

Guia de preparação
Databricks Data Engineer Associate

Tudo o que a prova cobra, organizado por domínio: conceitos-chave, comparações que confundem, pegadinhas reais, gatilhos do enunciado, questões comentadas e um plano para chegar pronto ao exame.

7	40	70%	6
capítulos por domínio	questões comentadas	nota de corte	provas completas no curso

ANTES DE COMEÇAR

Como usar este guia

Este guia foi feito para ser estudado junto com as 6 provas completas do curso Databricks Data Engineer Associate: o guia explica o que cai e onde estão as armadilhas; as provas treinam o ritmo e mostram o que você ainda não domina.

Quando	O que fazer
Semana 1	Faça o teste diagnóstico (página seguinte) e leia os capítulos 1 e 2. No fim da semana, faça a Prova 1 do curso no tempo do exame.
Semana 2	Leia os demais capítulos, começando pelos domínios em que você errou mais no diagnóstico. Faça as provas 2 e 3.
Semana 3	Releia os alertas de pegadinha e as tabelas de comparação. Faça as provas 4 e 5 e revise cada erro pela explicação.
Semana 4	Estude a folha de cola, faça a Prova 6 e refaça as provas abaixo de 80%. Agende o exame quando estiver acima de 80% em todas (a nota de corte é 70%).

Mapa do exame

Cada prova do curso tem 45 questões e 90 minutos, no formato do exame. Referência de aprovação: 70%.

Domínio (capítulo)	Peso	Neste guia
1. Databricks Intelligence Platform	6%	17 conceitos · 14 pegadinhas
2. Governança e segurança	15%	17 conceitos · 14 pegadinhas
3. Implementação de CI/CD	10%	16 conceitos · 14 pegadinhas
4. Ingestão e carga de dados	21%	18 conceitos · 14 pegadinhas
5. Solução de problemas, monitoramento e otimização	10%	17 conceitos · 14 pegadinhas
6. Trabalho com Lakeflow Jobs	16%	17 conceitos · 14 pegadinhas
7. Transformação e modelagem de dados	22%	19 conceitos · 14 pegadinhas

Como ler os capítulos: os **tópicos-chave** são o que mais cai; a **tabela de memorização** separa o que a prova costuma confundir; os **alertas de prova** mostram as armadilhas das alternativas erradas; os **gatilhos** ligam palavras do enunciado à resposta; e o **checklist** diz se você está pronto para o domínio.

Estude um pouco todo dia, de graça

Instagram @strecht.simulados

Um post de prova por dia: questão comentada, pegadinhas e o "não confunda" de cada certificação.

Seguir no Instagram →

YouTube Strecht Simulados

Shorts diários com pegadinhas de prova e, aos sábados, um simulado comentado.

Inscrever-se no canal →

DIAGNÓSTICO

Você já sabe isso?

Responda às 12 questões sem consultar nada. As respostas, com a explicação e o capítulo para revisar, estão no fim do guia. Comece o estudo pelos capítulos em que errou.

D1 · Databricks Intelligence Platform

Um job clássico da Exportadora Grãos do Sul tem fases leves e fases pesadas na mesma execução, e com tamanho fixo de workers a equipe paga por máquinas ociosas nas fases leves. O código não é compatível com serverless. Qual configuração da computação ajuda?

A. Autoscaling com mínimo e máximo de workers, ajustando os nós à carga
B. Auto termination de 10 minutos, para desligar os workers ociosos durante a execução
C. Número fixo maior de workers, para que as fases pesadas terminem antes e sobre folga
D. Single node com um driver grande, para eliminar o custo dos workers nas fases leves

D2 · Databricks Intelligence Platform

A Rede Supermercados Família agenda seus jobs de produção em um cluster all-purpose que fica ligado o dia todo e também é usado por analistas. O gerente quer reduzir custo e isolar os jobs. Qual é a recomendação da Databricks para essas tarefas de produção?

