

Data Engineer Interview Questions and Answers

Data Engineer | Intern to senior level

Data engineer interview questions and answers covering SQL, query optimization, indexing, normalization, transactions, Python and database scaling.

64 questions Read online: <https://www.freedevtemplates.com/interview-questions/data-engineer>

Python

1. What is the difference between a list and a tuple in Python?

Python Easy Intern, Junior

A list is mutable and can be changed after creation, while a tuple is immutable; tuples are slightly faster, use less memory and can be used as dictionary keys when their items are hashable.

Use a list for a collection that grows, shrinks or is reordered, such as the items in a shopping cart. Use a tuple for a fixed group of values that belong together, such as a coordinate (x, y) or a database row. Tuples also signal to the reader that the data should not change.

Immutability applies to the tuple itself: if a tuple contains a list, that list can still be modified. A one-item tuple needs a trailing comma, (5,), because (5) is just the number 5 in parentheses.

```
items = [1, 2, 3]
items.append(4) # fine

point = (10, 20)
point[0] = 5 # TypeError: 'tuple' object does not support item assignment

locations = {(10, 20): "home"} # tuples can be dict keys
```

2. What is the difference between is and == in Python?

Python Easy Intern, Junior

== checks whether two objects have equal values, while is checks whether they are the exact same object in memory.

== calls the object's `__eq__` method. is compares identity (the same as comparing `id(a) == id(b)`). Two separate lists with the same content are == but not is.

Use is for singletons, mainly None (if x is None), True and False. Do not use is to compare numbers or strings: CPython caches small integers and some strings, so the result can look right in a quick test and fail in real code.

```
a = [1, 2]
b = [1, 2]
a == b # True (same value)
a is b # False (different objects)
c = a
c is a # True

if result is None:
    ...
```

3. What are *args and **kwargs in Python?

Python Easy Junior, Mid-level

***args collects extra positional arguments into a tuple and **kwargs collects extra keyword arguments into a dictionary, which lets a function accept a variable number of arguments.**

The names args and kwargs are conventions; the stars are what matter. They are useful for wrappers and decorators that must pass any arguments through to another function, and for APIs that accept optional settings.

The same stars work at the call site to unpack: `print(*numbers)` spreads a list into separate arguments and `func(**options)` spreads a dictionary into keyword arguments. The order in a signature is: normal parameters, *args, keyword-only parameters, **kwargs.

```
def report(title, *args, **kwargs):
    print(title, args, kwargs)

report("Sales", 1, 2, region="EU", year=2026)
# Sales (1, 2) {'region': 'EU', 'year': 2026}

nums = [3, 4]
opts = {"sep": "-"}
print(*nums, **opts) # 3-4
```

4. Why should you not use a mutable default argument in Python?

Python Medium Junior, Mid-level

Default argument values are evaluated once when the function is defined, so a default list or dictionary is shared between every call that does not pass its own, causing data to leak between calls.

This is one of the most common Python gotchas. The fix is to use None as the default and create a new object inside the function.

```
def add_item(item, items=[]): # bug: one shared list
    items.append(item)
    return items

add_item("a") # ['a']
add_item("b") # ['a', 'b'] <- surprise

def add_item(item, items=None): # correct
    if items is None:
        items = []
    items.append(item)
    return items
```

5. What is the difference between a shallow copy and a deep copy in Python?

Python Medium Junior, Mid-level

A shallow copy creates a new container but reuses the same nested objects, while a deep copy recursively copies everything so the new object is fully independent.

With a shallow copy (`copy.copy()`, `list.copy()`, slicing `items[:]`), changing a nested list in the copy also changes it in the original, because both point to the same inner object. `copy.deepcopy()` duplicates the nested objects as well, handling circular references, but it is slower and uses more memory.

Plain assignment `b = a` does not copy at all: both names refer to the same object.

```
import copy

original = [[1, 2], [3, 4]]
shallow = copy.copy(original)
deep = copy.deepcopy(original)

original[0].append(99)
print(shallow[0]) # [1, 2, 99] - shared inner list
print(deep[0]) # [1, 2] - independent
```

6. What is the difference between @staticmethod and @classmethod in Python?

Python Medium Junior, Mid-level

A classmethod receives the class as its first argument (cls) and can create or configure instances, while a staticmethod receives no automatic argument and is just a function placed inside the class namespace.

Use @classmethod for alternative constructors such as Date.from_string(...), and because it uses cls, it works correctly for subclasses. Use @staticmethod for a utility function that is related to the class but needs neither the instance nor the class. An ordinary method receives self, the instance.

```
class User:
    def __init__(self, name):
        self.name = name

    @classmethod
    def from_email(cls, email):
        return cls(email.split("@")[0])

    @staticmethod
    def is_valid_email(email):
        return "@" in email
```

7. What are decorators in Python and how do they work?

Python Medium Mid-level, Senior

A decorator is a function that takes another function and returns a modified version of it, letting you add behaviour such as logging, caching or access control without changing the original code.

The @decorator syntax is shorthand for func = decorator(func). A typical decorator defines an inner wrapper function that runs code before and after calling the original function, and returns that wrapper. Use functools.wraps so the wrapped function keeps its name and docstring.

Decorators rely on functions being first-class objects and on closures. Common built-in ones include @property, @staticmethod, @classmethod and functools.lru_cache. Frameworks use them heavily, for example @app.route("/") in Flask.

```
import functools, time

def timed(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        start = time.perf_counter()
        result = func(*args, **kwargs)
        print(f"{func.__name__} took {time.perf_counter() - start:.3f}s")
        return result
    return wrapper

@timed
def load_data():
    ...
```

8. What are generators and the yield keyword in Python?

Python Medium Mid-level, Senior

A generator is a function that uses yield to produce values one at a time and remembers its state between calls, so it can process large or infinite sequences without holding everything in memory.

Calling a generator function does not run it; it returns an iterator. Each next() call resumes the function until the next yield. A list of ten million numbers takes hundreds of megabytes, while a generator producing them one by one uses almost none.

A generator expression, (x * x for x in range(10**7)), is the lazy counterpart of a list comprehension. Generators can only be consumed once. They are ideal for reading big files line by line, streaming API pages and building data pipelines.

Key points

- yield pauses the function and keeps local state
- Lazy evaluation saves memory
- A generator can be iterated only once

```
def read_large_file(path):
    with open(path) as f:
        for line in f:
            yield line.strip()

for row in read_large_file("big.csv"):
    process(row) # one line in memory at a time

total = sum(x * x for x in range(1_000_000)) # no list created
```

9. How does Python manage memory and garbage collection?

Python Medium Mid-level, Senior

CPython frees objects mainly through reference counting, deleting an object as soon as nothing refers to it, and uses a cyclic garbage collector to clean up groups of objects that reference each other.

