📊 SQL Interview Preparation Answers

---

1. What is SQL and why is it important in data engineering?
SQL (Structured Query Language) is a domain-specific language used for managing and manipulating relational databases. It allows data engineers to create, retrieve, update, and delete data efficiently. It is crucial for data engineering as it supports tasks like data extraction, transformation, loading (ETL), data analysis, and building data pipelines.

---

2. What is the difference between INNER JOIN, LEFT JOIN, and RIGHT JOIN?

- **INNER JOIN**: Returns records that have matching values in both tables.
- **LEFT JOIN**: Returns all records from the left table and matched records from the right table. Unmatched records from the right are shown as NULL.
- **RIGHT JOIN**: Returns all records from the right table and matched records from the left table. Unmatched records from the left are shown as NULL.

---

3. What are primary keys and foreign keys?

- **Primary Key**: Uniquely identifies each row in a table. Cannot be NULL.
- **Foreign Key**: A field in one table that references the primary key of another table. It creates a relationship between the two tables.

---

4. How do you use GROUP BY and HAVING clauses?

- GROUP BY groups rows with the same values into summary rows (e.g., to count, sum, etc.).
- HAVING is used to filter records after grouping (unlike WHERE, which filters before grouping).

Example:

```
SELECT department, COUNT(*)
FROM employees
GROUP BY department
```

```
HAVING COUNT(*) > 5;
```

---

5. What is the difference between WHERE and HAVING?

- WHERE filters rows before grouping.
- HAVING filters groups after the GROUP BY clause is applied.

---

6. What is the difference between DELETE, TRUNCATE, and DROP?

- DELETE: Removes rows based on condition, can be rolled back.
- TRUNCATE: Removes all rows, faster, cannot be rolled back in some systems.
- DROP: Deletes the entire table structure and data.

---

7. What is the difference between UNION and UNION ALL?

- UNION: Combines results of two queries and removes duplicates.
- UNION ALL: Combines results and includes duplicates.

---

8. How do you optimize a slow-running SQL query?

- Use proper indexing.
- Avoid SELECT *.
- Use EXPLAIN to analyze query plans.
- Limit the use of subqueries.
- Use JOINs efficiently.
- Partition large tables.

---

9. Write a query to calculate the cumulative sum of sales by month.

```
SELECT month,
    SUM(sales) OVER (ORDER BY month) AS cumulative_sales
```

FROM sales_data;

This uses a window function to compute cumulative sums.

---

10. What are indexes, and how do they improve performance?
Indexes are data structures that improve the speed of data retrieval operations. They work like a table of contents and help the database engine find data faster. However, indexes increase write time and storage usage.

---

11. How would you handle NULL values in SQL queries?

- Use IS NULL or IS NOT NULL to filter.
- Use COALESCE() or IFNULL() to replace NULLs.
- Use conditional logic (CASE) to handle NULLs.

---

12. What is a stored procedure, and when would you use one?
A stored procedure is a precompiled set of SQL statements stored in the database. It is used for reusability, performance improvement, and maintaining logic in one place. Useful for complex business logic and batch processing.

---

13. What are ACID properties in database systems?

- **Atomicity**: Transactions are all-or-nothing.
- **Consistency**: Data must be valid before and after the transaction.
- **Isolation**: Transactions do not interfere with each other.
- **Durability**: Once committed, changes are permanent.

---

14. What is a subquery and when would you use one?
A subquery is a query nested inside another query. It is used when the result of one query is needed to filter or compute values in another query.

Example:

```
SELECT name FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

---

15. Can you describe what a CTE (Common Table Expression) is?
A CTE is a temporary result set defined with WITH that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. It improves readability and modularity.

Example:

```
WITH SalesCTE AS (
  SELECT salesperson_id, SUM(sales) AS total_sales
  FROM sales_data
  GROUP BY salesperson_id
)
SELECT * FROM SalesCTE WHERE total_sales > 10000;
```

---

16. How would you write a query to find duplicate records in a table?

```
SELECT column1, COUNT(*)
FROM table_name
GROUP BY column1
HAVING COUNT(*) > 1;
```

This identifies values in column1 that appear more than once.

---

17. How do you find the Nth highest salary in an employees table?

