# Tech Interview Preparation Guide for Data Integration Engineers

## Overview

**This guide provides a comprehensive framework for candidates preparing for Data Integration Engineer interviews. It emphasizes essential technical skills, tools, methodologies, best practices, and evaluation criteria, along with sample interview questions and additional recommendations.**

---

**Section 1: Core Technical Competencies**

**1.1 Data Integration and Management**

- **Data Integration:**
    - **Design, develop, and maintain scalable data pipelines and ETL processes.**
    - **Integrate data from various sources (e.g., APIs) into a Data Warehouse (DWH) model.**
    - **Ensure integrity and efficiency in data integration processes.**
- **Database Development:**
    - **Proficiency in SQL databases: table creation, logical data models, and optimization.**
    - **Mastery of SQL execution stages and performance tuning (e.g., window functions).**

**1.2 Data Warehousing (DWH)**

- **Data Modeling:**
    - **Dimensional modeling: Star and Snowflake schemas.**
    - **Data Vault, Slowly Changing Dimensions (SCD), and multi-fact models.**
- **Data Architecture:**
    - **Data Lake concepts and modern architectures (Data Mesh, Lakehouse).**
    - **Normalize/denormalize for optimal database design.**

**1.3 Data Transformation**

- **ETL/ELT Methods:**
    - **Incremental loads and masking policies for PII data.**
    - **Mastery of data cleaning, transformation, and aggregation.**

**1.4 Cloud Concepts**

- **Expertise in cloud platforms (GCP, AWS).**
    - **Tools: BigQuery, AWS Glue, Redshift, and Databricks.**
    - **Cloud analytics and scalable DWH solutions.**

---

**Section 2: Relevant Tools and Frameworks**

**2.1 Data Integration Tools**

- **Apache Airflow:**
    - **Build DAGs, manage tasks, and utilize operators, sensors, and Jinja templates.**
- **AWS Glue:**
    - **ETL job development, data format transformation, and catalog management.**
- **GCP BigQuery:**
    - **Partitioning strategies, cost optimization, and data storage advantages.**
- **Databricks:**
    - **Delta Lake, Unity Catalog, and DLT experience.**
- **Others: PySpark, Kafka, Flink, Snowflake, Tableau, and Power BI.**

**2.2 Programming Languages**

- **SQL:**
    - **Advanced SQL queries, window functions, and performance tuning.**
    - **ACID, OLTP/OLAP concepts, and CTEs.**
- **Python:**
    - **Data integration and processing using libraries like Pandas and NumPy.**
    - **REST API integration, file operations, and scalable ETL pipeline design.**

- **PySpark:**
    - **Spark optimization techniques and understanding of Spark's architecture.**

---

**Section 3: Methodologies and Best Practices**

**3.1 Software Engineering Practices**

- **Version Control: Git, GitHub Actions.**
- **CI/CD Pipelines: Jenkins, Docker.**
- **Agile Methodologies: Jira, Confluence.**

**3.2 Data Quality and System Design**

- **Data Quality: Validation techniques, error handling, and recovery mechanisms.**
- **System Design: Scalability, data partitioning, and monitoring strategies.**

---

**Section 4: Interview Preparation**

**4.1 Sample Questions**

- **SQL:**
    - **Write a query to calculate a cumulative count of orders per customer.**
    - **Optimize a query with window functions.**
- **Python:**
    - **Create a function to extract email domains from a JSON file.**
    - **Design an ETL pipeline using Airflow.**
- **Data Warehousing:**
    - **Explain OLTP vs. OLAP.**
    - **Discuss the types of Slowly Changing Dimensions (SCD).**
- **Apache Spark:**
    - **Define lazy evaluation and explain transformations (wide vs. narrow).**

**4.2 Evaluation Criteria**

- **Knowledge Depth: Understanding of core concepts.**
- **Hands-On Experience: Proficiency with relevant tools and platforms.**
- **Problem-Solving Skills: Logical, structured approach to challenges.**
- **Communication: Clear articulation of technical solutions.**
- **Adaptability: Eagerness to learn new technologies.**

---

**Section 5: Additional Recommendations**

**5.1 Complementary Skills**

- **Cloud Certifications: GCP, AWS, or Databricks.**
- **DevOps Knowledge: CI/CD workflows, containerization.**
- **Streaming Technologies: Kafka, Flink.**
- **NoSQL Databases: MongoDB, DynamoDB.**