A. Usar um SQL warehouse classic como computação dos notebooks Python dos jobs
B. Manter o all-purpose e aumentar o tempo de auto termination para evitar reinícios do cluster compartilhado
C. Manter o all-purpose e configurar um pool para que os jobs iniciem mais rápido
D. Usar jobs compute (ou serverless para jobs), criado para a execução e encerrado ao final

D3 · Databricks Intelligence Platform

Na Companhia Aérea Céu Azul, um engenheiro quer fixar a versão do runtime usada por notebooks e jobs em serverless compute, como faz com o Databricks Runtime LTS na computação clássica. O que a documentação diz?

A. A versão do serverless é fixada por um init script que instala o runtime desejado
B. O serverless usa a versão LTS escolhida no momento da criação do workspace
C. A versão do serverless é escolhida no campo Databricks Runtime version de cada job
D. O serverless não tem versão fixa: a Databricks atualiza o runtime automaticamente para todos

D4 · Governança e segurança

Na Operadora Saúde Mais, a coluna telefone da tabela prd.cad.beneficiarios deve aparecer mascarada para os usuários, exceto para o grupo atendimento. A função prd.seg.mascara_tel(tel STRING) RETURNS STRING já existe e usa is_account_group_member. Qual comando aplica a máscara?

A. ALTER TABLE prd.cad.beneficiarios ALTER COLUMN telefone SET MASK prd.seg.mascara_tel
B. UPDATE prd.cad.beneficiarios SET telefone = prd.seg.mascara_tel(telefone)
C. ALTER TABLE prd.cad.beneficiarios SET ROW FILTER prd.seg.mascara_tel ON (telefone, nome)
D. GRANT SELECT (telefone) ON TABLE prd.cad.beneficiarios TO `atendimento`

D5 · Governança e segurança

O job de produção da Transportes Rápido Leste roda como a entidade de serviço sp-etl-prod, cujo applicationId é 7c1e9a2b-55d0-4c1f-9e3a-2f6b8d0c4a11. Ela precisa ler a tabela prd.frota.viagens. Como conceder o SELECT para essa entidade em SQL?

A. GRANT SELECT ON TABLE prd.frota.viagens TO 'sp-etl-prod' AS APPLICATION
B. GRANT SELECT ON TABLE prd.frota.viagens TO JOB 'sp-etl-prod'
C. GRANT SELECT ON TABLE prd.frota.viagens TO `7c1e9a2b-55d0-4c1f-9e3a-2f6b8d0c4a11`
D. GRANT SELECT ON TABLE prd.frota.viagens TO SERVICE_PRINCIPAL sp-etl-prod

D6 · Governança e segurança

Na Seguradora Rede Segura, uma engenheira com MANAGE e SELECT na tabela prd.sin.avisos quer aplicar um filtro de linha com a função prd.seg.f_regiao, criada por outro time. Ela já tem USE CATALOG e USE SCHEMA nos objetos envolvidos, mas a tentativa falha por falta de permissão. Qual privilégio adicional ela precisa?

A. EXECUTE na função prd.seg.f_regiao
B. MODIFY na tabela prd.sin.avisos
C. REFRESH na tabela prd.sin.avisos
D. APPLY TAG na função prd.seg.f_regiao

D7 · Implementação de CI/CD

No bundle da Cervejaria Malte Real, a mesma definição de cluster (versão do runtime, tipo de nó e número de workers) se repete em vários jobs e muda entre dev e prod. Como evitar a repetição no YAML?

A. Criar um job vazio só com o cluster e referenciá-lo com ${resources.jobs.cluster.id}
B. Declarar uma variável simples com a definição do cluster escrita como uma única string JSON
C. Usar lookup do tipo cluster com o nome de um cluster all-purpose já existente no workspace
D. Declarar uma variável com type: complex contendo a definição e usar ${var.cluster} em new_cluster

D8 · Implementação de CI/CD

Ao fazer Pull no Git folder, uma engenheira da Indústria Cerâmica Barro Fino recebeu conflitos de merge em dois notebooks que ela e um colega alteraram nas mesmas linhas. Como a documentação orienta resolver?

A. Executar Reset para o branch remoto, que mescla sozinho as duas versões dos notebooks
B. Resolver na UI do Git folder, mantendo as linhas desejadas sem os marcadores, ou clicar em Abort para desfazer
C. Fazer Commit & Push com os marcadores de conflito, que o provedor Git remove no pull request
D. Apagar o Git folder e clonar de novo, porque a UI não resolve conflitos em notebooks

D9 · Implementação de CI/CD

No bundle da Indústria Gráfica Impressão Fácil, o grupo operacao deve poder executar todos os jobs implantados, sem editá-los, em todos os targets. Como declarar isso uma única vez no databricks.yml?