```
SELECT DISTINCT salary
FROM employees e1
WHERE N - 1 = (
  SELECT COUNT(DISTINCT salary)
  FROM employees e2
```

```
  WHERE e2.salary > e1.salary
);
```

Replace N with the rank (e.g., 2 for second highest).

☁️ **Dell Boomi**

1. What is Dell Boomi and what are its primary use cases?

Dell Boomi is a **cloud-based Integration Platform as a Service (iPaaS)** that enables businesses to connect applications, data, and systems seamlessly. It supports integration between cloud applications, on-premises systems, and hybrid environments.

Primary Use Cases:

- Application-to-Application (A2A) integration (e.g., Salesforce to SAP)
- B2B/EDI integrations (e.g., partners exchanging data)
- Data migration and synchronization
- Master Data Management (MDM)
- API management and publishing
- Workflow automation

---

2. What are the core components of the platform (Atoms, Molecules, Atom Clouds)?

- **Atom:** A lightweight, runtime engine that executes integration processes locally or in the cloud.
- **Molecule:** A cluster of Atoms used for **high availability** and **load balancing** in large enterprises.
- **Atom Cloud:** A cloud-hosted environment where Atoms run, maintained by Boomi or organizations.

These components help determine **where** and **how** your integration logic runs.

---

3. What is a Boomi process?

A Boomi process is a **workflow or set of steps** that define how data moves between systems. It typically includes:

- Start shape (trigger or source)
- Connectors (to systems like databases or apps)
- Data transformations (e.g., mapping fields)

- Business logic (e.g., decisions, loops)
- End shape (destination or final output)

---

4. How does Boomi differ from traditional middleware?

Traditional Middleware	Dell Boomi
Requires heavy infrastructure	Cloud-native, no on-premise setup needed
Complex and time-consuming to scale	Scalable on-demand
Manual deployment	Automated deployment and version control
Siloed integration logic	Centralized, visual process builder

Boomi is low-code, faster to implement, and integrates easily with modern cloud applications.

---

5. What is an Atom in Boomi?

An **Atom** is the **runtime engine** that executes Boomi integration processes. It contains all the components (libraries, connectors, logic) needed to:

- Connect to systems
- Transfer and transform data
- Handle errors

Atoms can be deployed **on-premises**, in a **cloud server**, or in a **private data center**.

---

6. What are connectors in Boomi? Provide examples.

Connectors are pre-built components that allow Boomi to **connect to external applications and data sources**.

Examples:

- Salesforce Connector (connects to Salesforce APIs)
- HTTP Client Connector (RESTful APIs)

- FTP/SFTP Connector (for file transfers)
- Database Connector (e.g., MySQL, SQL Server)
- NetSuite, SAP, Workday, Oracle connectors

Each connector has two parts: **operation** (action like get/post) and **connection** (authentication/config).

---

7. How do you perform error handling in a Boomi process?

Boomi provides multiple ways to handle errors:

- **Try/Catch shapes** to capture and handle exceptions
- **Process reporting and alerts**
- **Custom logging** for detailed debugging
- Use of **Decision shapes** to handle conditional errors
- **Automatic retries** and error queues for temporary failures
- Integration with **email or ticketing systems** for alerts

---

8. Can Boomi roll back to a previous version?

Yes, Boomi offers **version control**. You can:

- **Track changes** with each deployment
- **Revert** to a previous version of a process
- **Deploy old versions** as needed
  This ensures safer updates and rollback in case of bugs or issues.

---

9. Where can you host Boomi Atoms?

Boomi Atoms can be hosted in multiple environments:

- **On-premise servers**
- **Cloud platforms** (e.g., AWS, Azure, GCP)
- **Dell Boomi Atom Cloud**

- **Private cloud**
  The choice depends on network architecture, security, and performance needs.

---

10. How does cloud computing benefit from Boomi (Atmosphere)?

Boomi Atmosphere is cloud-native, offering:

- **Rapid deployment** without managing infrastructure
- **Scalability** for growing workloads
- **Multi-tenancy** to serve multiple clients securely
- **Central monitoring** and **automated updates**
  It simplifies complex integrations across cloud ecosystems.

---

11. What is ETL and how does Boomi support it?

ETL (Extract, Transform, Load) refers to:

- **Extracting** data from source systems
- **Transforming** it into the required format
- **Loading** it into a destination (e.g., database)

Boomi supports ETL through:

- **Connectors** for data sources/destinations
- **Mapping shapes** for transformations
- **Process orchestration** for sequencing

---

12. What data formats does Boomi support (e.g., JSON, XML)?

Boomi supports a wide range of data formats:

- **JSON**
- **XML**
- **CSV**
- **Flat files (TXT)**

- **EDI (X12, EDIFACT)**
- **Databases**
- **Proprietary formats via scripting/custom connectors**

---

13. How would you handle data transformation between different formats (e.g., CSV to JSON)?

Steps to handle this in Boomi:

1. **Read CSV** using a connector (e.g., FTP or file)
2. Use a **Map shape** to map fields from CSV to JSON structure
3. Define the **destination profile** as JSON
4. Use **functions** or scripting if needed for format adjustments
5. Output the result via a connector (e.g., HTTP, API)

---

14. How does Boomi ensure data security during integration?

Boomi provides:

- **Data encryption** (at rest and in transit)
- **Secure agents (Atoms)** with controlled access
- **Token-based authentication (OAuth, API keys)**
- **Role-based access control (RBAC)**
- **Audit logs** and **data masking**
- **Compliance** with GDPR, HIPAA, SOC 2, etc.

---

15. What are some common scenarios where Boomi would be used over custom-coded integrations?

- Fast integration between SaaS apps (e.g., Salesforce ↔ NetSuite)
- Connecting ERP to CRM with minimal coding
- Real-time or scheduled data sync

- Replacing fragile Excel/email-based workflows
- Organizations lacking large development teams

Boomi saves time, reduces errors, and is easier to maintain than hand-coded integrations.

---

16. What challenges might you face when integrating data from multiple sources?

- **Data inconsistency** (formats, structures)
- **Latency** in real-time sync
- **Security and compliance**
- **API limitations** or failures
- **Version mismatches**
- **Error handling across systems**
- **Network/firewall issues**

Boomi helps mitigate many of these with pre-built connectors, transformation tools, and logging.

---

17. What is data mapping and why is it important?

Data mapping is the process of defining how fields from a source structure correspond to a destination structure.

Importance:

- Ensures accurate data transformation
- Enables format conversions (e.g., JSON → XML)
- Helps apply business rules during transformation
- Essential for integrations, reporting, and ETL

---

18. How do you troubleshoot a failed Boomi process?

1. **Check process logs** in Boomi Dashboard
2. Review **execution details** and **error messages**

3. Use **Try/Catch shapes** to isolate faults
4. Use **Process Reporting** to see trends
5. Use **test mode** to simulate data
6. Validate **connections, mappings, credentials**
7. Check **system APIs or third-party logs**

---

19. Have you worked with any other integration tools? How does Boomi compare?

(If you have experience with MuleSoft, Informatica, or Talend, you can compare. Otherwise, say this:)

“I’ve reviewed other tools like MuleSoft and Informatica. Compared to them, I find Boomi to be faster to deploy, more user-friendly due to its visual drag-and-drop interface, and better suited for hybrid cloud/on-premise setups. It also simplifies versioning and process reuse.”

🔧 Postman & APIs

Below is a compact but **interview-ready guide** that hits the key talking points for each question. Skim it for last-minute revision or expand on any section as you practise out loud.

---

1 | What is Postman and why is it used?

Postman is a **desktop/browser tool for building, sending, and automating HTTP-based API calls**. It helps developers, testers, and DevOps teams:

- **Design & document** REST/SOAP/GraphQL APIs
- **Send requests** and inspect responses quickly
- **Save examples & share collections** for collaboration
- **Automate tests** and integrate them in CI/CD pipelines (Newman CLI)

---

2 | How do you send GET and POST requests in Postman?

1. **Create a request tab** → choose **GET** or **POST** from the dropdown.
2. **Enter URL** (e.g., https://api.example.com/items).
3. Optional: add **query params** (for GET) via Params tab or send a **body** (for POST) via Body → raw/form-data/JSON.
4. Click **Send** and review the response pane (status, headers, body, time, size).

---

3 | Common HTTP methods and typical use-cases

Method	Typical Use
GET	Retrieve resources (safe, idempotent)
POST	Create a new resource or submit data that causes a change
PUT	Replace an entire resource
PATCH	Partially update a resource
DELETE	Remove a resource

Method	Typical Use
HEAD	Same as GET but returns headers only
OPTIONS	Discover supported methods / CORS pre-flight
TRACE	Diagnostic loop-back (rarely used)

---

4 | Status codes 200, 404, 500

- **200 OK** – Successful request, body may contain data.
- **404 Not Found** – Client asked for a non-existent endpoint or ID.
- **500 Internal Server Error** – Unhandled exception on the server side; nothing inherently wrong with the request.

---

5 | REST vs SOAP APIs

Feature	REST	SOAP
Protocol	Any (mostly HTTP/HTTPS)	Strictly XML over multiple protocols
Message format	JSON, XML, text	XML (envelope, header, body)
Style	Resource-oriented, stateless	Operation-oriented
Payload size	Lightweight	Verbose
Standards	Loose (HTTP verbs + URIs)	Strong (WS-Security, WS-Addressing)
Use cases	Web/mobile, microservices	Enterprise B2B, legacy ERP, formal contracts

---

6 | Setting headers & parameters in Postman

- **Params tab**: key/value pairs are auto-appended as query string.
- **Headers tab**: set custom headers like Content-Type, Authorization.

- **Bulk edit** or **preset** headers to reuse across requests.

---

7 | Extracting a value from JSON in Postman

In the **Tests** tab (runs *after* the response):