Every object keeps a count of references to it. When the count reaches zero its memory is released immediately. This makes cleanup predictable, but reference counting cannot free cycles, such as two objects that point at each other. The gc module's generational collector periodically finds and frees those.

Memory leaks in Python are usually objects kept alive by long-lived references: global lists or dictionaries, caches without limits, or closures. Use weakref for caches that should not keep objects alive, __slots__ to shrink many small objects, generators to stream data, and tools such as tracemalloc to see what is allocating memory.

10. What is the GIL in Python and how does it affect multithreading?

Python Hard Senior

The Global Interpreter Lock (GIL) in CPython lets only one thread execute Python bytecode at a time, so threads do not speed up CPU-bound code but still help with I/O-bound work.

The GIL exists to keep CPython's memory management (reference counting) simple and safe. A thread releases it while waiting on I/O such as network calls and file reads, so many I/O-bound threads can overlap their waiting. For pure-Python CPU-bound loops, threads take turns and the program is no faster, sometimes slower.

To use multiple cores, use multiprocessing or concurrent.futures.ProcessPoolExecutor, which run separate interpreters, or push the heavy work into C extensions such as NumPy that release the GIL. Recent Python versions have an experimental free-threaded build that removes the GIL, but the default interpreter still has it.

Key points

- One thread runs Python bytecode at a time in CPython
- Threads are fine for I/O-bound work
- Use processes for CPU-bound work

11. What is the difference between threading, multiprocessing and asyncio in Python?

Python Hard Senior

Threading runs concurrent threads that share memory and suit blocking I/O, multiprocessing runs separate processes to use multiple CPU cores for CPU-bound work, and asyncio runs many non-blocking tasks cooperatively on one thread for high-volume I/O.

Choose by the type of bottleneck. For CPU-bound work such as image processing or number crunching, use multiprocessing so each worker has its own GIL, and accept the cost of starting processes and passing data between them. For I/O-bound work with libraries that block, such as requests, threads are simple and effective.

For thousands of concurrent network connections, asyncio with async/await has the lowest overhead because tasks yield control at each await, but every library in the path must be async-aware, and one blocking call stalls the whole event loop. Many production systems combine them, for example an async web server that sends CPU-heavy jobs to a process pool or task queue.

Key points

- CPU-bound: multiprocessing
- Blocking I/O: threading
- Massive concurrent I/O: asyncio

```
import asyncio, httpx

async def fetch(client, url):
    r = await client.get(url)
    return r.status_code

async def main(urls):
    async with httpx.AsyncClient() as client:
        return await asyncio.gather(*(fetch(client, u) for u in urls))

asyncio.run(main(["https://example.com"] * 5))
```

SQL

12. What is the difference between INNER JOIN and LEFT JOIN in SQL?

SQL Easy Intern, Junior

INNER JOIN returns only rows that have a match in both tables, while LEFT JOIN returns every row from the left table and fills the right table's columns with NULL when there is no match.

Use INNER JOIN when you only want records that exist on both sides, such as orders that have a customer. Use LEFT JOIN when the left table is the main list and the related data is optional, such as all customers including those who have never placed an order.

A common trick is to find rows with no match by left joining and filtering on WHERE right.id IS NULL. Be careful: putting a condition on the right table in the WHERE clause turns a left join back into an inner join, so put it in the ON clause instead. RIGHT JOIN is the mirror image of LEFT JOIN, and FULL OUTER JOIN keeps unmatched rows from both sides.

In the interview

INNER JOIN returns only rows that have a match in both tables. LEFT JOIN returns every row from the left table and fills the right side with NULL when there is no match. I reach for LEFT JOIN when missing data matters, for example customers with zero orders.

Key points

- INNER: matches only. LEFT: all left rows plus matches
- Unmatched right-side columns are NULL
- Filter the right table in ON, not WHERE, to keep left-join behaviour

```
-- every customer, with order count (0 if none)
SELECT c.id, c.name, COUNT(o.id) AS orders
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.id
GROUP BY c.id, c.name;

-- customers who never ordered
SELECT c.*
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.id
WHERE o.id IS NULL;
```

13. What is the difference between WHERE and HAVING in SQL?

SQL Easy Intern, Junior

WHERE filters individual rows before grouping, while HAVING filters groups after GROUP BY, so only HAVING can use aggregate functions like COUNT or SUM.

SQL processes a query in a logical order: FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY. Because WHERE runs before grouping, the aggregates do not exist yet. HAVING runs after grouping, so it can test them.

Put every condition that does not need an aggregate in WHERE. It reduces the rows before grouping, which is faster than filtering the groups afterwards.

```
SELECT department, COUNT(*) AS staff, AVG(salary) AS avg_salary
FROM employees
WHERE status = 'active' -- filters rows first
GROUP BY department
HAVING COUNT(*) >= 5; -- filters groups after aggregation
```

14. What is the difference between DELETE, TRUNCATE and DROP in SQL?

SQL Easy Intern, Junior

DELETE removes selected rows and can be rolled back, TRUNCATE quickly removes all rows and keeps the table structure, and DROP removes the entire table including its structure.

DELETE is a DML command: it can use a WHERE clause, fires row-level triggers, and is logged row by row, so it is slower on big tables but fully transactional. TRUNCATE is a DDL command that empties the whole table, usually resets identity counters and is much faster because it deallocates data pages instead of logging each

row. DROP TABLE deletes the table definition, indexes and permissions as well.

Transaction behaviour differs by database: in PostgreSQL and SQL Server TRUNCATE can be rolled back inside a transaction, while in MySQL and Oracle it causes an implicit commit. Always check your database's documentation before relying on it.

```
DELETE FROM orders WHERE created_at < '2024-01-01'; -- some rows
TRUNCATE TABLE temp_imports; -- all rows, keep table
DROP TABLE temp_imports; -- table is gone
```

15. What is the difference between a primary key and a foreign key?

SQL Easy Intern, Junior

A primary key uniquely identifies each row in its own table and cannot be NULL, while a foreign key is a column that references the primary key of another table to link the two.

A table has one primary key (which may be made of several columns). It guarantees uniqueness and is indexed automatically. A table can have many foreign keys. A foreign key enforces referential integrity: you cannot insert an order for a customer that does not exist, and you cannot delete a customer that still has orders unless you define a cascade rule such as ON DELETE CASCADE.

