column for previous_salary or prev_role. Thus, one (or more) records of histories are maintained, but its not that in depth. If you want in-depth, then use a was active. Historic data is preserved, and can be audited and analysed. This is where surrogate keys come in handy.

3. **Add New Attribute**: Type 3 SCD maintains a limited history by adding columns to the dimension table to track specific changes. Instead of creating new rows, only certain attributes are updated. This approach is suitable when only a subset of dimension attributes are expected to change over time.

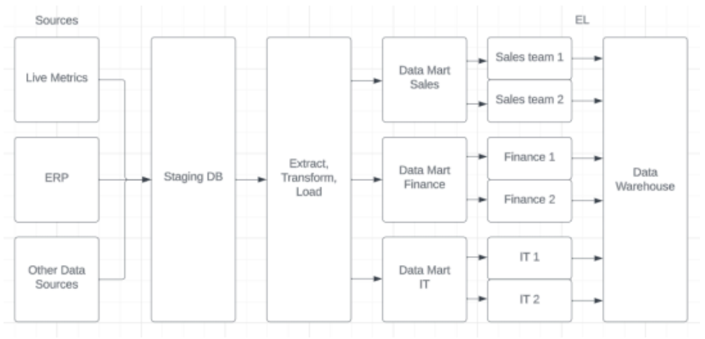
In the example above, this would include adding a column for previous_salary or prev_role. Thus, one (or more) records of histories are maintained, but its not that in depth. If you want in-depth, then use a SCD Type 2.

Data Warehouse

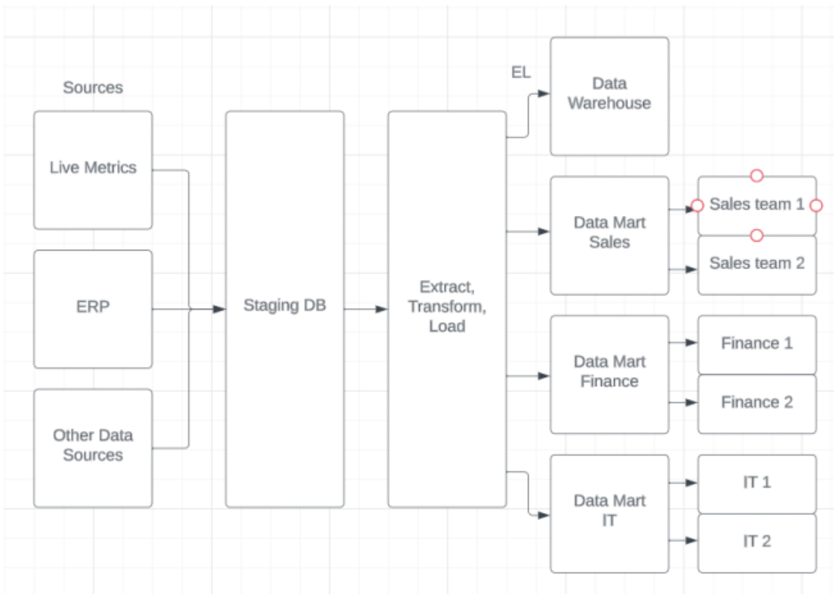
An enterprise wide collection of historical data, used for decision making and analytics.
Centralized, Subject Oriented, Integrated, Time Variant, Non-volatile.
Data from a datawarehouse would flow down to a data mart and so on, in increasingly smaller chunks.
Data here is dirty, and usually has to undergo an ETL function before it goes into the warehouse. Stores structured data.



Used if the data flow is less. Top down approach



Used if data flow is high. ETL is conducted by each team, if needed. It is uploaded to the warehouse AFTER it is used by each team. This breaks up the work into smaller chunks, and allows the data to be kept for historical/review purposes.



A hybrid method, where the data is ETL'd and loaded into the marts and WH simultaneously, ETL done by individual teams.

Big Data

Big Data is a huge amount of data that is also growing exponentially. And is very complex.
Thus, characteristics:

1. Volume
2. Variety
3. Velocity

Hadoop

Hadoop is an open-source framework designed to store and process large volumes of data across a distributed cluster of commodity hardware.
Hadoop provides a powerful and scalable solution for dealing with the challenges posed by processing massive amounts of data in parallel.
Can work with a variety of data, structured, unstructured or semi structured
Hadoop's distributed architecture and fault tolerance mechanisms make it well-suited for processing and analyzing data that is too large to be handled by traditional databases or single machines.
At its core, Hadoop consists of two primary components:

Hadoop Distributed File System (HDFS):
HDFS is a distributed file system designed to store vast amounts of data across a cluster of machines. It breaks large files into smaller blocks (typically 128MB or 256MB in size) and replicates these blocks across different nodes in the cluster for fault tolerance. This replication ensures that if a node fails, the data can still be accessed from other nodes. HDFS is optimized for data streaming and is well-suited for storing large files, such as log files, images, videos, and other unstructured or semi-structured data.

MapReduce:
MapReduce is a programming model and processing framework that enables the parallel processing of large datasets. It's used to write programs that can process and analyze data stored in HDFS. MapReduce breaks down tasks into two stages: the Map stage and the Reduce stage. In the Map stage, data is processed and transformed into key-value pairs. In the Reduce stage, these key-value pairs are grouped and aggregated to produce final results. MapReduce provides fault tolerance, as tasks can be rerun on other nodes in case of failure. Also has built in resource manager and a scheduler, like our Docker restartattempts system.

Apache provides other software that can be plugged in from its ecosystem. These include
Apache Hive: A data warehousing and SQL-like query language for querying and analyzing data stored in HDFS.
Apache Pig: A platform for analyzing large datasets using a high-level scripting language.
Apache HBase: A NoSQL database that provides random access to large amounts of structured data stored in HDFS.
Apache Spark: A fast and flexible data processing and analytics engine that can run on top of Hadoop clusters.
Apache YARN: A resource management and job scheduling framework that allows different data processing frameworks (like MapReduce and Spark) to share cluster resources efficiently.
Apache ZooKeeper: A distributed coordination service used for managing configuration information, naming, synchronization, and group services.

Hadoop Terminologies
Node-a server, Rack, Cluster. Hadoop works over a distributed architecture composed of thousands of nodes.

Hadoop Distributed File System
HDFS runs on top of an existing Block Storage, and is designed to tolerate high component failure rates.
In HDFS, large files are divided into fixed-sized blocks, typically ranging from 128 MB to 256 MB. These blocks are then distributed and replicated across the nodes in a Hadoop cluster.
Each block is replicated across multiple nodes in the Hadoop cluster. The replication factor is also configurable but is often set to three by default.
HDFS performs checksum validation to ensure data integrity. When data is read, the checksum is verified to detect any corruption.