```
let json = pm.response.json();

let token = json.access_token;

pm.environment.set('authToken', token);  // saves into env var
```

---

8 | Pre-request script

JavaScript executed **before** Postman sends the request. Typical tasks:

- Dynamic **timestamp/nonces**
- **HMAC** signature calc
- Pulling **variables** from previous responses

---

9 | Managing environment variables

1. Click **Environments** icon → **Add** new env (e.g., Dev, QA, Prod).
2. Define key/value pairs (baseUrl, apiKey).
3. Select the env from the dropdown in the top-right.
4. Reference with {{variableName}} in URLs, headers, tests.

---

10 | Postman Collection Runner

A built-in runner that executes an entire **collection** (or folder) of requests in sequence, optionally:

- Iterating over **CSV/JSON data files** (data-driven testing)
- Logging results and **test assertions**
- Exporting a summary report (for CI use Newman CLI).

---

11 | Chaining requests

- Store data (IDs, tokens) from **Test script** of request A into env variables.
- Use those variables in request B's URL, headers, or body.
- Example flow: **Auth → save token → use token in subsequent calls**.

---

12 | Manual vs automatic API testing

Aspect	Manual (Postman UI)	Automated (Collection + Newman)
Speed	Interactive, ad-hoc	Batch, CI/CD
Scope	Exploratory, debugging	Regression, smoke
Runner	You click Send	CLI, pipeline
Code	Little/none	JS tests, build scripts

---

13 | Types of API testing

- **Contract/schema**: Does response match OpenAPI/Swagger definition?
- **Unit**: Test a single endpoint in isolation (often in code, e.g., JUnit).
- **Integration**: End-to-end flow across services.
- **Performance/load**: TPS, latency (e.g., JMeter, k6).
- **Security**: AuthZ, injection, fuzzing.
- **Reliability/chaos**: How APIs behave under failures.

---

14 | Debugging a 500 Internal Server Error

1. Confirm request is valid (URL, headers, body).
2. Check **server logs** if accessible.
3. Reduce payload to minimal repro.

4. Try same request in a different environment or with curl.
5. Validate server-side dependencies (DB, downstream services).
6. Capture **Correlation-ID** if provided and share with back-end team.

---

15 | API rate limits & handling them

- Limits throttle requests (e.g., 1000 req/min per API key).
- Servers return **429 Too Many Requests** with Retry-After.
- Use **exponential back-off**, client-side caching, or batch endpoints.
- Monitor headers like X-RateLimit-Remaining.

---

16 | Parsing nested JSON

```
let userCity = pm.response.json().data.user.address.city;

// Handle arrays:

let firstItemId = pm.response.json().items[0].id;
```

Postman's built-in **lodash**: _.get(obj, 'path.to.prop').

---

17 | Alternative API-testing tools

- **curl / HTTPie** (quick CLI calls)
- **Insomnia** (GUI similar to Postman)
- **Swagger UI / ReDoc** (interactive docs)
- **JMeter / k6 / Gatling** (load tests)
- **SoapUI** (SOAP & REST functional tests)
- **Pact / Schemathesis** (contract testing)

---

18 | Handling authentication

- **API keys** in headers/params

- **Basic Auth** (username:password → Base64)
- **Bearer/OAuth 2.0** token: Use Postman's **Auth** tab to fetch/refresh tokens automatically.
- **HMAC/signature**: compute in pre-request scripts.
- **Cookie/session**: capture cookie in earlier response, send in subsequent requests.

---

19 | Typical REST API response structure