A foreign key column may contain duplicates and, unless declared NOT NULL, may be NULL. A unique key also enforces uniqueness but, unlike a primary key, allows NULL (the exact number of NULLs allowed depends on the database).

```
CREATE TABLE customers (
  id INT PRIMARY KEY,
  name VARCHAR(100) NOT NULL
);

CREATE TABLE orders (
  id INT PRIMARY KEY,
  customer_id INT NOT NULL,
  FOREIGN KEY (customer_id) REFERENCES customers(id) ON DELETE RESTRICT
);
```

16. What is the difference between UNION and UNION ALL in SQL?

SQL Easy Junior, Mid-level

UNION combines two result sets and removes duplicate rows, while UNION ALL combines them and keeps every row, which makes it faster.

UNION has to sort or hash the combined result to find duplicates, which costs time and memory. If you know the two queries cannot return the same row, or you want the duplicates, use UNION ALL. Both require the same number of columns with compatible types, and the column names come from the first query.

```
SELECT email FROM newsletter_subscribers
UNION -- one row per distinct email
SELECT email FROM customers;

SELECT email FROM newsletter_subscribers
UNION ALL -- keeps duplicates, faster
SELECT email FROM customers;
```

17. How do you find the second highest salary in SQL?

SQL Medium Junior, Mid-level

Use a subquery that takes the MAX salary below the overall MAX, or use DENSE_RANK() and pick rank 2, which also handles ties and generalises to the Nth highest.

The subquery version works on every database. The window function version is the one interviewers like best, because changing 2 to N gives you the Nth highest salary, and DENSE_RANK correctly treats equal salaries as the same rank. Use DISTINCT so tied rows return one value.

Another common answer is ORDER BY salary DESC LIMIT 1 OFFSET 1 (MySQL, PostgreSQL) or OFFSET 1 ROWS FETCH NEXT 1 ROWS ONLY (SQL Server), but you need SELECT DISTINCT salary first or ties break it. If there is no second salary, wrap the query so it returns NULL rather than an empty result when the question asks for that.

Key points

- Subquery with MAX is the simplest answer
- DENSE_RANK handles ties and any N
- ROW_NUMBER would be wrong when salaries tie

```
-- works everywhere
SELECT MAX(salary) AS second_highest
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);

-- Nth highest with window function (here N = 2)
SELECT DISTINCT salary
FROM (
  SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rn
  FROM employees
) ranked
WHERE rn = 2;
```

18. How do you find and delete duplicate rows in SQL?

SQL Medium Junior, Mid-level

Find duplicates with GROUP BY and HAVING COUNT(*) > 1, then delete them by numbering rows within each duplicate group with ROW_NUMBER() and removing every row after the first.

First decide what makes two rows duplicates (for example the same email) and which row you want to keep (usually the oldest, so the lowest id). Always run the matching SELECT before the DELETE, and take a backup or work inside a transaction.

ROW_NUMBER() OVER (PARTITION BY email ORDER BY id) numbers rows inside each group of identical emails, so any row with a number above 1 is a duplicate. In MySQL older than 8.0 window functions are unavailable, and you cannot select from the same table in a DELETE subquery directly, so you wrap the subquery in a derived table or self-join instead. Afterwards, add a unique constraint so the duplicates cannot come back.

```
-- 1. find
SELECT email, COUNT(*) AS copies
FROM users
GROUP BY email
HAVING COUNT(*) > 1;

-- 2. delete, keeping the lowest id (PostgreSQL / SQL Server / MySQL 8+)
WITH ranked AS (
  SELECT id, ROW_NUMBER() OVER (PARTITION BY email ORDER BY id) AS rn
  FROM users
)
DELETE FROM users WHERE id IN (SELECT id FROM ranked WHERE rn > 1);

-- 3. prevent it happening again
ALTER TABLE users ADD CONSTRAINT uq_users_email UNIQUE (email);
```

19. What is database normalization and what are the normal forms?

SQL Medium Junior, Mid-level

Normalization is organising tables to reduce redundancy and update anomalies: 1NF means atomic values, 2NF removes partial dependencies on part of a composite key, and 3NF removes dependencies between non-key columns.

In 1NF every column holds a single value (no comma-separated lists) and each row is unique. In 2NF the table is in 1NF and every non-key column depends on the whole primary key, which matters for composite keys. In 3NF the table is in 2NF and non-key columns depend only on the key, not on other non-key columns (a city and zip_code pair stored with an order is an example of what to move out).

Normalisation avoids storing the same fact in many places, so updates and deletes cannot leave inconsistent data. The trade-off is more joins. Reporting databases and read-heavy systems often denormalise on purpose, accepting some redundancy for faster reads.

Key points

- 1NF: atomic values. 2NF: no partial dependency. 3NF: no transitive dependency
- Goal: remove insert, update and delete anomalies
- Denormalise deliberately for read performance

20. What are ACID properties in a database?

SQL Medium Mid-level, Senior

ACID stands for Atomicity, Consistency, Isolation and Durability: a transaction either fully succeeds or fully fails, keeps the data valid, does not interfere with other transactions, and survives crashes once committed.

Atomicity: all statements in a transaction are applied or none are, so a bank transfer never debits one account without crediting the other. Consistency: a transaction moves the database from one valid state to another, respecting constraints. Isolation: concurrent transactions behave as if they ran in some order, to the degree set by the isolation level. Durability: once committed, data is stored permanently, typically via a write-ahead log.

Relational databases provide ACID by default. Many NoSQL systems relax some of these guarantees to gain availability or scale, which is the trade-off behind the BASE and CAP discussions.

In the interview

ACID describes what a transaction guarantees. Atomicity: all of it happens or none of it does. Consistency: constraints hold before and after. Isolation: concurrent transactions do not see each other half-finished. Durability: once committed, the change survives a crash. A bank transfer is the usual example: the debit and credit must both happen or neither.

```
BEGIN;  
UPDATE accounts SET balance = balance - 500 WHERE id = 1;  
UPDATE accounts SET balance = balance + 500 WHERE id = 2;  
COMMIT; -- or ROLLBACK; if anything failed
```

21. What is an index in SQL and how does it improve performance?

SQL Medium Mid-level, Senior

An index is a separate sorted data structure (usually a B-tree) that lets the database find rows without scanning the whole table, at the cost of extra storage and slower writes.

Without an index, a query like WHERE email = 'a@b.com' reads every row (a full table scan). With an index on email the database walks the tree in a handful of steps and jumps straight to the matching rows. Indexes also speed up JOIN, ORDER BY and GROUP BY on the indexed columns.