HDFS does not use RAID in the traditional sense that RAID is used in conventional storage systems. However, HDFS has its own mechanisms for ensuring data reliability and fault tolerance, which are similar in concept to RAID but implemented differently to suit the distributed and scalable nature of Hadoop clusters.

Hadoop has different types of nodes:

- Name Node: Actual data doesn't go through or store into the name node, but it does store storage system namespaces and metadata such as permissions and replication factor), and the mapping of data blocks to DataNodes. The NameNode keeps track of the data blocks and their locations across the cluster. Since it is a SPoF, it should always be mirrored. The name node is critical, and therefore, the hardware and architecture for it should be enterprise grade. The metadata that this node stores is also stored in RAM, so it is very expensive. Responsibilities:
    - Maintains the namespace of the file system.
    - Manages the metadata for files and directories.
    - Keeps track of the data block locations on DataNodes.
    - Handles client requests for file operations, such as reads, writes, and deletions.
    - Monitors the health and status of DataNodes in the cluster.
- Data Node: DataNodes are responsible for storing the actual data blocks that make up the files in HDFS. They manage the physical storage of data on local disks and communicate with the NameNode to report the status of the stored data blocks. DataNodes periodically send heartbeat signals and block reports to the NameNode to provide information about their health and the blocks they are storing. These do not require high quality hardware. Responsibilities:
    - Store actual data blocks on local storage devices.
    - Report the status of stored data blocks to the NameNode.
    - Respond to read and write requests from clients and other DataNodes.
    - Replicate data blocks to achieve the desired replication factor.
    - Communicate with the NameNode for cluster management and health monitoring.
- JobTracker: part of the MapReduce infra, one per Hadoop cluster. Manages MapReduce jobs in the cluster. Receives job request submitted by a client, and then selects resources, breaks the job into Map and Reduce tasks. Monitors and schedules the jobs through the TaskTracker nodes.
- TaskTracker: actually runs the MapReduce tasks. Monitors task execution and reports progress, status, and task completion to the JobTracker.

These are old terminologies. these tasks have been taken over by the ResourceManager and ApplicationMaster functions.

YARN (Yet Another Resource Negotiator)
Introduced in hadoop2.0, Now has node managers, application managers, and resource managers, all of which manage containers where the actual jobs are executed. Allows for more name nodes and redundancy.

Hadoop also has Rack Awareness
Apache Spark
MapReduce reads and writes to a disk, which makes it slow. Spark stores intermediate data (read, but not written) into memory caches and optimizes query execution, making it significantly faster. Hadoop MapReduce works best with batch data, spark works well with real-time data.

AzureDataLake

A central repo designed to store, process and secure large amt of structured, semi-structured or unstructured data. Uses ELT, over ETL, as in, loads first, the provides the option of editing and transforming.
It stores data in its native format and can process any variety of data, regardless of size. Usually using inexpensive storage, it is made highly scalable, if on prem.
Azure data lake uses Azure Blob to store, and YARN as a base to analyze.
ADLAnalytics uses no visible VMs and abstracts all backend processes, allowing it to process Big Data batches in seconds, upto petabytes. It uses U-SQL, a variant of SQL.
ADLStorage has 2 gens. Has all the cloud perks of scalability, etc.
Gen 1 is made for Hadoop, which allows easy processing through Hive and MapReduce. It is practically unlimited in storage, and is highly available and secure.
Gen2 is build on Azure Blob storage and has all the key features of Gen1.

Also adds:	File System Semantics	File-level security	Directory	HA/DR

General improvements to be more optimized and custom for the interactions between Hadoop and ADLSGen2.

Data lakes generally allows for more in-depth analysis, suitable for MLOps etc, that require large amounts of data.

DAY 3

User: Shellunext_1693422149102@npunext.onmicrosoft.com
Pwd: [redacted]
Azure SQL hands-on:
DDL (data definition)
CREATE: to create any SQL objects
TRUNCATE: used to delete entire rows/columns etc. cannot be rolled back. No conditions attached
ALTER
DROP

DML(data Manipulation)
INSERT
UPDATE
DELETE: can be rolled back and can have conditions attached
SELECT

DCL(Data Control)
GRANT
REVOKE

TCL (transactional control)
COMMIT: permanently changed the DB table
ROLLBACK: Undoes the changes from the last commit
SAVE POINT: allows rollback to a prev save. Rolling back also destroys the save point, so it has to be recreated.

Sp_tables to list tables
Sp_columns <tabel_name> to list all columns
Alter table table_name drop column column_name
Alter table table_name alter column column_name data_type null/not null/unique/primary_key/foreign_key
`sp_rename '<table_name>.<column>', new_name, 'column'`
Sp_rename old_table_name, new_table_name
Check constraint (checks condt, only then allows the value to be inserted)

```
create table sizet(id int identity(1,1), sizet text, color text, constraint sizeq uniq
ue(sizet,color))
insert into sizet values ('l', 'red')
insert into sizet values ('m', 'blue')
insert into sizet values('l', 'red')
```
would fail, because the combination of sizet and color should be unique

```
CREATE TABLE supplier (
    supplier_id INT PRIMARY KEY,
    supplier_name VARCHAR(100),
    contact_person VARCHAR(50),
    phone_number VARCHAR(20)
);
CREATE TABLE product (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    price DECIMAL(10, 2),
    supplier_id INT,
    FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id)
);
INSERT INTO supplier (supplier_id, supplier_name, contact_person, phone_number)
VALUES
  (1, 'ABC Electronics', 'John Smith', '123-456-7890'),
  (2, 'XYZ Components', 'Jane Doe', '987-654-3210'),
  (3, 'GHI Manufacturing', 'Michael Johnson', '555-555-5555'),
  (4, 'LMN Supplies', 'Emily Williams', '111-222-3333'),
  (5, 'PQR Tech', 'Daniel Brown', '444-333-2222');
-- Inserting 5 tuples into the "product" table
INSERT INTO product (product_id, product_name, price, supplier_id)
VALUES
  (101, 'Laptop', 999.99, 1),
  (102, 'Smartphone', 599.99, 1),
  (103, 'LED TV', 699.99, 2),
  (104, 'Hard Drive', 79.99, 3),
  (105, 'Keyboard', 19.99, 4);
```
A Query like this would fail:
```
delete from [dbo].[supplier] where supplier_id=1

This is because the supplier ID here is a FK constraint and this bars it from being
deleted.

Neither would DROP TABLE supplier
```
EMP and DEPT table script – SQL Server – Data Analytics and BI WORLD (wordpress.com)

