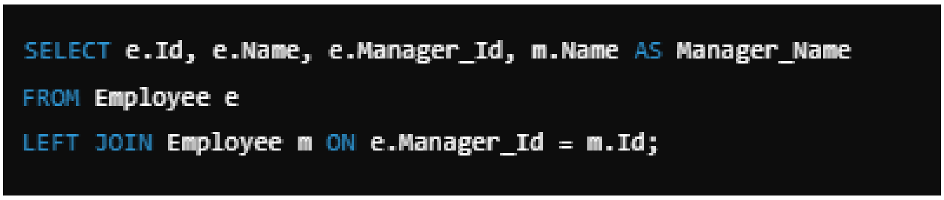
# HashedIn by Deloitte Data Engineer Interview Guide – Experienced 3+

## Technical Round 1

### 1. Add a new column with manager names for each employee

Given a table structure with Id, Name, and Manager_Id, use a **self-join** to find the manager's name.

**SQL Query**:



```sql
SELECT e.Id, e.Name, e.Manager_Id, m.Name AS Manager_Name
FROM Employee e
LEFT JOIN Employee m ON e.Manager_Id = m.Id;
```

### 2. Identify who is a manager and who is not

To find managers (who have employees reporting to them) and non-managers:

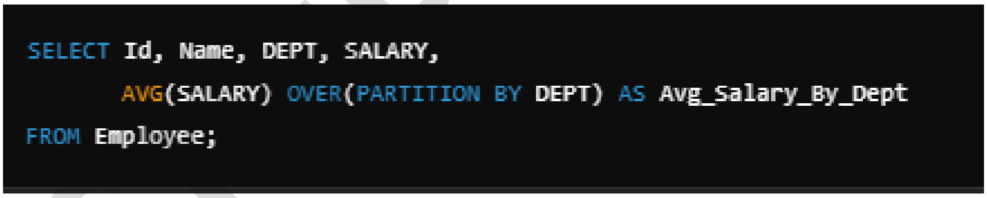
**SQL Query**:

SELECT DISTINCT Manager_Id AS Manager FROM Employee WHERE Manager_Id IS NOT NULL;

SELECT Id FROM Employee WHERE Id NOT IN (SELECT DISTINCT Manager_Id FROM Employee WHERE Manager_Id IS NOT NULL);

### 3. Add a new column with the average salary by department

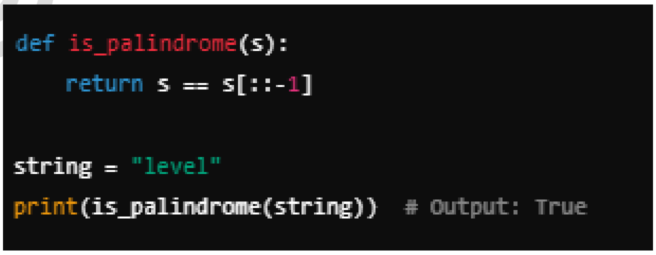
**SQL Query**:



```sql
SELECT Id, Name, DEPT, SALARY,
       AVG(SALARY) OVER(PARTITION BY DEPT) AS Avg_Salary_By_Dept
FROM Employee;
```

### 4. Check if a string is a palindrome

**Python Code**:

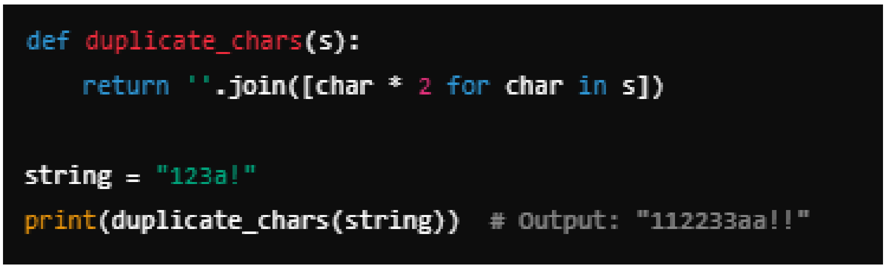


```python
def is_palindrome(s):
    return s == s[::-1]

string = "level"
print(is_palindrome(string))  # Output: True
```

### 5. Duplicate characters in a string ("123a!" → "112233aa!!")

**Python Code**:

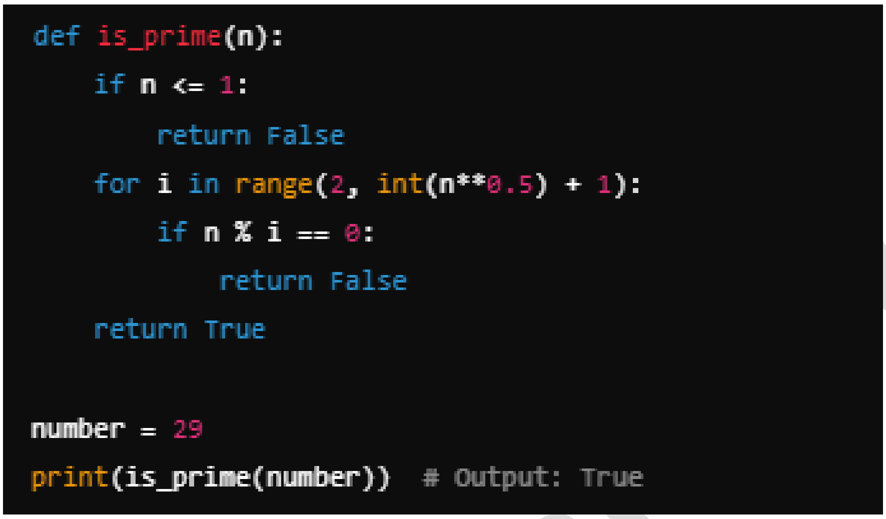


```python
def duplicate_chars(s):
    return ''.join([char * 2 for char in s])

string = "123a!"
print(duplicate_chars(string))  # Output: "112233aa!!"
```