Every INSERT, UPDATE and DELETE must also update every index on the table, so do not index everything. In

a composite index the column order matters: an index on (last_name, first_name) helps a search on last_name alone but not on first_name alone. An index is also skipped when you wrap the column in a function (WHERE LOWER(email) = ...) or use a leading wildcard (LIKE '%smith'), and it is of little use on low-cardinality columns such as a boolean.

In the interview

An index is a separate sorted structure, usually a B-tree, that lets the database find rows without scanning the whole table. It speeds up reads that filter, join or sort on the indexed columns, but every insert and update must also maintain the index, so I add indexes for real query patterns and check the execution plan.

Key points

- Faster reads, slower writes, more storage
- Column order in composite indexes matters
- Functions on the column and leading wildcards prevent index use

```
CREATE INDEX idx_orders_customer_date ON orders (customer_id, created_at);

-- uses the index
SELECT * FROM orders WHERE customer_id = 42 AND created_at >= '2026-01-01';
-- cannot use it efficiently (function on the column)
SELECT * FROM orders WHERE DATE(created_at) = '2026-01-01';
```

22. What is the difference between a clustered and a non-clustered index?

SQL Medium Mid-level, Senior

A clustered index defines the physical order in which the table's rows are stored, so there can be only one per table, while a non-clustered index is a separate structure that points back to the rows.

In SQL Server and MySQL InnoDB the primary key is the clustered index by default, so the table data is the leaf level of that tree. Looking up by the clustered key is the fastest possible read, and range scans on it are efficient because neighbouring keys are stored together.

A non-clustered index stores the indexed columns plus a pointer (the clustered key or a row locator). A lookup finds the entry and then does a second lookup to fetch the remaining columns, unless the index covers the query by including all needed columns. Choose a narrow, ever-increasing clustered key, because wide or random keys (such as random GUIDs) are copied into every non-clustered index and cause page splits.

23. What are transaction isolation levels in SQL and what problems do they prevent?

SQL Hard Senior

Isolation levels (Read Uncommitted, Read Committed, Repeatable Read, Serializable) control how much concurrent transactions can see of each other's changes, trading consistency against concurrency to prevent dirty reads, non-repeatable reads and phantom reads.

A dirty read means reading data another transaction has not committed. A non-repeatable read means reading the same row twice in one transaction and getting different values because someone committed in between. A phantom read means running the same query twice and getting a different set of rows because rows were inserted or deleted.

Read Uncommitted allows all three. Read Committed prevents dirty reads (the default in PostgreSQL, SQL Server and Oracle). Repeatable Read also prevents non-repeatable reads (the default in MySQL InnoDB, and it also prevents most phantoms there). Serializable prevents all three by making transactions behave as if they ran one after another, at the highest cost in locking or retries.

Choose the lowest level that is correct for the business rule, and use explicit locking (SELECT ... FOR UPDATE) or optimistic concurrency with a version column for specific hot spots such as inventory or balances.

Key points

- Higher isolation = fewer anomalies but less concurrency

- Defaults differ by database
- Use row locks or version columns for critical updates

24. How do you optimize a slow SQL query?

SQL Hard Senior

Read the execution plan with EXPLAIN to find full scans and expensive joins, add or fix indexes for the filter and join columns, select only the columns and rows you need, and rewrite predicates so indexes can be used.

Start by measuring, not guessing: run EXPLAIN (or EXPLAIN ANALYZE in PostgreSQL, or the actual execution plan in SQL Server) and look for sequential scans on large tables, big gaps between estimated and actual row counts, sorts that spill to disk, and nested loops over huge inputs.

Typical fixes: add an index matching the WHERE and JOIN columns (and ORDER BY where possible); avoid SELECT *; make predicates sargable (compare the raw column, not a function of it); replace OFFSET pagination on large offsets with keyset pagination (WHERE id > last_seen_id ORDER BY id LIMIT 50); avoid N+1 query patterns from the application; update table statistics; and archive or partition very large tables.

If the query is still too slow, consider a read replica, caching, a materialised view or precomputed summary table. Confirm every change with the plan and with timings on production-sized data.

Key points

- Measure with the execution plan first
- Indexes, sargable predicates, fewer columns and rows
- Keyset pagination beats large OFFSET

```
EXPLAIN ANALYZE
SELECT o.id, o.total
FROM orders o
WHERE o.customer_id = 42
AND o.created_at >= '2026-01-01'
ORDER BY o.created_at DESC
LIMIT 50;

-- keyset pagination instead of a large OFFSET
SELECT id, total FROM orders
WHERE id < 98765
ORDER BY id DESC
LIMIT 50;
```

Git

25. What is Git and how is it different from GitHub?

Git Easy Intern, Junior

Git is a distributed version control system that tracks changes to files on your machine; GitHub is a hosting service for Git repositories that adds collaboration features such as pull requests, code review and CI.

Git records snapshots of a project as commits. Every clone holds the full history, so you can commit, branch, view logs and compare versions without a network connection. Branches are cheap pointers to commits, which is what makes feature branches and experiments practical.

GitHub, GitLab and Bitbucket host a copy of the repository on a server so a team can share it. On top of Git they add pull requests, code review, issue tracking, permissions and automation such as GitHub Actions. You can use Git without GitHub, but not the other way round.

In day-to-day work you commit locally with Git, push your branch to the hosted remote, open a pull request for review, and merge it once checks pass.

In the interview

Git is the version control tool that runs locally and stores the full history in every clone. GitHub is a service that hosts Git repositories and adds pull requests, reviews, issues and CI on top. I use Git to commit and branch, and GitHub to share work and get it reviewed.

Key points

- Git: the version control tool, works offline
- GitHub: a hosting and collaboration platform for Git repositories
- Every Git clone contains the full history

```
git init # start tracking a project
git add src/app.js # stage a change
git commit -m "Add login form"
git remote add origin https://github.com/you/project.git
git push -u origin main # share it on GitHub
```

26. What is the difference between git fetch and git pull?

Git Easy Intern, Junior

git fetch downloads new commits from the remote without changing your working branch; git pull runs a fetch and then merges (or rebases) those commits into your current branch.

git fetch updates your remote-tracking branches such as origin/main. Your own branch and working files are untouched, so it is always safe. You can then inspect what changed with `git log main..origin/main` before deciding how to integrate it.

git pull is git fetch followed by git merge by default. If your branch and the remote have both moved, it creates a merge commit or stops with conflicts. Many teams set `git config pull.rebase true` so pull replays your local commits on top of the remote instead, which keeps history linear.