Single row functions: that execute one row at a time
These include: String, Number, Date, Conversion and Analytical functions

Group Functions

```
select concat(upper(left(ename,1)), Lower(substring(ename,2,len(ename)))) from emp
```
Uses the CONCAT, UPPER, LOWER, SUBSTRING and LEN functions
Gives Result:

Smith
Allen
Ward
Jones

i.e properly capitalizes a fully capital name

Other functions: ABS, RAND, CEILING, FLOOR, POWER, LOG, PI

ISNULL(givenval, column) replaces nulls with the given value
NULLIF(column, <string_to_be_matched>) replaces the matched string with NULL
IIF(<condition>, If_true, If_false), can be nested
Eg:

```
select sal, iif(sal>3000, 'good salary', iif(sal>2000, 'avg salary', 'bad salary')) sa
l from emp
```
Gives Result:

sal	sal_desc
5000.00	good salary
3000.00	avg salary
3000.00	avg salary
2975.00	avg salary
2850.00	avg salary
2450.00	avg salary
1600.00	bad salary

CAST(val, AS <datatype>)

Eg:

Do the same stuff

```
Select cast('07-11-2023' AS date)
```
Gives result:
2023-07-11T00:00:00.0000000

GETDATE(), SYSDATETIME(), YEAR/MONTH/DATE(GETDATE()), DATENAME(day/month/year, <date>)
Eg:

```
Select SYSDATETIME()
```
Gives result in UTC

```
select distinct(Year(hiredate)) as Joining_Year from emp
```
Gives:

Joining_Year
1980
1981
1982
1983

DATEDIFF(dd/mm/WK. '<date1>', '<date2>')
DATEADD (year/month/day/week/hour, <no_toadd>, '<date/time>') as <alias>

SELECT <stuff1>, stuff2, RANK() OVER(ORDER BY <thing_to_be_ranked_over>) as ALIAS from TABLE
Eg:

```
SELECT empno, ename, sal, rank()over(order by sal desc) as Sal_rank from EMP
```
Gives:

empno	ename	sal	Sal_rank
7839	KING	5000.00	1
7902	FORD	3000.00	2
7788	SCOTT	3000.00	2
7566	JONES	2975.00	4
7698	BLAKE	2850.00	5

7782	CLARK	2450.00	6
7499	ALLEN	1600.00	7
7844	TURNER	1500.00	8
7934	MILLER	1300.00	9
7521	WARD	1250.00	10
7654	MARTIN	1250.00	10
7876	ADAMS	1100.00	12
7900	JAMES	950.00	13
7369	SMITH	800.00	14

Join Constraint to find each employee's manager+the president(who doesn't have a manager)

```
select e.empno, e.ename, e.mgr, m.ename as managername, d.deptno, dname from emp e lef
t join emp m on e.mgr=m.empno join dept d on e.deptno=d.deptno
```

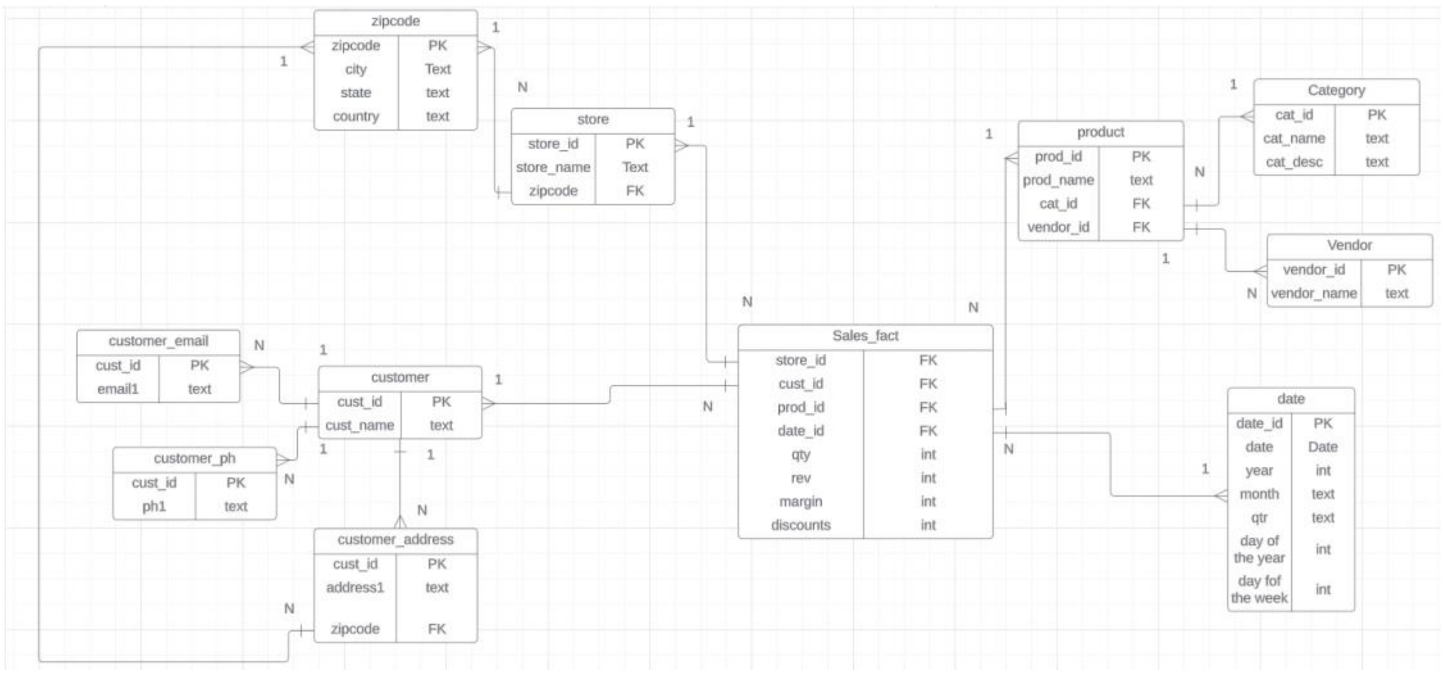
If in a select statement you select a non-aggregated column with a bunch of other aggregates, you have to use a group by on the non-aggregated col.
Having will operate on the result of this aggregation

```
select dname, sum(sal) as sum, MAX(sal) as max, min(sal) as min, avg(sal) as avg, coun
t(sal) as count
from emp e join dept d on e.deptno=d.deptno
group by dname
having count(sal)>=5
```