```
{
 "status": "success",
 "data": {
  "id": 123,
  "name": "Widget",
  "price": 9.99
 },
 "meta": {
  "timestamp": "2025-07-01T16:50:00Z",
  "requestId": "abc-123"
 },
 "errors": []      // only present on failure
}
```

- **Status**: high-level success/failure.
- **Data**: actual resource(s).
- **Meta**: paging, links, debug info.
- **Errors**: message, code, details when status ≠ 2xx.

🧼 Data Cleaning & Analysis

Here are **clear, interview-focused answers** to each question about data cleaning and preprocessing, with examples where helpful. These will help you confidently explain your understanding during the interview:

---

1. What steps would you take to clean raw data (duplicates, nulls, formats)?

1. **Remove duplicates** using .drop_duplicates() in Python or DISTINCT in SQL.
2. **Handle nulls** by removing or imputing values.
3. **Fix data types** (e.g., converting strings to dates or floats).
4. **Standardize formats** (e.g., upper/lowercase text, consistent date format).
5. **Trim whitespace** and clean special characters.
6. **Check for outliers or invalid entries**.
7. **Validate against known rules or reference data**.

---

2. How would you handle missing or incomplete data in a dataset?

Options depend on context:

- **Drop rows or columns** if missing values are excessive.
- **Impute values** using:
    - Mean/median (for numeric)
    - Mode (for categorical)
    - Forward/backward fill (for time series)
- **Use domain-specific defaults** (e.g., "Unknown", 0)
- **Flag missing values** with a new column (for modeling)

---

3. How do you find and remove duplicate records?

- In **Pandas (Python)**:

- df.drop_duplicates(inplace=True)
- In **SQL**:
- SELECT DISTINCT * FROM table;

To find them:

df[df.duplicated()]

Use subset parameter to target specific columns.

---

4. How would you standardize inconsistent date or text formats?

- Convert date formats using:
- pd.to_datetime(df['date_column'], errors='coerce')
- Standardize text:
- df['column'] = df['column'].str.strip().str.lower().replace({...})
- Use regex for cleaning or splitting complex formats.

---

5. How do you validate the accuracy of a cleaned dataset?

- **Cross-check** with source data or known constraints.
- Use **summary statistics** (mean, count, ranges) before/after cleaning.
- **Check distributions** for abnormalities.
- Validate keys and relationships (foreign keys, referential integrity).
- Use **data profiling tools** like pandas-profiling.

---

6. What tools or languages do you use for data cleaning (SQL, Python)?

- **Python** (Pandas, NumPy, Pyjanitor)
- **SQL** for large structured data (filtering, joins, aggregations)
- **OpenRefine** or Excel for manual inspection
- Sometimes **Power Query** in Excel/Power BI

---

7. How to handle outliers or incorrect data entries?

- **Visualize** using box plots, histograms, scatter plots.
- Apply rules:
    - Z-score or IQR method
    - Business logic (e.g., salary can't be negative)
- Options:
    - **Remove** if errors
    - **Cap (winsorize)** to limits
    - **Impute** or **investigate** for correction

---

8. What would you do with inconsistent categorical entries?

- **Standardize** with mapping or string operations:
- df['city'] = df['city'].str.lower().replace({'nyc': 'new york', 'n.y.': 'new york'})
- Use **category types** in Pandas for efficiency.
- Perform **grouping** or **clustering** if needed.

---

9. How would you document data cleaning steps?

- Maintain a **data cleaning log** or notebook.
- Add **comments in code** explaining transformations.
- Use **version control** (e.g., Git).
- Write a **data dictionary** or **summary report** describing:
    - Original issues
    - Cleaning steps
    - Assumptions made
    - Final structure

---

10. How do you ensure cleaned data remains reproducible?

- **Use code** (scripts, Jupyter Notebooks, SQL scripts)
- **Version data** and scripts
- Store **raw and processed copies**
- Avoid manual steps; automate pipelines with tools like **Airflow** or **Prefect**
- Document assumptions and changes

---

11. What is data normalization and why is it important?

- In databases: **structuring data to reduce redundancy** (1NF, 2NF, 3NF).
- In analytics: **scaling numeric features** (e.g., MinMax, Z-score) to a uniform range.
- **Importance:**
    - Improves data quality and consistency
    - Makes data suitable for modeling
    - Avoids bias from large numeric ranges

---

12. How do you merge two datasets with mismatched column names?

In **Pandas**:

```
df1.merge(df2, left_on='cust_id', right_on='customer_id', how='inner')
```

In **SQL**:

```
SELECT * FROM table1 t1
JOIN table2 t2 ON t1.cust_id = t2.customer_id;
```

Handle mismatches by:

- Renaming columns
- Changing data types to match

---

13. What are common data quality issues and how do you address them?

- **Missing values** → Impute or drop
- **Duplicates** → Use drop_duplicates()
- **Inconsistent formats** → Normalize (e.g., dates, units)
- **Outliers** → Investigate, cap, or remove
- **Invalid data types** → Use astype() or conversion functions
- **Typos** in categorical data → Standardize or use fuzzy matching
- **Inaccurate joins** → Validate keys and record counts

---

14. How do you handle date/time formatting inconsistencies?

- Use pd.to_datetime() in Python to parse various formats.
- Specify format= for strict parsing if needed.
- Convert all dates to **ISO 8601** (YYYY-MM-DD) for consistency.
- Handle **timezone awareness** with .dt.tz_convert() if applicable.

---

15. Explain the difference between structured and unstructured data.

Type	Structured	Unstructured
Format	Tabular (rows/columns)	Free-form (text, images, video)
Storage	Databases, spreadsheets	Files, NoSQL, object stores
Examples	SQL databases, Excel	Emails, PDFs, social media posts
Processing	SQL, Pandas	NLP, OCR, audio/image processing

Structured data is easier to clean and analyze. Unstructured data requires more preprocessing.

📦 Data Warehousing & Pipelines

1. What is a data warehouse and why is it used?

A **data warehouse** is a **centralized system** designed for **storing, aggregating, and analyzing historical data** from multiple sources. It supports **business intelligence, reporting, and decision-making**.

Used for:

- Long-term storage of cleaned, structured data
- Running complex queries without affecting operational systems
- Trend analysis, forecasting, dashboards

---

2. What is the difference between OLTP and OLAP?

OLTP	OLAP
Online Transaction Processing	Online Analytical Processing
Used for day-to-day operations	Used for analysis & reporting
INSERTs, UPDATEs, DELETEs	Complex SELECT queries
Real-time, high-speed transactions	Aggregated, historical data
Highly normalized	Often denormalized

---

3. What is a fact table and a dimension table?

- **Fact Table**: Contains **measurable, quantitative data** (e.g., sales, revenue). Has foreign keys to dimension tables.
- **Dimension Table**: Stores **descriptive attributes** (e.g., customer name, product category) that provide context for facts.

Example:

- **Fact_Sales**: sale_id, date_id, product_id, quantity, amount
- **Dim_Product**: product_id, name, category

---

4. What is a star schema vs. snowflake schema?

Star Schema	Snowflake Schema
Dimension tables are denormalized	Dimension tables are normalized
Faster for querying	More space-efficient
Simple structure	More complex joins
Best for performance	Best for storage optimization

---

5. Why use stored procedures in a data warehouse?

Stored procedures:

- Automate **ETL tasks** (e.g., data cleaning, loading)
- Ensure **data consistency and reusability**
- Improve **performance** through precompiled logic
- Centralize business logic for easier maintenance

---

6. What is ETL and how does it relate to warehousing?

ETL stands for:

- **Extract**: Pull data from source systems
- **Transform**: Cleanse, standardize, and reshape the data
- **Load**: Insert into the data warehouse

ETL is **the backbone of data warehousing**, moving raw operational data into an analyzable format.

---

7. How would you design an ETL pipeline for daily sales data?

Steps:

1. **Extract** sales data from POS or ERP systems (daily dump or API)

2. **Transform**: Handle missing data, convert currencies, map product IDs to names
3. **Load** into a **fact_sales** table, update dimensions if needed
4. Schedule with **cron/Airflow** and include **logging & error handling**
5. Maintain **audit logs** for troubleshooting

---

8. What is a slowly changing dimension (SCD)?

An SCD tracks **changes in dimension data over time**.

Types:

- **Type 1**: Overwrite old data (e.g., correcting a typo)
- **Type 2**: Add a new row with versioning (track history)
- **Type 3**: Add new columns for current and previous values

---

9. What is schema normalization vs denormalization?

Normalization	**Denormalization**
Removes redundancy	Allows redundancy
More joins, less storage	Fewer joins, faster read
Used in OLTP	Used in OLAP (warehousing)
1NF, 2NF, 3NF...	Flattened structures

In warehouses, **denormalization is preferred** for **faster reporting**.

---

10. What are indexed views or materialized views?

- **Indexed Views (SQL Server)**: Views with persistent indexes; improve performance.