A good habit before starting work is to fetch, review, then merge or rebase deliberately, rather than pulling blindly into a branch with uncommitted changes.

Key points

- fetch: download only, never touches your branch
- pull: fetch + merge (or rebase with `--rebase`)
- Fetch first when you want to review incoming changes

```
git fetch origin
git log --oneline main..origin/main # what is new on the remote?
git merge origin/main # integrate when ready

# or in one step
git pull --rebase origin main
```

27. What is the difference between git merge and git rebase?

Git Medium Junior, Mid-level

Both integrate changes from one branch into another: merge keeps both histories and joins them with a merge commit, while rebase rewrites your commits so they appear on top of the target branch, giving a linear history.

Merging main into a feature branch creates a merge commit with two parents. Nothing is rewritten, so it is safe on shared branches, and the history shows exactly when branches were combined. The cost is a busier history with many merge commits.

Rebasing a feature branch onto main takes each of your commits and replays it on the latest main, creating new commits with new hashes. The result reads as if you started your work from the current main, which makes git

log and git bisect easier to follow.

Because rebase rewrites history, the golden rule is: never rebase commits that other people have already based work on. Rebase your private feature branch to tidy it up, then merge it (often with a squash or fast-forward) into the shared branch.

In the interview

Merge combines two branches with a merge commit and never rewrites history, so it is safe on shared branches. Rebase replays my commits on top of the target branch, which gives a clean linear history but changes commit hashes. I rebase my own feature branch to keep it current and tidy, and I never rebase a branch someone else is working on.

Key points

- Merge preserves history and is safe on shared branches
- Rebase rewrites commits to produce a linear history
- Never rebase commits that others already use
- Use --force-with-lease, not --force, after rebasing

```
# merge: keeps history, adds a merge commit
git switch feature
git merge main

# rebase: replays feature commits on top of main
git switch feature
git rebase main
git push --force-with-lease # needed because hashes changed
```

28. How do you resolve a merge conflict in Git?

Git Medium Junior, Mid-level

Open the conflicted files, decide what the final code should be between the conflict markers, remove the markers, run the tests, then stage the files and continue the merge or rebase.

A conflict happens when two branches change the same lines, or one deletes a file the other edited. Git pauses the merge and marks each conflicted region with <<<<<<, ===== and >>>>>>. git status lists the files that need attention.

Resolving is not choosing a side. Read both changes, understand why each was made, and write the code that keeps the intent of both. Editors and git mergetool show the two versions side by side, but the decision is yours.

After editing, build and run the tests, because a conflict-free file can still be logically broken. Then git add the resolved files and run git merge --continue or git rebase --continue. If it goes wrong, git merge --abort returns you to the state before the merge.

Key points

- git status shows conflicted files
- Combine the intent of both changes, then remove the markers
- Run tests before continuing
- git merge --abort or git rebase --abort to start over

After editing: git add config.js && git merge --continue

```
<<<<<< HEAD
const timeout = 5000;
=====
const timeout = Number(process.env.TIMEOUT_MS);
>>>>>> feature/config

# resolved: keep the env variable, fall back to the old default
const timeout = Number(process.env.TIMEOUT_MS ?? 5000);
```

29. What is the difference between git reset and git revert?

Git Medium Mid-level, Senior

git revert creates a new commit that undoes an earlier one and is safe on shared branches; git reset moves the branch pointer back to an earlier commit, effectively removing later commits from that branch's history.

git revert <commit> calculates the inverse of a commit and records it as a new commit. History only grows, so teammates who already pulled are not affected. This is how you undo a bad change that has reached main or production.

git reset moves the current branch to another commit. With --soft the undone changes stay staged, with the default --mixed they stay in your working tree unstaged, and with --hard they are discarded. It is ideal for cleaning up local commits that have not been pushed.

If you reset by mistake, the commits are usually still recoverable for a while through git reflog, which records where your branch pointed before each move.

Key points

- revert: new commit, history preserved, safe when shared
- reset: moves the branch pointer, rewrites history
- --soft keeps changes staged, --mixed unstaged, --hard discards them
- git reflog can recover "lost" commits

```
# undo a pushed commit safely
git revert a1b2c3d

# rewrite local history (not yet pushed)
git reset --soft HEAD~1 # undo commit, keep changes staged
git reset --hard HEAD~1 # undo commit and discard changes

# recover from a bad reset
git reflog
git reset --hard HEAD@{2}
```

System Design

30. When should you use a SQL database vs a NoSQL database?

System Design Medium Mid-level, Senior

Choose SQL when your data is relational and you need transactions, joins and strong consistency; choose NoSQL when you need flexible schemas, very high write throughput or horizontal scaling for a specific access pattern.

Relational databases such as PostgreSQL and MySQL give you ACID transactions, constraints and ad-hoc queries, which suit orders, payments, users and most business applications. They are a safe default and scale a long way with indexes, replicas and caching.

NoSQL is a family: document stores (MongoDB) for flexible, nested records; key-value stores (Redis, DynamoDB) for very fast lookups by key; wide-column stores (Cassandra) for huge write-heavy workloads; graph databases for relationship queries. Pick by access pattern, not fashion. Many systems use both, for example PostgreSQL as the source of truth and Redis for caching and rate limits.

Key points

- SQL: relations, transactions, ad-hoc queries
- NoSQL: specific access patterns and horizontal scale
- Polyglot persistence is common

31. What is the difference between database sharding and replication?

System Design Hard Senior

Replication copies the same data to several servers to improve read capacity and availability, while sharding splits the data across servers so each holds only a part, which increases write capacity and total storage.

With replication, one primary accepts writes and replicas follow it. Reads can be spread across replicas and a replica can be promoted if the primary fails, but replicas can lag, so reads from them may be slightly stale, and write throughput is still limited by the single primary.

Sharding partitions rows by a shard key, such as `user_id`, so each shard handles a fraction of the traffic. Choosing the key is the critical decision: a bad key creates hot shards, and cross-shard queries, joins and transactions become expensive or impossible. Range sharding supports range queries but can be uneven; hash sharding spreads load evenly; consistent hashing makes adding shards cheaper. Shard only when replication, caching, indexing and a bigger primary no longer suffice, because it adds permanent operational complexity.

Key points

- Replication: copies, for reads and failover
 - Sharding: partitions, for writes and storage
 - Shard key choice decides hot spots and query cost
-

Behavioral

32. How do you answer "Tell me about yourself" in a software engineer interview?

Behavioral Easy Intern, Junior

Give a 60-90 second summary in three parts: what you do now, one or two achievements that show your strengths, and why this role is the logical next step.