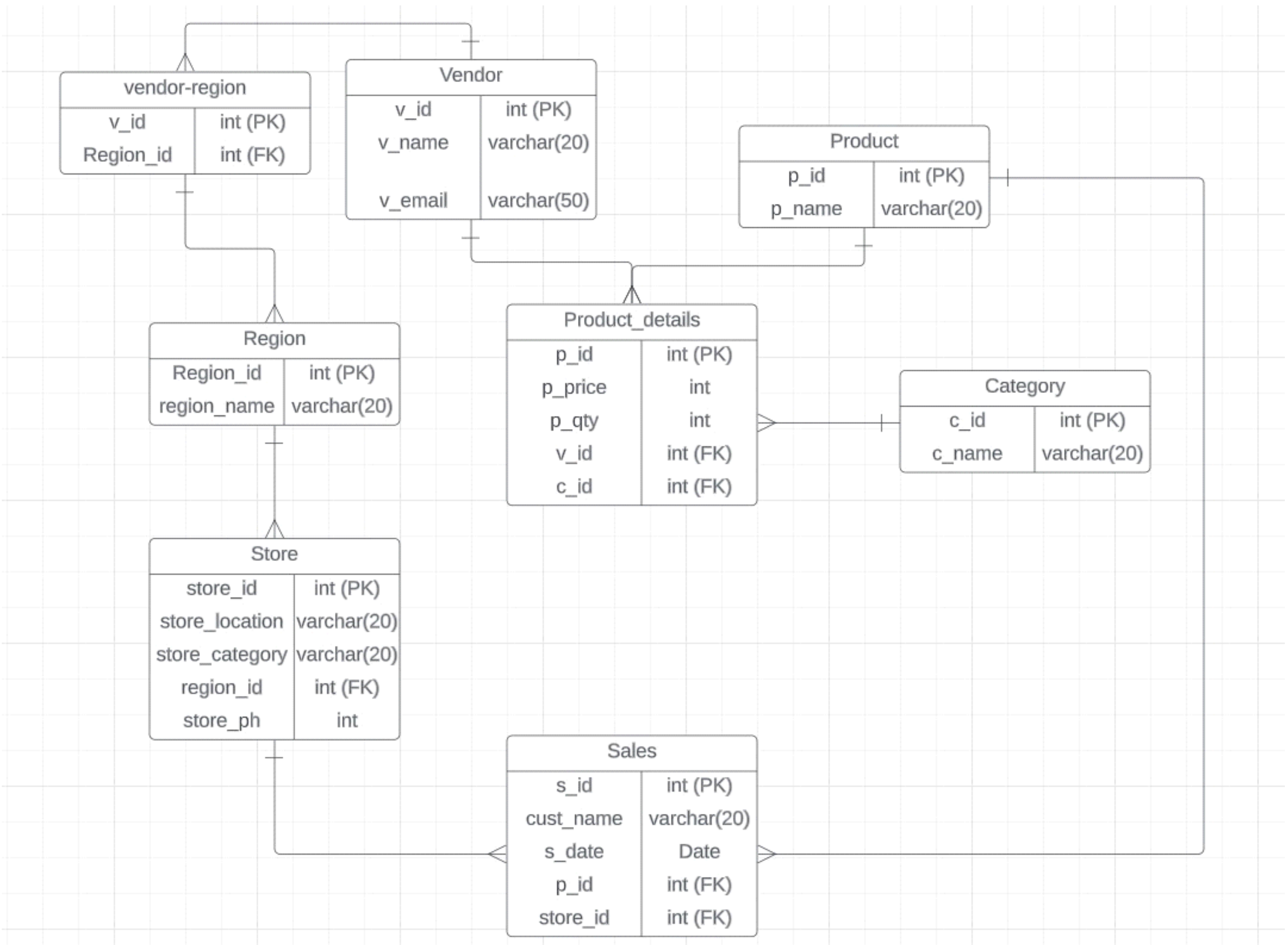
SET operators:
UNION, MINUS, INTERSECTION, UNION ALL