- **Materialized Views (Oracle, Snowflake, Redshift)**: Precomputed, stored query results that refresh periodically.

They **speed up complex queries** and **reduce processing time**.

---

11. How would you optimize query performance in a data warehouse?

- Use **partitioning** on large fact tables
- Create **appropriate indexes**
- Use **materialized views**
- **Denormalize** frequently joined tables
- Optimize **ETL transformations**
- Avoid SELECT *

---

12. What are the common challenges in data warehousing?

- **Data quality** issues from inconsistent sources
- **Slow load times** for large datasets
- Handling **SCDs and changing business logic**
- Managing **storage costs**
- Ensuring **real-time or near real-time updates**
- Data **security and access control**

---

13. What is the difference between incremental and full data loads?

Full Load	**Incremental Load**
Reloads entire dataset	Loads only new or changed data
Simpler logic	Faster and more efficient
More resource-intensive	Needs tracking columns (e.g., last_updated)
Risk of duplication if not handled carefully	Requires delta detection

---

14. How do you ensure data quality in a data warehouse?

- **Data validation rules** in ETL

- **Profiling tools** (e.g., Great Expectations, Talend)
- Implement **checksums or row counts**
- Maintain **audit tables**
- Use **reference tables** for valid lookups
- Regular **QA/testing** before reporting

---

15. Have you worked with any cloud data warehouses (e.g., Snowflake, Redshift)?

(If yes, answer based on your experience. If not, say this:)

"I've explored cloud data warehouses conceptually and understand how they differ from traditional systems. I know tools like **Snowflake** allow elastic scaling and **separate compute from storage**, while **Amazon Redshift** is based on PostgreSQL and supports **parallel processing** for fast performance. I am comfortable learning and adapting to cloud platforms."

🔗 Data Integration

1. What is data integration?

Data integration is the process of **combining data** from multiple sources into a single, unified view. It's used to:

- Consolidate data from APIs, databases, files, or apps
- Power dashboards, analytics, and machine learning
- Enable real-time business decisions

---

2. How would you integrate data from an API into a database?

Steps:

1. **Connect to the API** using tools like Python (requests), Postman, or an integration platform.
2. **Extract data** (usually in JSON/XML).
3. **Transform data** into the schema required by the database (e.g., flatten JSON, rename fields).
4. **Load into DB** using SQL INSERT or bulk loaders like Pandas .to_sql() or ETL tools like Boomi or SSIS.
5. Schedule the job or trigger it via event/webhook.
6. Add **error handling and logging**.

---

3. What is real-time vs. batch integration?

Real-Time	Batch
Data flows immediately	Data flows at set intervals
Event-driven or API/webhook	Scheduled jobs or file transfers
Used in live apps, fraud detection	Used in reporting, data warehousing
Complex and resource-intensive	Simpler to manage and scale

---

4. What tools do you know for data integration (Boomi, SSIS)?

- **Dell Boomi** – Cloud-native iPaaS with low-code interface
- **SSIS (SQL Server Integration Services)** – On-premise ETL for Microsoft stack
- **Apache Nifi / Airbyte / Talend** – Open-source ETL tools
- **Apache Airflow** – Workflow orchestration for data pipelines
- **Python scripts with Pandas/requests/sqlalchemy**
- **Fivetran / Stitch** – Managed connectors for SaaS apps
- **Informatica, MuleSoft, Zapier** – For enterprise or light automation use

---

5. How do you ensure data accuracy in integration pipelines?

- **Validate input** data types, nulls, formats
- Use **checksums** and **row counts** to verify loads
- Apply **data profiling** and quality checks
- Add **logging** and **alerts** for unusual values
- Build **unit tests** and **sample data comparisons**
- Use **transactional control** in DB (rollback on error)

---

6. What are common integration challenges and how would you solve them?

Challenge	Solution
Schema mismatch	Use transformation/mapping layers
API rate limits	Add throttling, retries, batch requests
Network failures	Implement retry logic, alerts
Duplicate data	Use deduplication logic or upserts
Inconsistent formats	Apply format validation and standardization

Challenge	Solution
Security/auth issues	Use secure token handling, OAuth2, encrypted storage

---

7. How do you monitor a data pipeline?

- Use platform tools (Boomi monitoring, SSIS logs)
- Set up **email/slack alerts** for failures
- Log process start/end, record counts, errors
- Use **Dashboards** (e.g., Grafana, Kibana) for metrics
- Track **latency, data freshness, throughput**

---

8. What is data mapping and transformation?