This question is an invitation to set the agenda for the interview, not a request for your life story. Interviewers want a short, relevant summary that tells them where to dig deeper.

Use a present, past, future shape. Present: your current role or studies and the kind of work you do. Past: one or two concrete things you built or improved, with a result. Future: why this team and role is the next step for you.

Tailor it to the job description. For a backend role, lead with APIs and data work; for a frontend role, with user-facing features and performance. Practise it out loud until it sounds natural rather than memorised.

Key points

- Keep it to about 60-90 seconds
 - Present, past, future structure
 - Pick achievements relevant to this job
 - End with why you want this role
-

33. How do you answer "Why should we hire you?" as a developer?

Behavioral Easy Intern, Junior

Connect two or three of your strongest, provable skills directly to the needs in the job description, give a short piece of evidence for each, and show genuine interest in the team's work.

This is a closing pitch. The interviewer wants a confident, specific summary of why you fit this role, not a list of adjectives like "hard-working" and "passionate".

Before the interview, pick the two or three requirements in the job description that matter most and match each one with evidence from your experience: a project, a metric, a responsibility you held.

Finish with motivation that is specific to the company: its product, technical challenges or engineering culture. Keep the whole answer under a minute.

Key points

- Match your strengths to their stated needs
 - Back each claim with a concrete example
 - Show specific interest in the company
 - Keep it under a minute
-

34. What are you currently doing to grow as a software engineer?

Behavioral Easy Intern, Junior

Give specific evidence of deliberate learning through projects, feedback, reading, experimentation or mentoring rather than listing technologies.

Interviewers want to know whether your growth is intentional and connected to your current level and future responsibilities.

A strong answer combines a skill you are improving, how you practice it and evidence that your behavior has changed.

For senior candidates, include growth beyond coding: architecture, communication, mentoring, product understanding or leadership.

Key points

- Choose specific growth areas
 - Practice deliberately
 - Use feedback
 - Show evidence of improvement
-

35. Why do you want to work for our company?

Behavioral Easy Intern, Junior

Connect something specific about the company, product, engineering challenges or team to your own experience and career goals.

A good answer demonstrates that you understand what the company actually does and why the role matters to you.

Choose two or three specific reasons and connect each to evidence from your background or interests.

Avoid generic statements that could be copied into any interview.

Key points

- Research the company
 - Be specific
 - Connect to your experience
 - Show realistic motivation
-

36. How do you answer "Tell me about a difficult project you worked on"?

Behavioral Medium Junior, Mid-level

Choose a real project with a technical or organisational challenge, explain your specific role and decisions with the STAR method, and finish with the result and what you learned.

The interviewer is testing how you handle complexity and pressure, and whether you can explain technical work to someone who was not there. The difficulty itself matters less than how you reasoned through it.

Pick a project where you made meaningful decisions. Briefly set the context, then spend most of the answer on your actions: how you broke the problem down, what options you considered, why you chose one, and how you worked with others.

Close with an honest result. If something went wrong, say so and explain what you would do differently. Senior

candidates should also show how they reduced risk for the team, for example by phasing a migration or adding monitoring.

Key points

- Use STAR: Situation, Task, Action, Result
 - Spend most of the time on your actions
 - Say "I" for your work and "we" for the team's
 - Include what you learned
-

37. How do you answer "Tell me about a production bug you fixed"?

Behavioral Medium Junior, Mid-level

Walk through how you noticed the issue, limited the impact, found the root cause, fixed it, and what you changed afterwards so the same class of bug could not happen again.

This question tests your debugging process and how you behave under pressure. Interviewers want a calm, methodical story, not a hero story.

A strong answer has a clear order: detection (alert, logs, user report), mitigation first (rollback, feature flag, scaling), investigation (reproducing it, narrowing the cause with logs, metrics or git bisect), the actual fix, and prevention (tests, monitoring, a post-mortem).

Be specific about the technical cause, at a level the interviewer can follow. A memorable detail, such as a timezone edge case or a missing index that only hurt at peak traffic, makes the story credible.

Key points

- Mitigate impact before hunting for the root cause
 - Explain how you narrowed the cause down
 - Finish with prevention: tests, alerts, process changes
 - Stay blameless
-

38. Tell me about a mistake you made and what you learned from it.

Behavioral Medium Junior, Mid-level

Choose a real mistake, take ownership, explain the impact, what you did to recover, and what you changed so it would not happen again.

Interviewers are testing accountability, self-awareness and whether you learn from mistakes rather than whether you have a perfect record.

Choose a mistake that is meaningful but safe to discuss. Spend more time on your response and learning than on the mistake itself.

A strong answer shows prevention: a test, checklist, monitoring, review practice, documentation or process improvement.

Key points

- Take ownership
 - Explain impact honestly
 - Show recovery
 - Explain prevention
-

39. Tell me about a time you received difficult or critical feedback.

Behavioral Medium Junior, Mid-level

Describe feedback that was uncomfortable but useful, explain how you responded, and show the measurable or observable improvement that followed.

The interviewer wants evidence that you can accept feedback without becoming defensive and turn it into better

performance.

Choose feedback that was genuinely actionable. Explain what you initially thought, what you understood after reflection, and what you changed.

For senior candidates, show how the experience changed the way you give feedback to others as well.

Key points

- Listen without defensiveness
 - Ask clarifying questions
 - Act on feedback
 - Show improvement
-

40. Tell me about a project or decision that failed.

Behavioral Medium Junior, Mid-level

Choose a genuine failure, explain why it failed, what you owned, how you recovered, and what you changed in future projects.

Good answers demonstrate resilience and learning rather than trying to make the failure sound like a success.

Be precise about the original assumption, the signal that showed it was wrong, and the decision you made once you knew.

Senior candidates should explain how they changed team practices or reduced similar risk afterwards.

Key points

- Be honest
 - Identify the failed assumption
 - Recover quickly
 - Show systemic learning
-

41. Tell me about a time you had to deliver under a very tight deadline.

Behavioral Medium Junior, Mid-level

Show how you prioritized the essential outcome, communicated trade-offs, reduced risk and delivered without pretending everything could be done.

The interviewer is testing prioritization and communication under pressure, not willingness to work unlimited hours.

Explain what you cut, deferred or simplified and why. If quality or risk was temporarily accepted, explain how you contained it.

Strong answers include early communication when the original scope was unrealistic.

Key points

- Define the must-have outcome
 - Prioritize ruthlessly
 - Communicate trade-offs
 - Protect critical quality
-

42. Tell me about a time you disagreed with a code review comment.

Behavioral Medium Junior, Mid-level

Show that you can defend a technical decision with evidence while remaining open to a better argument.