**5.2 Recommended Resources**

- **Books:**
    - ***The Data Warehouse Toolkit* by Ralph Kimball.**
    - ***Designing Data-Intensive Applications* by Martin Kleppmann.**
    - ***Python for Data Analysis* by Wes McKinney.**
- **Online Courses:**
    - **Coursera: Data Engineering specializations.**
    - **Udemy: SQL, Python, and Airflow.**
    - **Cloud platform-specific certifications.**
- **Technical Documentation:**
    - **Apache Airflow: https://airflow.apache.org/docs/**
    - **Databricks: https://docs.databricks.com/**
    - **dbt: https://docs.getdbt.com/**

---

# Coding tests examples on the interviews

Orders

| cid | order_id | order_date | order_value |
|---|---|---|---|
| A | qwerty | 1-Jan | 10 |
| A | asdfgh | 3-Jan | 20 |
| A | zxcvbn | 10-Jan | 30 |
| B | uiopyy | 2-Jan | 40 |
| B | lkjhgf | 6-Jan | 50 |
| B | mnbvcx | 8-Jan | 60 |
| B | rtyfgh | 10-Jan | 70 |
| C | fghcvb | 1-Feb | 80 |
| C | bnmghj | 1-Feb | 90 |
| C | wersdf | 3-Feb | 100 |
| C | asdzxc | 4-Feb | 110 |

Write SQL for generating below output i.e. cumulative/ running count of orders and cumulative/ running order values for every order per customer.

Expected output

| cid | order_id | order_date | order_value | order_count_history | order_value_history |
|---|---|---|---|---|---|
| A | qwerty | 1-Jan | 10 | 1 | 10 |
| A | asdfgh | 3-Jan | 20 | 2 | 30 |
| A | zxcvbn | 10-Jan | 30 | 3 | 60 |
| B | uiopyy | 2-Jan | 40 | 1 | 40 |
| B | lkjhgf | 6-Jan | 50 | 2 | 90 |
| B | mnbvcx | 8-Jan | 60 | 3 | 150 |
| B | rtyfgh | 10-Jan | 70 | 4 | 220 |
| C | fghcvb | 1-Feb | 80 | 1 | 80 |
| C | bnmghj | 1-Feb | 90 | 2 | 170 |
| C | wersdf | 3-Feb | 100 | 3 | 270 |
| C | asdzxc | 4-Feb | 110 | 4 | 380 |

Employee table:

| id | name | department | managerId |
|---|---|---|---|
| 101 | John | A | null |
| 102 | Dan | A | 101 |
| 103 | James | A | 101 |
| 104 | Amy | A | 101 |
| 105 | Anne | A | 101 |
| 106 | Ron | B | 101 |

Write a solution to find managers with at least five direct reports.

Expected output:

```
+------+
| name |
+------+
| John |
+------+
```

Employee table:

```
+-------+--------+------------+--------+
| empId | name   | supervisor | salary |
+-------+--------+------------+--------+
| 3     | Brad   | null       | 4000   |
| 1     | John   | 3          | 1000   |
| 2     | Dan    | 3          | 2000   |
| 4     | Thomas | 3          | 4000   |
+-------+--------+------------+--------+
```

Bonus table:

```
+-------+-------+
| empId | bonus |
+-------+-------+
| 2     | 500   |
| 4     | 2000  |
+-------+-------+
```

Write a solution to report the name and bonus amount of each employee with a bonus less than 1000.

Expected output:

```
+------+-------+
| name | bonus |
+------+-------+
| Brad | null  |
| John | null  |
| Dan  | 500   |
+------+-------+
```

employees

```
emp_id | emp_name | job_name  | manager_id | start_date  | salary  | commission | dep_id

--------+----------+-----------+------------+------------+---------+------------+--------

 68319 | KAYLING  | PRESIDENT |          | 1991-11-18 | 6000.00 |          |  1001

 66928 | BLAZE    | MANAGER   |     68319 | 1991-05-01 | 2750.00 |          |  3001

 67832 | CLARE    | MANAGER   |     68319 | 1991-06-09 | 2550.00 |          |  1001

 65646 | JONAS    | MANAGER   |     68319 | 1991-04-02 | 2957.00 |          |  2001

 67858 | SCARLET  | ANALYST   |     65646 | 1997-04-19 | 3100.00 |          |  2001

 69062 | FRANK    | ANALYST   |     65646 | 1991-12-03 | 3100.00 |          |  2001
```

63679 | SANDRINE | CLERK | 69062 | 1990-12-18 | 900.00 | | 2001

List emp_name with the same salary in same departments

Write a Python code that can count the words separated by a space and print the result.

The Python code should read one line from standard input and output for each unique word in that line the number of times it occurs (case insensitive) in "word count" format.

The order of output of words can be arbitrary, each unique word must be output only once.