- **Data Mapping**: Linking source fields to target fields.
    - E.g., customer_name → full_name
- **Transformation**: Modifying data to match the target schema or business logic.
    - E.g., currency conversion, date reformatting, merging fields

These ensure that data is clean, consistent, and ready for use in its destination.

---

9. How do you handle schema changes between systems?

- Detect changes with schema comparison tools or column audits
- Use **version control** for schemas and ETL code
- Add **flexibility** in mappings (e.g., dynamic column handling)
- Alert team if breaking change occurs
- Maintain **backward compatibility** (e.g., optional columns)
- In Boomi/SSIS, update the mapping profile and retest

---

10. Describe a simple integration workflow end-to-end.

Example: Daily API to Database integration

1. **Trigger**: Schedule runs every morning
2. **Extract**: Call a weather API to pull JSON data
3. **Transform**: Clean fields, convert timestamps, rename keys
4. **Load**: Write to a PostgreSQL weather_reports table
5. **Log**: Record the number of records inserted
6. **Alert**: Send email on success/failure

Tools: Python + requests + Pandas + SQLAlchemy + cron job

---

11. How do you handle errors or retries in data pipelines?

- Add **Try/Except** blocks or error handling shapes (Boomi)
- Use **retry logic** with exponential backoff
- Log detailed errors with timestamps and trace info
- Store failed records in an **error queue or dead letter file**
- Alert support teams on critical failures
- For repeatability: track **last processed ID/timestamp** to resume

💬 Soft Skills & Scenario-Based

1. How do you prioritize tasks when managing multiple data projects?

I prioritize tasks based on:

- **Deadlines and business impact**
- **Dependencies**—whether others are waiting on my input
- The **complexity or time required**—I may complete smaller tasks quickly to clear bandwidth
- If unclear, I proactively ask my **supervisor to help set priorities**

I also break down large tasks into smaller steps, track them in tools like Trello or Excel, and adjust my plan daily based on urgency.

---

2. How do you stay updated with the latest trends in data engineering?

I stay updated by:

- Following **blogs and newsletters** like Towards Data Science, KDnuggets, and Data Engineering Weekly
- Watching **YouTube channels** and **conference recordings** (e.g., from Snowflake, AWS re:Invent, or dbt)
- Taking short **online courses** on Coursera or YouTube
- Joining **tech communities** on LinkedIn, Reddit, and Slack groups
- Reading official docs for tools like Airflow, Boomi, or Spark

I try to apply what I learn through small projects or tutorials.

---

3. What would you do if you were assigned a task you've never done before?

I would start by **understanding the goal and context** of the task clearly. Then I'd:

- Break it down into parts I do understand
- **Research tutorials, documentation, and examples**
- **Ask teammates or mentors** for guidance where needed

- If allowed, create a **test environment** to experiment safely
  I'm not afraid to ask questions and learn fast—I enjoy learning by doing.

---

4. What interests you about data integration and ETL processes?

I find it fascinating how **raw data is transformed into valuable insights**. ETL and integration are the backbone of modern systems—whether it's syncing real-time orders or cleaning millions of records for analysis.

I enjoy:

- Automating workflows
- Working with APIs and data pipelines
- Solving real-world problems by connecting systems together

The mix of logic, creativity, and problem-solving makes it an exciting field for me.

---

5. Where do you see yourself in this role in the next 1–2 years?

In the next 1–2 years, I hope to:

- Gain deep hands-on experience with **ETL tools, databases, and cloud platforms**
- Become confident in designing and maintaining **end-to-end data pipelines**
- Contribute meaningfully to projects and start mentoring new interns or juniors
- Begin specializing in areas like **data modeling**, **real-time streaming**, or **data quality engineering**

---

6. Can you describe a typical project a trainee would work on in the first 3 months?

A typical trainee project might involve:

- Building or improving a **data pipeline** (e.g., importing data from CSV/API into a database)
- **Cleaning and transforming data** using Python or SQL
- Creating **data mappings** between source and target systems

- Writing basic **unit tests or validations** to ensure data quality
- Working under a mentor to learn internal tools, workflows, and best practices

This gives exposure to both **technical skills and real business use cases**, while also building confidence with the tools and team.