Code review is collaborative engineering, not a contest of authority.

Explain the concern behind the comment before explaining your position. A strong answer may end with you changing your implementation.

If the disagreement remained, describe how you used project conventions, documentation, tests or a short experiment to resolve it.

Key points

- Understand the concern
 - Use evidence
 - Stay respectful
 - Accept better reasoning
-

43. Why are you looking to leave your current company?

Behavioral Easy Mid-level, Senior

Keep the answer professional and future-focused: explain what you are looking for next without attacking your current or former employer.

Interviewers are checking motivation, professionalism and whether your expectations match the role.

Focus on positive reasons such as greater ownership, a different technical challenge, product scale or opportunities to grow.

If there were genuine difficulties, describe them factually and briefly, then return to what you want to build next.

Key points

- Stay positive
 - Be truthful
 - Focus on the future
 - Connect to the role
-

44. How do you answer "Tell me about a time you disagreed with a teammate"?

Behavioral Medium Mid-level, Senior

Describe a real technical or process disagreement, show that you listened and argued with evidence rather than opinion, explain how a decision was reached, and how you supported it afterwards even if it was not your idea.

Interviewers want to know whether you can disagree productively. Teams need people who raise concerns, but also people who commit once a decision is made.

Choose a disagreement about something that mattered, such as an architecture choice, a deadline or code review standards. Avoid personal conflicts. Show that you first understood the other person's reasoning, then brought data: a benchmark, a prototype, an incident report.

The outcome does not need to be that you won. "We went with their approach, and I helped make it work" is a strong answer if you explain what you learned.

Key points

- Pick a professional, not personal, disagreement
 - Show you listened first
 - Use evidence: prototypes, data, constraints
 - Disagree and commit
-

45. Tell me about a time you took ownership of something that was not explicitly assigned to you.

Behavioral Medium Mid-level, Senior

Describe a problem you noticed, why you decided to act, how you coordinated with others, and the impact of taking ownership.

This question separates people who only complete assigned tickets from engineers who notice and solve important problems.

Do not describe work you simply volunteered for without explaining why it mattered to users, reliability, delivery or the team.

Show initiative without suggesting that you ignored ownership boundaries or other people.

Key points

- Identify an unowned problem
 - Explain why it mattered
 - Coordinate before changing shared systems
 - Show impact
-

46. Tell me about a time you had to work with unclear or changing requirements.

Behavioral Medium Mid-level, Senior

Explain how you identified ambiguity, clarified the important decisions, made reasonable assumptions and kept delivery moving.

Real engineering work often starts with incomplete information. Interviewers want to see structured thinking rather than waiting passively for perfect requirements.

Explain which questions you asked, which assumptions you documented and which decisions you intentionally kept reversible.

Strong candidates distinguish between uncertainty that must be resolved now and uncertainty that can be handled later.

Key points

- Clarify goals
 - Document assumptions
 - Prioritize unknowns
 - Keep decisions reversible where possible
-

47. Tell me about a time you had several important tasks competing for your attention.

Behavioral Medium Mid-level, Senior

Explain how you assessed urgency, impact, dependencies and risk, then communicated the resulting priorities.

Strong engineers do not prioritize only by who asks first. They consider customer impact, deadlines, dependencies, severity and reversibility.

Show how you made the priority decision visible to stakeholders so expectations were aligned.

For senior candidates, explain how you helped the team prioritize rather than managing only your own queue.

Key points

- Assess impact and urgency
 - Consider dependencies
 - Communicate priorities
 - Revisit when conditions change
-

48. Tell me about a time you disagreed with your manager or technical lead.

Behavioral Medium Mid-level, Senior

Show respectful disagreement, evidence-based reasoning, willingness to listen and commitment after the final decision.

Disagreeing professionally is healthy. The key is to challenge the decision rather than the person.

Explain what evidence you brought, what you learned from their perspective and how the final decision was made.

If the decision went against your recommendation, explain how you supported it afterwards.

Key points

- Challenge ideas, not people
 - Use evidence
 - Listen to constraints
 - Disagree and commit
-

49. Tell me about a process you improved for your team.

Behavioral Medium Mid-level, Senior

Identify a recurring source of wasted time or risk, make a focused improvement, and demonstrate its impact.

This question looks for engineers who improve the system around the code, not just the code itself.

Choose a process with repeated cost: deployment, code review, testing, incident response, onboarding or local setup.

Quantify the improvement when possible, but a clear reduction in errors or friction is also useful.

Key points

- Identify recurring pain
 - Make a focused change
 - Get team buy-in
 - Measure impact
-

50. Tell me about a repetitive task you automated.

Behavioral Medium Mid-level, Senior

Explain the manual problem, why automation was worthwhile, how you implemented it safely and what time or error reduction resulted.

Automation is valuable when it reduces recurring human effort or prevents avoidable errors.

Explain why you chose the level of automation you did rather than building a complicated system for a small problem.

Mention maintenance, failure handling and how users or teammates adopted the automation.

Key points

- Identify recurring cost
 - Choose proportional automation
 - Handle failures
 - Measure savings
-

51. Tell me about a time you explained a complex technical concept to a non-technical person.

Behavioral Medium Mid-level, Senior

Show how you adapted your explanation to the audience, focused on the outcome and avoided unnecessary technical jargon.

Senior engineers often communicate with product, support, sales and leadership. The ability to make technical risk understandable is important.

Explain the audience, what they needed to decide and how you translated the technical issue into business impact.

Avoid presenting technical simplification as hiding important information; explain the right level of detail for the decision.

Key points

- Understand the audience
 - Explain impact first
 - Avoid unnecessary jargon
 - Confirm understanding
-

52. Tell me about a problem you prevented before it reached production.

Behavioral Medium Mid-level, Senior

Show how you noticed a risk early, validated it, communicated it and added a safeguard.

Preventative engineering demonstrates maturity because the best incident is often the one that never happens.

Explain the signal that made you suspicious and how you verified the risk instead of relying on intuition alone.

Finish with the lasting safeguard: test, validation, monitoring, review, rollout control or design change.

Key points

- Spot risk early
 - Validate before escalating
 - Communicate clearly
 - Add a lasting safeguard
-

53. Tell me about a time you had to work with a difficult stakeholder.

Behavioral Medium Mid-level, Senior

Describe how you understood the stakeholder's goals, handled disagreement professionally and reached a workable outcome.

The interviewer is evaluating communication and emotional maturity, not whether you can complain about difficult people.