Input example: "apple aPPle banana baNAna TEST s Apple S Test s Te te est"

Expected output:

banana 2

test 2

s 3

te 2

est 1

apple 3

Input - JSON file:

```
{
  "data": [
    {
      "id": 1,
      "name": "John Doe",
      "email": "john.doe@example.com"
    },
    {
```

```
    "id": 2,
    "name": "Jane Smith",
    "email": "jane.smith@exxxample.com"
   },
   {
    "id": 3,
    "name": "Bob Johnson",
    "email": "bob.johnson@example.com"
   },
   {
    "id": 4,
    "name": "Sarah Denth",
    "email": "sarah.denth@dummydata.com"
   }
 ]
}
```


Write a Python code to get the following output:

{"example.com", "exxxample.com", "dummydata.com"}

Courses table:

| student | class |
|---|---|
| A | Math |
| B | English |
| C | Math |
| D | Biology |
| E | Math |

| F | Computer |

| G | Math |

| H | Math |

| I | Math |

+---------+----------+

Output:

+---------+

| class |

+---------+

| Math |

+---------+

Write a solution to find all the classes that have at least five students.

Sales table:

+---------+------------+------+----------+-------+

| sale_id | product_id | year | quantity | price |

+---------+------------+------+----------+-------+

| 1 | 100 | 2008 | 10 | 5000 |

| 2 | 100 | 2009 | 12 | 5000 |

| 7 | 200 | 2011 | 15 | 9000 |

+---------+------------+------+----------+-------+

Product table:

+------------+--------------+

| product_id | product_name |

+------------+--------------+

| 100 | Nokia |

| 200 | Apple |

| 300 | Samsung |

+------------+--------------+

Output:

+------------+------------+----------+-------+

| product_id | first_year | quantity | price |

+------------+------------+----------+-------+

| 100 | 2008 | 10 | 5000 |

| 200 | 2011 | 15 | 9000 |

+------------+------------+----------+-------+

Write a solution to select the product id, year, quantity, and price for the first year of every product sold.

Return the resulting table order by product_id.

Input:

+------------+---------+-----+

| student_id | name | age |

+------------+---------+-----+

| 101 | Ulysses | 13 |

| 53 | William | 10 |

| 128 | Henry | 6 |

| 3 | Henry | 11 |

+------------+---------+-----+

Output:

+---------+-----+

| name | age |

+---------+-----+

| Ulysses | 13 |

+---------+-----+

Write a solution to select the name and age of the student with student_id = 101

## Basic Level

sql Copy

```sql
-- Find employees with highest salary in each department
SELECT d.name AS Department, e.name AS Employee, e.salary AS Salary
FROM Employee e
JOIN Department d ON e.departmentId = d.id
WHERE (e.departmentId, e.salary) IN (
    SELECT departmentId, MAX(salary)
    FROM Employee
    GROUP BY departmentId
);
```

## Intermediate Level

sql Copy

```sql
-- Calculate running totals with window functions
SELECT
    cid,
    order_id,
    order_date,
    order_value,
    COUNT(*) OVER (PARTITION BY cid ORDER BY order_date) as order_count_history,
    SUM(order_value) OVER (PARTITION BY cid ORDER BY order_date) as order_value_history
FROM Orders
ORDER BY cid, order_date;
```

## Advanced Level

sql Copy

```sql
-- Handle complex hierarchical data
WITH RECURSIVE EmployeeHierarchy AS (
    SELECT id, name, manager_id, 1 as level
    FROM Employees
    WHERE manager_id IS NULL
    UNION ALL
    SELECT e.id, e.name, e.manager_id, eh.level + 1
    FROM Employees e
    JOIN EmployeeHierarchy eh ON e.manager_id = eh.id
)
SELECT * FROM EmployeeHierarchy;
```

## Data Transformation

python Copy

```python
def transform_data(df):
    """
    Sample data transformation function
    """
    return (df
        .groupby('customer_id')
        .agg({
            'order_value': ['sum', 'mean', 'count'],
            'order_date': ['min', 'max']
        })
        .reset_index()
        .pipe(lambda x: x.set_axis([
            'customer_id', 'total_value', 'avg_value',
            'order_count', 'first_order', 'last_order'
        ], axis=1))
    )
```

## API Integration

python Copy

```python
def fetch_paginated_data(base_url, params=None):
    """
    Handle API pagination
    """
    all_data = []
    page = 1

    while True:
        try:
            response = requests.get(
                base_url,
                params={**(params or {}), 'page': page}
            )
            response.raise_for_status()
            data = response.json()

            if not data:
                break

            all_data.extend(data)
            page += 1

        except requests.exceptions.RequestException as e:
            logger.error(f"Error fetching data: {e}")
            raise

    return all_data
```