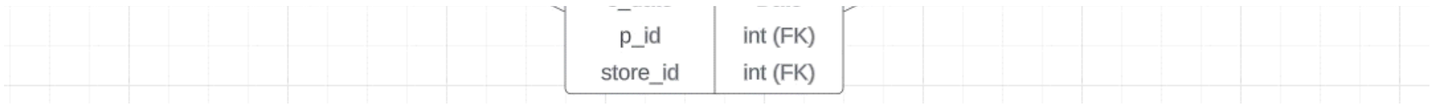
Schema Diagrams

30 August 2023 10:04

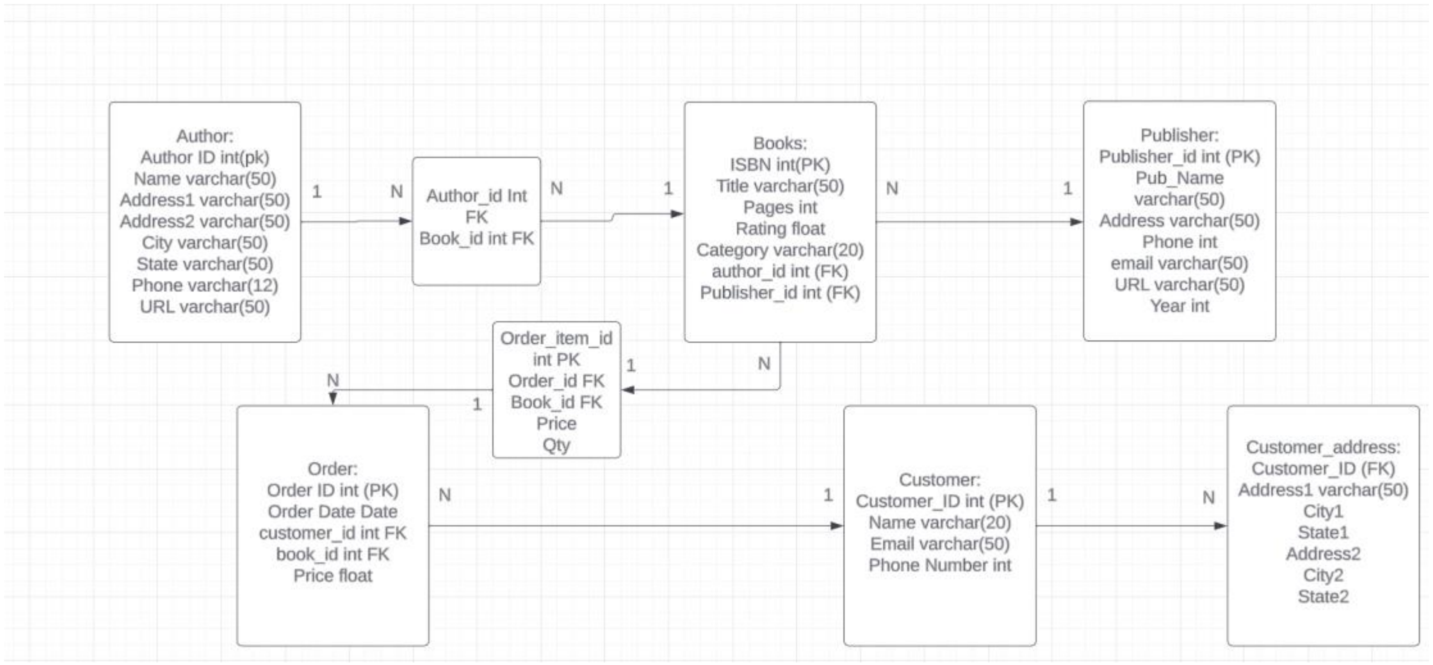


SNOWFLAKE SCHEMA





SCHEMA EX 1



N:N relation SCHEMA

Custom Bootcamp: Azure Storage

30 August 2023 15:05

Day1:

Types of data:
Dead: file>storage
Live: stream

To access file storage (which is what Azure blob storage is):
3rd party APIs, Portal, Templates, CLI

Azure Storage Account:
Blob: Raw data (Binary Large Object). Can be any type of file
Tables: structured data
Queues: ordered input and output, creates a pipeline
Files: File storage (Network File System, EFS, etc)

BLOB: serves images, audio, video, docs directly to a browser
Store files in a distributed architecture
Use it for audio and video streaming
Reading/writing logs
Store data for backup and restore, DR/HA and archiving
Can be on-prem, hybrid or hosted

Container accessible thru web by: https://StorageAccountName.blob.core.windows.net/mycontainer
Blob supports upto 190TB of data

https://sabloba2590.blob.core.windows.net/website/index.html

AzCopy (CLI) command to copy to azure blob

Blob Gen1 and Gen2 (also used by ADLS)

Category	Blob	Gen1	Gen2
Purpose	General Purpose obj storage for a wide variety of data types	Optimized Storage for Big Data Analytics Workspace	Extends the capabilities of Blob, well optimized for large scale analytics workspace
Redundancy	Local, Zonal, Geo	Local and Geo	L, G, Z, GZ
Authentication	Access Key Shared Account Access	Azure Active Directory IAM	AAD, Shared Account Access
Auth	RBAC	RBAC, ACL	RBAC, ACL
Vnet Support	Vnet integration	Vnet integration	Service endpoints Private endpoints

Azure Active Directory: works as a central access control system. Once authenticated, it gives access to everything too, base d on that auth.

Types of auth
SSH
AK/SS
Connection String
JWT
PAT
SHA
Monitoring: Metrics, Ingress and Egress monitoring, Alerts, Webhooks
Logging: Critical logs, verbose logs, LOG_LEVEL, error log, Auditing etc.

Storage presents a nice dashboard, with various metrics such as:
Aggregated transactions, statistics and capacity data
Capacity Metrics:
Account-level metrics:

Name	Used Capacity
Unit	Bytes
Aggregation Type	Average
Value	1024

Blob storage metrics

Name	Ingress
Unit	Bytes
Aggregation Type	Total
Applicable Dimensions	GeoType, APIName, Authent
Threshold Value	10

Name	Egress
Unit	Bytes
Applicable dimensions	Same as above
Value	1024

For Alerts

About requests to the storage service.
Metric data can be used to:
Analyse the storage service's usage.
Diagnose and troubleshoot issues with requests made to the service
Improve the performance of applications that use these services.

Alerts:
Create alert rules based on monitoring metrics, then create an Action.
Actions support notification based communication or service level actions such as running an automation playbook, or communic ating to another service or triggering a webhook.

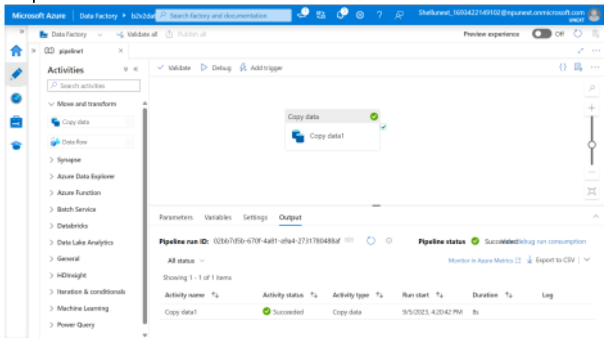
Day 2:
Various storage metrics and alerts configured, ss taken
Azure Data Factory: ETL-as-a-Service

Linked Service: Connects and creates workflows from and to ADF.
Data Sources/destination

Uses Azure Active Directory to connect to other
Azure Services

Demo:
Create 2 Storage Acc containers: input and output.
Keep a file in input, use ADF to push this file into the output container.
Go to ADF, and create a linked service for the SA
Manage>LS>SA
Author>Pipeline>Tasks(COPY)-Source and Sink define
Execute it

Demo2:
Copy a file from a Blob SA to a DataLake v2 SA
Requires two linked services from each SA



Pipeline run ID: 02bb7d5b-670f-4a81-a9a4-2731780488af
Pipeline status: Succeeded
View debug run consumption

Showing 1 - 1 of 1 items

Activity name	Activity status	Activity type	Run start	Duration	Log
Copy data1	Succeeded	Copy data	9/5/2023, 4:20:42 PM	8s	

Can be done by
Author>Dataset>New Dataset with the input file from the blob SA and putput lined to the Gen2 SA.
Then create a new pipeline, but rather than link the SA, we can now link the datasets.