Describe the stakeholder's legitimate concern before explaining the conflict. This makes the story balanced and credible.

Focus on shared goals, explicit trade-offs and communication rather than personality differences.

Key points

- Understand their goals
 - Stay professional
 - Use facts and options
 - Find shared outcomes
-

54. Tell me about a technical debt decision you made.

Behavioral Hard Mid-level, Senior

Explain why you accepted, reduced or paid down technical debt and how you managed the associated risk.

Technical debt is not automatically bad. Interviewers want to see whether you understand its cost and deliberately manage it.

Describe the business constraint, the engineering risk and why the chosen approach was appropriate at that time.

Strong answers include a trigger or plan for addressing the debt instead of leaving it invisible.

Key points

- Identify the debt
 - Understand its cost
 - Make the trade-off explicit
 - Create a follow-up trigger
-

55. Tell me about a time you introduced a new technology or tool to a project.

Behavioral Hard Mid-level, Senior

Start with the problem, evaluate alternatives, validate the technology with a small experiment and explain how you managed adoption and risk.

Interviewers want to know that you choose technology because it solves a problem, not because it is fashionable.

Discuss the existing approach, why it was insufficient, alternatives considered, proof of concept and operational implications.

Senior candidates should include how they helped the team adopt and maintain the new technology.

Key points

- Start with the problem
 - Evaluate alternatives
 - Prototype before committing
 - Plan adoption and maintenance
-

56. Tell me about a time you mentored or helped another developer grow.

Behavioral Medium Senior

Explain how you identified what the developer needed, coached rather than simply solved the problem, and measured their progress.

Mentoring is about increasing another engineer's ability to solve problems independently.

Strong answers describe questions, pairing, feedback, documentation or progressively larger ownership rather than simply giving answers.

Senior candidates should show patience and adaptation to the individual developer's level.

Key points

- Understand their starting point
 - Teach reasoning, not only solutions
 - Give actionable feedback
 - Increase independence
-

57. Tell me about a time you influenced a technical decision without having formal authority.

Behavioral Hard Senior

Show how you built trust, understood competing concerns, used evidence and helped the group reach a decision.

Senior engineers often influence architecture and engineering practices without being the manager or final decision-maker.

The strongest stories show persuasion through clarity and evidence rather than relying on seniority or force of personality.

Explain what happened after the decision and whether you supported the team's final choice.

Key points

- Build alignment
 - Use evidence
 - Understand other constraints
 - Support the final decision
-

58. Tell me about a time you had to choose between two good technical approaches.

Behavioral Hard Senior

Explain the constraints, alternatives, trade-offs, decision criteria and why the chosen option was appropriate for the context.

Senior engineering decisions rarely have a universally correct answer. Interviewers want to see explicit trade-off reasoning.

Compare options using factors such as reliability, complexity, performance, cost, team expertise, delivery time and future change.

A strong answer acknowledges what you gave up by choosing the preferred option.

Key points

- Define decision criteria
 - Compare alternatives
 - Make trade-offs explicit
 - Choose for the context
-

59. Tell me about a major production incident you helped handle.

Behavioral Hard Senior

Cover detection, communication, mitigation, investigation, recovery and the changes made afterwards to reduce recurrence.

Incident questions test calm decision-making and ownership under pressure.

Separate immediate mitigation from root-cause analysis. During an incident, restoring service is usually more important than finding the perfect explanation immediately.

For senior candidates, include how you coordinated people, communicated status and improved the system after the incident.

Key points

- Protect users first
 - Communicate clearly
 - Investigate systematically
 - Prevent recurrence
-

60. Tell me about a time you decided not to use a technology or framework.

Behavioral Hard Senior

Explain why the technology looked attractive, what problem it actually solved, and why its cost or complexity was not justified in your context.

Knowing when not to add technology is an important engineering skill.

A strong answer compares the expected benefit against learning cost, operational complexity, team capability, migration cost and long-term maintenance.

Avoid making the decision sound ideological; show that you would reconsider if the constraints changed.

Key points

- Define the actual problem
 - Compare total cost
 - Avoid unnecessary complexity
 - State when the decision could change
-

61. Tell me about a time you had to balance speed of delivery with technical quality.

Behavioral Hard Senior

Explain the deadline, the quality risks, what you protected, what you simplified and how you managed any remaining debt.

Engineering teams constantly balance delivery and quality. Strong answers show deliberate trade-offs rather than treating one as always correct.

Separate reversible shortcuts from dangerous shortcuts. Never present security, data integrity or critical reliability risks as acceptable just to ship faster.

Explain what you documented or scheduled for follow-up.

Key points

- Identify non-negotiable quality
 - Make trade-offs explicit
 - Contain risk
 - Track follow-up work
-

62. Tell me about a technical decision you made that turned out to be wrong.

Behavioral Hard Senior

Explain the original reasoning, the evidence that changed your view, how you corrected course and what decision-making practice improved afterwards.

This tests intellectual honesty and the ability to update your beliefs when reality disagrees with your assumptions.

Do not rewrite the story as if the decision was obviously correct. Explain what information was available at the time and what you missed.

The strongest answers show a change in process, such as better experiments, metrics or review criteria.

Key points

- Own the decision
 - Explain original assumptions
 - Recognize evidence
 - Improve the decision process
-

63. Tell me about a project you led from idea to production.

Behavioral Hard Senior

Walk through how you shaped the problem, planned the work, aligned stakeholders, managed risks and measured the outcome.

For senior engineers, leadership means creating clarity and reducing risk, not simply assigning tasks.

Cover discovery, design, execution, testing, rollout, monitoring and follow-up. Make your individual leadership contribution clear.

Include a moment where you made a meaningful trade-off or changed the plan based on new information.

Key points

- Define the problem
-

- Align stakeholders
 - Plan and de-risk
 - Lead delivery
 - Measure outcome
-

64. Tell me about a time you worked with a teammate who was struggling.

Behavioral Hard Senior

Show empathy, clear expectations, practical support and appropriate escalation rather than blame.

The interviewer wants to see whether you can help a teammate without becoming their manager or doing their work for them.

Start by understanding the cause: unclear requirements, missing knowledge, workload, personal circumstances or process problems.

For senior candidates, explain how you balanced support with accountability and involved the appropriate lead when necessary.

Key points

- Understand the cause
 - Set clear expectations
 - Offer targeted support
 - Maintain accountability
-

These answers are written for real interview conversations. Prefer explaining a trade-off out loud over reciting a definition. Full explanations and related questions: <https://www.fredevtemplates.com/interview-questions/data-engineer>