### 6. Check if a number is prime

**Python Code**:



```python
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

number = 29
print(is_prime(number))  # Output: True
```

## Technical Round 2

### 1. Processing 1 TB of data in Spark

Steps to efficiently process 1 TB of data:

- Use partitioning and bucketing to distribute data evenly.
- Leverage columnar file formats like Parquet or ORC.
- Optimize shuffle operations with coalesce and reduceByKey instead of groupByKey.
- Utilize broadcast joins for small lookup tables.
- Enable Adaptive Query Execution (AQE) to dynamically optimize the query plan.
- Use caching only for reused datasets.
- Tune executor memory and number of cores based on the cluster size.

### 2. Designing a Data Warehouse (DWH) Problem Statement

Consider designing a DWH for an e-commerce platform:

- **Fact Tables**:
    - Sales_Fact: Stores transaction details (product_id, customer_id, quantity, revenue, date).
    - Inventory_Fact: Tracks inventory levels by warehouse.
- **Dimension Tables**:
    - Product_Dim: product_id, product_name, category.
    - Customer_Dim: customer_id, name, address.
    - Date_Dim: date_key, day, month, year, quarter.
- **Schema Design**:
    - Use a star schema for easy navigation.
    - Apply partitioning on large tables based on date for efficient querying.
    - Ensure surrogate keys for dimensions.

### 3. How would you design a data pipeline to handle semi-structured and unstructured data?

- Explain the design using distributed file systems (HDFS, S3), data processing frameworks (Spark, Flink), and NoSQL databases (HBase, MongoDB).
- Include stages: Ingestion (Kafka, Kinesis), Data Lake (Parquet/ORC storage), Transformation (Spark DataFrame, PySpark), and Serving Layer (Redshift, Snowflake, Elasticsearch).
- Discuss schema inference and schema-on-read strategies.

**4. Explain the differences between Spark's shuffle and broadcast join. When would you use each?**

- **Shuffle Join**:
  - Used when both datasets are large.
  - Requires shuffling of data across nodes.
  - Slower and resource-intensive.
- **Broadcast Join**:
  - Broadcasts a smaller dataset to each executor.
  - Used when one dataset is small enough to fit in memory.
  - Reduces shuffle and improves performance.

**When to use:**

- Prefer broadcast joins when a small lookup table is involved.
- Use shuffle join for larger datasets, optimizing with partitioning strategies.

**5. What strategies can you use to handle skewed data in Spark?**

- Salting: Add a random key to distribute data evenly across partitions.
- Increase parallelism: Adjust the number of partitions using repartition().
- Broadcast smaller tables: Use broadcast joins.
- Adaptive Query Execution (AQE): Enable Spark’s dynamic partitioning feature.

**6. Explain Adaptive Query Execution (AQE) in Spark 3.x.**

- AQE optimizes queries dynamically at runtime based on the actual data processed.
- **Key Features**:
  - Dynamic partition pruning: Reduces partitions scanned based on filter conditions.
  - Join optimization: Automatically switches join strategies (broadcast, shuffle) during execution.
  - Coalescing shuffle partitions: Reduces the number of shuffle partitions.

**7. How would you optimize a Spark job that takes too long to run in production?**

- **Profiling tools**: Use Spark UI to identify bottlenecks.
- **Techniques**:
  - Cache frequently used DataFrames.
  - Use columnar file formats like Parquet/ORC with predicate pushdown.
  - Avoid using collect() or count() on large datasets.
  - Reduce shuffle operations with efficient transformations (reduceByKey over groupByKey).

**8. Explain how you would implement a Slowly Changing Dimension (SCD) Type 2 in Spark.**

- Use merge operations with Delta Lake or Apache Hudi.
- Steps:
    - Load current data as the target DataFrame.
    - Load incoming data as the source DataFrame.
    - Identify new records, unchanged records, and updated records.
    - Assign start_date, end_date, and current flag to version data.

**9. How do you monitor and debug Spark applications in production?**

- **Spark UI**: Provides DAG visualization and job stages.
- **Ganglia/Prometheus**: Used for resource monitoring.
- **Event logs**: Enable detailed job logging.
- **Metrics**: Monitor executor memory usage, garbage collection, and shuffle read/write.

**10. How do you design a scalable and fault-tolerant data warehouse on a cloud platform?**

- **Storage**: Use S3 or Azure Blob Storage for data lakes.
- **Compute**: Spark on EMR or Databricks for processing.
- **Metadata Management**: Hive Metastore or AWS Glue.
- **Query Engine**: Presto, Redshift, or Synapse Analytics.
- **Fault Tolerance**: Leverage **checkpointing** and **retry policies**.

**Glassdoor HashedIn by Deloitte Review –**

https://www.glassdoor.co.in/Reviews/HashedIn-by-Deloitte-Reviews-E689428.htm

**HashedIn by Deloitte Careers –**

https://hashedin.com/careers/

**Subscribe to my YouTube Channel for Free Data Engineering Content –**

https://www.youtube.com/@shubhamwadekar27

**Connect with me here –**

https://bento.me/shubhamwadekar

**Checkout more Interview Preparation Material on –**

https://topmate.io/shubham_wadekar