Synapse Analytics

12 September 2023 12:10

Data:

- Live
- Dead:
  - Traditional: SQL style structured data. DWHs: PowerBI/MSBI for viz, Informatica for ETL
  - Big Data: Spark
  - New Data: Azure Data Explorer

Synapse is good for DWH and Big Data: Data Integration, Management, Monitoring, and Security
Collects On-prem, cloud, SaaS and streaming data.
Data governance managed by Azure Purview.
Accelerate time to insight across enterprise warehouses and big data systems
It brings together the best of SQL tech in DWF, Spark for ML enabled analytics and azure data explorer for log and timeseries analytics

Synapse is used when:

- Need a managed service
- Large dataset and complex query
- Manage structured and unstructured DS
- Data Pipeline Orchestration
- Real time data
- Integrate MS Services

Serverless SQL Pool	Dedicated SQL Pool
Pay-as-you-go Model	Pay for Dedicated compute resources which allow you control and optimize performance at a DataWarehousing Unit Level

https://sa2590.dfs.core.windows.net/containername/filename

```
select top 100 *
from OPENROWSET(
    bulk 'https://sa2590.blob.core.windows.net/sa2590fs/yellow_tripdata_2023-01.parquet',
    format = 'PARQUET'
) As [Result]
Create database DataExplorationDB
    Collate Latin1_General_100_BIN2_UTF8
use DataExplorationDB
Create external data SOURCE sa2590
with (LOCATION='https://sa2590.dfs.core.windows.net')

CREATE LOGIN a2590_user with PASSWORD ='qwerty@1234' --pool user


create user DBa2590_user for login a2590_user; --database user
GO
GRANT ADMINISTER database BULK OPERATIONS to DBa2590_user;
GO
```

```
--with a relative data path and source
select top 100 *
from OPENROWSET(
    bulk '/sa2590fs/yellow_tripdata_2023-01.parquet',
    DATA_SOURCE = 'sa2590',
    format = 'PARQUET'
) As [Result]
```

Pool(Cluster)
- SQL
  - Serverless
    - Custom
    - Built-in
  - Dedicated
    - Custom
    - Built-in
- SPARK
  - Big data>Notebook>set of cells
    - MarkDown
    - System/Magic Command
    - Python Code

Dedicated SQL pool name: a2590_dpool

Pyspark Notebook

```
%%pyspark
url='abfss://sa2590fs@sa2590.dfs.core.windows.net/yellow_tripdata_2023-01.parquet'
df = spark.read.load(url, format='parquet')
display(df.limit(10))
#displays the first 10 rows of the parquet file using dataframes
```

PowerBI

13 September 2023 09:48

Instance Storage Data
Ephemeral Storage Data
MDF-Meta Descriptor File/Master Database File
LDF-Log File
Created when you make a DB
BAK file-a compressed backup and restore file in a DB
NDF

PySpark

20 September 2023 09:45

Lifecycle of spark:

1. Submit application
2. Initialization of the env: when the application is submitted, a Spark program is launched on a cluster node
3. The Driver program initializes Spark Context, which is responsible for coordinating job executions
4. The Spark Application is divided into one or more jobs
5. A Stage is created for each RDB transformation
6. DAG generation:
    a. Spark constructs a DAG (Directed Acyclic Graph) that represents the logical execution plan.
    b. DAG contains info about dependencies between stages and tasks
7. Task Scheduling: The scheduler takes the DAG and schedules tasks for execution
8. Task Execution: Tasks are executed on worker nodes in parallel or in a distributed workload
9. (OPTIONAL) If your application involves transformations that require data shuffling then you can use this as another lifecycle stage: Data shuffling and exchange
10. Task Completion: As tasks complete, they produce immediate results. These results are cached or persistent (in memory) for subsequent tasks. This reduces the need for any recalculation/compilation for future tasks. It also creates a completed flag
11. Stage completion: Stages are marked as complete when all of their tasks are finished successfully
12. Job Completion: When all stages are complete
13. Garbage Collection

Job:

1. Read
2. Write
3. For Each RDB, collect/count etc

Stages:

1. Map
2. Filter

Tasks: A single operation applied to a single partition. It is executed as a single thread in an executor

Spark unifies:

1. Batch Processing
2. Real-time processing
3. Stream analytics
4. ML
5. Interactive SQL

All of this works on:

1. Yarn
2. Mesos
3. Standalone Scheduler
4. Kubernetes

```
#Create a PySpark session
Spark-submit \
--master spark://MasterIP:7077 \
--py-files sparkapp.zip \
--files input_data.csv \
Sparkapp.py

from pyspark.sql import SparkSession as ss
spark=ss.builder.appName("app1").getOrCreate()

#Read data and store in a variable
data = spark.read.csv("temp.csv", header=True, inferSchema=True)
print(data)

>>DataFrame[Day: string, Weather: string, Temperature: double, Wind: int, Humidity: int]

data.show()

>>>>>+---+-------+-----------+----+--------+
     |Day|Weather|Temperature|Wind|Humidity|
     +---+-------+-----------+----+--------+
     |Mon|  Sunny|      12.79|  13|      30|
     |Tue|  Sunny|      19.67|  28|      96|
     |Wed|  Sunny|      17.51|  16|      20|
     |Thu| Cloudy|      14.44|  11|      22|
     |Fri| Shower|      10.51|  26|      79|
     |Sat| Shower|      11.07|  27|      62|
     |Sun|  Sunny|       17.5|  20|      10|
     +---+-------+-----------+----+--------+
```

```
df.filter(df.Day.isin ("Sat", "Sun")).show()
>>
+---+-------+-----------+----+--------+
|Day|Weather|Temperature|Wind|Humidity|
+---+-------+-----------+----+--------+
|Sat| Shower|      11.07|  27|      62|
|Sun|  Sunny|       17.5|  20|      10|
+---+-------+-----------+----+--------+
```

```
df.filter(df.Day.isin ("Sat", "Sun")).write.parquet('output.parquet')

spark.read.parquet('output.parquet').show()

>>
+---+-------+-----------+----+--------+
|Day|Weather|Temperature|Wind|Humidity|
+---+-------+-----------+----+--------+
|Sat| Shower|      11.07|  27|      62|
|Sun|  Sunny|       17.5|  20|      10|
+---+-------+-----------+----+--------+
```

App:

```
  GNU nano 4.8                                              a1.py
from pyspark.sql import SparkSession as ss
spark=ss.builder.appName("RDDMapExample").getOrCreate()
data = [1,2,3,4,5]
rdd = spark.sparkContext.parallelize(data)
def double(x):
        return x*2
result_rdd=rdd.map(double)
result=result_rdd.collect()
print(result)
```

Output:

```
[2, 4, 6, 8, 10]
```

Resilient Distributed Datasets Ops:

1. Actions:
   a. Collect
   b. First
   c. Reduce
   d. Count
   e. SaveAsText
2. Transformations:
   a. Map
   b. Filter
   c. ReduceBy
   d. Join

Actions	Transformations
Actions are ops on RDD that triggers the execution of Spark computation	Ops on RDD that create a new RDD as a result
The force the evaluation of Transformations and produce a result (perform an Action)	Used to build the computation pipeline and define Data Pipeline sequences
Eagerly executed	They are lazy evaluated
Actions are used to retrieve results, save data, and trigger side effects in Spark	

Why RDD:

- Performance Optimization
- Easy to use and to abstract
- In-memory processing
- Fault Tolerant
- Data Sharing/Distributed Processing
- Parallel Processing

- Compatibility
- Integrate with external storage

SparkContext is the main entrypoint for spark
It enables and contains all the functionality for Spark
The Spark Driver contains the Directed Acyclic Graph creator and scheduler, Task Scheduler, Backend Scheduler, and block manager, which are responsible for translating user written code into jobs that are actually executed on the cluster

===

SparkContext vs SparkSession
Creating a schema for SparkSession SQL and putting it on to a DataFrame

```
from pyspark import SparkContext
import findspark
!pip install findspark
import findspark
findspark.init()

#create a spark context
sc=SparkContext("local", "SparkPractice")

    Setting default log level to "WARN".
    To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
    23/09/22 05:52:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
    builtin-java classes where applicable

data=[1,2,3,4,5]

rdd=sc.parallelize(data)

names=sc.parallelize(['Krishna', 'Mark', 'Geralt', 'Ciri', 'Triss', 'Yennefer', 'Dandelion', 'Zoltan', 'Vesemir'])

names.collect()

    ['Krishna',
     'Mark',
     'Geralt',
     'Ciri',
     'Triss',
     'Yennefer',
     'Dandelion',
     'Zoltan',
     'Vesemir']
num=sc.parallelize([1,2,3,3,2,4],5)

num.getNumPartitions()

    5
num.collect()

    [1, 2, 3, 3, 2, 4]

sc.getConf().getAll()

    [('spark.master', 'local'),
     ('spark.app.startTime', '1695361923460'),
     ('spark.driver.port', '36445'),
     ('spark.executor.id', 'driver'),
     ('spark.driver.extraJavaOptions',
      '-Djava.net.preferIPv6Addresses=false -XX:+IgnoreUnrecognizedVMOptions --add-
    opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UNNAMED --add-
    opens=java.base/java.lang.reflect=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-
    opens=java.base/java.net=ALL-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED --add-
    opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-
    opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED --add-opens=java.base/sun.nio.ch=ALL-
    UNNAMED --add-opens=java.base/sun.nio.cs=ALL-UNNAMED --add-opens=java.base/sun.security.action=ALL-
    UNNAMED --add-opens=java.base/sun.util.calendar=ALL-UNNAMED --add-
    opens=java.security.jgss/sun.security.krb5=ALL-UNNAMED -Djdk.reflect.useDirectMethodHandle=false'),
     ('spark.rdd.compress', 'True'),
     ('spark.serializer.objectStreamReset', '100'),
     ('spark.submit.pyFiles', ''),
     ('spark.submit.deployMode', 'client'),
     ('spark.driver.host', 'ip-172-31-2-124.ap-south-1.compute.internal'),
     ('spark.app.id', 'local-1695361925272'),
     ('spark.app.submitTime', '1695361923222'),
     ('spark.app.name', 'SparkPractice'),
     ('spark.ui.showConsoleProgress', 'true'),
     ('spark.executor.extraJavaOptions',
      '-Djava.net.preferIPv6Addresses=false -XX:+IgnoreUnrecognizedVMOptions --add-
    opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UNNAMED --add-
    opens=java.base/java.lang.reflect=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-
    opens=java.base/java.net=ALL-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED --add-
    opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-
    opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED --add-opens=java.base/sun.nio.ch=ALL-
    UNNAMED --add-opens=java.base/sun.nio.cs=ALL-UNNAMED --add-opens=java.base/sun.security.action=ALL-
    UNNAMED --add-opens=java.base/sun.util.calendar=ALL-UNNAMED --add-
    opens=java.security.jgss/sun.security.krb5=ALL-UNNAMED -Djdk.reflect.useDirectMethodHandle=false')]
```

```
num.countByValue()
```

```
defaultdict(int, {1: 1, 2: 2, 3: 2, 4: 1})
```

```
a=sc.parallelize([1,2,3,4,5]).foreach(lambda x: f(x))
```

```
1
2
3
4
5
```

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

data=[('Alice', 25), ('Bob', 30), ('Krishna', 22), ('Charlie', 21), ('Eve', 22), ('David', 19)]
schema=StructType([
   StructField('Name', StringType(), True),
   StructField('Age', IntegerType(), True)
])

spark=SparkSession.builder.appName("A").getOrCreate()

rdd=spark.sparkContext.parallelize(data)

rdd.collect()
```

```
[('Alice', 25),
 ('Bob', 30),
 ('Krishna', 22),
 ('Charlie', 21),
 ('Eve', 22),
 ('David', 19)]
```

```
df=spark.createDataFrame(rdd, schema=schema)
df.show()
```

```
+-------+---+
|   Name|Age|
+-------+---+
|  Alice| 25|
|    Bob| 30|
|Krishna| 22|
|Charlie| 21|
|    Eve| 22|
|  David| 19|
+-------+---+
```

