Describe a time you resolved an RDS outage— what metrics did you monitor?

One evening, our production RDS database — it was PostgreSQL — suddenly started causing issues. Users were complaining that the checkout process was hanging, and our API logs showed database connection errors.
As the on-call DevOps engineer, my first goal was to find out what exactly was wrong and get the system back up quickly.
I went straight to **CloudWatch** to check the key metrics.
I noticed the **CPU utilization** was very high — around 90%.
The **database connections** were also at their maximum limit, which meant the app was opening too many connections.
Storage space looked normal, so it wasn't a disk issue.
But I saw **ReadIOPS and WriteIOPS** were unusually high, which told me there was heavy query activity.
Next, I opened **Enhanced Monitoring** to look deeper at the OS level.
There I could see that the system load was high and a lot of threads were waiting on disk I/O — basically, the database was busy waiting for the disk to read and write data.
Then I checked **Performance Insights**, which shows query-level details.
There I found one particular query doing a full table scan on the *orders* table — it was consuming most of the resources.
To fix things quickly, I killed that long-running query and temporarily increased the instance size from db.m5.large to db.m5.xlarge to give the system some breathing room.
Once things stabilized, I worked with the developers to rewrite the query and add the missing index so it wouldn't happen again.
We also set up **PgBouncer** for connection pooling to prevent the database from hitting the connection limit in the future.
As a preventive step, I created **CloudWatch alarms** — for example, alerts if CPU goes above 80%, or if connections reach 85% of the limit.
I also enabled **SNS notifications** so we'd get an immediate alert for any RDS failover or restart.

The service was restored within about 10 minutes.
After we fixed the query and added the index, response times dropped from around 5 seconds to under 200 milliseconds.
And because of the new alarms and monitoring setup, we could catch performance issues much earlier in future incidents.
So overall, this experience taught me the value of using all three views together —

- **CloudWatch** for system metrics,
- **Enhanced Monitoring** for OS-level details, and
- **Performance Insights** for query-level visibility.

That combination gave us a complete picture and helped us build a strong monitoring and incident response system."

During an RDS outage, what's the *first* place you would check for signs of resource saturation?
A. RDS parameter groups
B. CloudWatch metrics like CPUUtilization, DatabaseConnections, and IOPS
C. Secrets Manager
D. Route 53 health checks
✅ **Correct Answer: B**
💡 **Explanation:** CloudWatch gives **immediate visibility** into RDS resource usage (CPU, connections, I/O) — key to identifying early bottlenecks.

If your application logs show "too many connections" to RDS, which CloudWatch metric would confirm this issue?
A. FreeStorageSpace
B. DatabaseConnections
C. PoolConnection
D. FreeableMemory
✅ **Correct Answer: B.**
💡 **Explanation: DatabaseConnections** shows the number of active DB connections; when it nears the **max_connections** limit, new connections fail.

Which RDS feature gives **query-level visibility** to identify slow or blocking SQL statements?
A. Enhanced Monitoring
B. Performance Insights
C. CloudTrail
D. DB Parameter Groups
✅ **Correct Answer: B.**
💡 **Explanation: Performance Insights** shows top SQL queries by DB load, wait types, and users — helping pinpoint runaway or inefficient queries.

During the outage, CloudWatch showed CPUUtilization >90% and DatabaseConnections near the max. What does this likely indicate?

A. App connection pool exhaustion and runaway queries causing CPU saturation
B. Storage corruption
C. Missing parameter group
D. Snapshot in progress

✅ **Correct Answer: A.**

💡 **Explanation:** High CPU + high connections = **inefficient queries or app-side connection leaks**, not storage or configuration issues.

What are Runaway Queries?

Runaway queries are **database queries that consume excessive system resources** — CPU, memory, or I/O — for a long time, often much longer than expected.

Think of them as *"queries that go out of control"* — they keep running, scanning too much data, joining too many tables, or looping due to bad design or missing indexes.

💥 **Why do Runaway Queries cause CPU Saturation?**

When a query becomes runaway:

- It may perform **full table scans** on large datasets.
- It may have **inefficient joins** or missing indexes.
- It might be **stuck in a loop or lock wait**.
- It might be **executed repeatedly** due to poor application logic.

🧩 **MCQ 6 — (Mid-Level)**

If FreeStorageSpace is healthy but ReadIOPS and WriteIOPS are spiking, what's the likely cause?

A. Query or index inefficiency causing high disk I/O
B. IAM misconfiguration
C. DNS caching
D. CloudWatch throttling

✅ **Correct Answer: A.**

💡 **Explanation:** High IOPS without low storage indicates **queries causing heavy reads/writes** — usually missing indexes or large scans.

🧩 **MCQ 7 — (Mid-Level)**

Which of these **immediate mitigations** helps stabilize an RDS instance suffering from high CPU due to a runaway query?

A. Increase read replica count
B. Kill the offending query (pg_terminate_backend or KILL QUERY) and scale instance
C. Disable backups
D. Change subnet group

✅ **Correct Answer: B.**
💡 **Explanation:** Terminating the long-running query stops the load spike; temporarily scaling up adds compute headroom.