```
df.filter(df.Age>=22).show()
```

```
+-------+---+
|   Name|Age|
+-------+---+
|  Alice| 25|
|    Bob| 30|
|Krishna| 22|
|    Eve| 22|
+-------+---+
spark.stop()
```

==

Working with the Movie Dataset:

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
import findspark
findspark.init()

sc=SparkContext("local", "SparkPractice")
    Setting default log level to "WARN".
    To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
    23/09/22 06:40:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
    builtin-java classes where applicable
    23/09/22 06:40:59 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.

spark=SparkSession.builder.appName("A").getOrCreate()
movies=spark.read.csv('movies.csv', header=True, inferSchema=True)

#remove space from headers
mcol=movies.columns
ncol=list(map(lambda item: item.replace(' ',''), mcol))
from functools import reduce

movies_df = reduce(lambda data, idx: data.withColumnRenamed(mcol[idx], ncol[idx]), range(len(mcol)), movies)
movie_rdd=movies_df.rdd
movies_df.createOrReplaceTempView('MOVIETABLE')
spark.sql("SELECT * FROM MOVIETABLE WHERE Genre='Romance'").show()
    +--------------------+-------+--------------------+--------------+-------------+---------------+--------------+----+
    |                Film|  Genre|          LeadStudio|Audiencescore%|Profitability|RottenTomatoes%
    |WorldwideGross|Year|
    +--------------------+-------+--------------------+--------------+-------------+---------------+--------------+----+
    |Zack and Miri Mak...|Romance|The Weinstein Com...|            70|  1.747541667|             64|       $41.94 |2008|
    |            Waitress|Romance|         Independent|            67|   11.0897415|             89|       $22.18 |2007|
    | Waiting For Forever|Romance|         Independent|            53|        0.005|              6|        $0.03 |2011|
    |Tyler Perry's Why...|Romance|         Independent|            47|    3.7241924|             46|       $55.86 |2007|
    |Twilight: Breakin...|Romance|         Independent|            68|  6.383363636|             26|      $702.17 |2011|
    |            Twilight|Romance|              Summit|            82|  10.18002703|             49|      $376.66 |2008|
    |  Something Borrowed|Romance|         Independent|            48|  1.719514286|             15|       $60.18 |2011|
    |     P.S. I Love You|Romance|         Independent|            82|  5.103116833|             21|      $153.09 |2007|
    |             One Day|Romance|         Independent|            54|  3.682733333|             37|       $55.24 |2011|
    |     New Year's Eve|Romance|        Warner Bros.|            48|  2.536428571|              8|      $142.04 |2011|
    |    Music and Lyrics|Romance|        Warner Bros.|            70|   3.64741055|             63|      $145.90 |2007|
    |         Monte Carlo|Romance|    20th Century Fox|            50|       1.9832|             38|       $39.66 |2011|
    |           Jane Eyre|Romance|           Universal|            77|          0.0|             85|       $30.15 |2011|
    +--------------------+-------+--------------------+--------------+-------------+---------------+--------------+----+

drama=spark.sql("SELECT * FROM MOVIETABLE WHERE Genre='Drama'")
drama_rdd=drama.rdd
drama.filter(drama.Year>2009).collect()

    [Row(Film='Water For Elephants', Genre='Drama', LeadStudio='20th Century Fox', Audiencescore%=72,
    Profitability=3.081421053, RottenTomatoes%=60, WorldwideGross='$117.09 ', Year=2011),
     Row(Film='Remember Me', Genre='Drama', LeadStudio='Summit', Audiencescore%=70, Profitability=3.49125,
    RottenTomatoes%=28, WorldwideGross='$55.86 ', Year=2010),
     Row(Film='My Week with Marilyn', Genre='Drama', LeadStudio='The Weinstein Company', Audiencescore%=84,
    Profitability=0.8258, RottenTomatoes%=83, WorldwideGross='$8.26 ', Year=2011),
     Row(Film='Dear John', Genre='Drama', LeadStudio='Sony', Audiencescore%=66, Profitability=4.5988,
    RottenTomatoes%=29, WorldwideGross='$114.97 ', Year=2010),
     Row(Film='A Dangerous Method', Genre='Drama', LeadStudio='Independent', Audiencescore%=89, Profitability=
    0.44864475, RottenTomatoes%=79, WorldwideGross='$8.97 ', Year=2011)]
```

==

```
rdd.collect()
    [1, 2, 3, 4, 5, 4, 3, 2]

num2=rdd.map(lambda x: x*2)

num2.collect()
```

```
        [2, 4, 6, 8, 10, 8, 6, 4]

names=sc.parallelize(['Bills', 'Mark', 'Brian', 'Moick'])

name2=names.map(lambda x: 'Mr. '+x)

name2.collect()
        ['Mr. Bills', 'Mr. Mark', 'Mr. Brian', 'Mr. Moick']

rdd2=sc.parallelize([1,2,3,4,5])

(rdd2.map(lambda x: range(0,x))).collect()
        [range(0, 1), range(0, 2), range(0, 3), range(0, 4), range(0, 5)]

(rdd2.flatMap(lambda x:range(0,x))).collect()
        [0, 0, 1, 0, 1, 2, 0, 1, 2, 3, 0, 1, 2, 3, 4]

(rdd2.flatMap(lambda x: x*x)).collect()
#this gives an error since flatMap needs a function to evaluate. Instead, map() would work

(rdd2.map(lambda x: x*x)).collect()
        [1, 4, 9, 16, 25]

#to make flatMap() work, it needs a function that returns a set of values
def sq(rdd2): return (rdd2, rdd2*rdd2)
(rdd2.flatMap(lambda x: sq(x))).collect()
        [1, 1, 2, 4, 3, 9, 4, 16, 5, 25]

#Union:
Rdd.union(rdd2).collect()
        [1,2,3,4,5,4,3,2,1,2,3,4,5]

Rdd2=sc.parallelize([1,2,3,4,5,6], 2)

Rdd2.glom().collect()
        [[1,2,3], [4,5,6]]

rdd2.fold(2, lambda a,b: a+b)
        27
        #Logic: [1,2,3] and [4,5,6] are the 2 main partitions. The 2 is added to the sum of each at the start, and then to
        the overall sum: (2+[1+2+3])+(2+[4+5+6])+2
rdd2.fold(3, lambda a,b:a+b)
        30
============================================================================================

from pyspark.sql import functions as f

data=[
   ('phones', 55),
   ('Laptop', 43),
   ('Keyboard', 37),
   ('mouse', 40)
]
schema=['product', 'count']

sampleDF=spark.createDataFrame(data, schema)

display(sampleDF)
        DataFrame[product: string, count: bigint]

#perform Transformations
advDF=(sampleDF
      .withColumn("count", f.col('count')+12)
      .withColumn('count', f.col('count')-4)

     )
Print(advDF.take(2))
        [Row(product='phones', count=63, Row(product='Laptop', count=51)]

advDF.show()
        +--------+-----+
        | product|count|
        +--------+-----+
        |  phones|   63|
        |  Laptop|   51|
        |Keyboard|   45|
        |   mouse|   48|
        +--------+-----+
```

Transformations:

- Narrow Transformations:
  - In these ops, each input partition is used to compute one output partition (1:1)
  - Eg: Map, Filter, union, join with co-partitioned inputs
- Wide Transformations:
  - Multiple input partitions can be used to complute multiple output partions (n:n)
  - Eg: groupByKey2, join with non-co-partitioned inputs