A. Mapeamento permissions de nível superior com group_name operacao e level CAN_RUN
B. Variável complexa com a lista de grupos e lookup do tipo group no deploy
C. Mapeamento permissions em cada job com group_name operacao e level CAN_MANAGE no target
D. Mapeamento run_as de nível superior com group_name operacao em cada target

D10 · Ingestão e carga de dados

Antes de carregar um lote grande de arquivos CSV de um novo fornecedor na tabela bronze.fornecedores, a engenheira da Distribuidora Norte Sul quer verificar se os dados podem ser lidos, se o esquema bate com a tabela e se as restrições são atendidas, sem gravar nada. Qual cláusula do COPY INTO usar?

A. VALIDATE ALL
B. PATTERN = '*.csv'
C. FORMAT_OPTIONS ('mode' = 'FAILFAST')
D. COPY_OPTIONS ('force' = 'false')

D11 · Ingestão e carga de dados

O stream de Auto Loader da Fábrica de Doces Mel e Cia caiu no meio de um lote por falha da computação. Ao ser reiniciado com o mesmo checkpoint, o que garante que os arquivos não sejam perdidos nem processados duas vezes na tabela Delta?

A. O job reprocessa a pasta inteira a cada reinício, e o MERGE final remove as duplicatas
B. O Auto Loader apaga os arquivos já processados da origem, então só restam os que faltam processar
C. O Delta Lake compara o conteúdo de cada linha nova com as existentes e descarta as repetidas
D. Os metadados dos arquivos descobertos ficam em um RocksDB no checkpoint, e o stream retoma de onde parou

D12 · Ingestão e carga de dados

A equipe de plataforma da Siderúrgica Ferro Forte quer criar, sem usar a interface do Catalog Explorer, a conexão do Unity Catalog para um conector gerenciado de banco de dados, como parte de uma automação. Qual opção a documentação apresenta?

A. Definir as credenciais no mapeamento variables do bundle como texto
B. Criar um volume com um arquivo de credenciais lido pelo ingestion gateway
C. Gravar usuário e senha do banco em um widget do notebook do pipeline
D. Executar databricks connections create com --json pela Databricks CLI

01

CAPÍTULO 1 · 6% DA PROVA

Databricks Intelligence Platform

Aqui eu preciso escolher a computação certa para cada carga (all-purpose, jobs compute, serverless, SQL warehouse) e entender a arquitetura básica: control plane, compute plane clássico e serverless, Delta Lake, arquitetura medalhão e metastore do Unity Catalog. O domínio vale 6% (cerca de 3 questões por prova), e os enunciados trazem um cenário de custo ou operação: analistas SQL com picos, jobs Python curtos esperando partida, time de ML com GPU, cluster ocioso à noite. Onde mais erro é confundir modo de acesso com tipo de computação, achar que serverless tem versão fixa ou Spark UI e esquecer que pool ocioso ainda gera custo de nuvem. Como são poucas questões, a meta é acertar todas com regras simples.

Neste capítulo
17 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Control plane e compute plane

O control plane abriga os serviços de backend e a aplicação web, na conta da Databricks; os dados são processados no compute plane. O classic compute plane fica na conta de nuvem do cliente; o serverless compute plane fica na conta da Databricks, com isolamento de rede por workspace.

Metastore do Unity Catalog

Sistema central de governança de dados e de IA, definido no nível da conta e ligado aos workspaces. Um metastore pode atender vários workspaces da mesma região, e os objetos são referenciados como catalogo.esquema.objeto, sem o nome do workspace.

Delta Lake e log de transações

Cada operação vira uma nova versão no log de transações, o que viabiliza time travel, RESTORE e DESCRIBE HISTORY. Registrada no Unity Catalog, a mesma tabela é a fonte única governada para BI e IA. Parquet puro ou CSV não têm log, logo não têm versões nem histórico.

Controle de concorrência otimista (ACID)

Escritas passam por leitura, escrita dos arquivos e validação com commit atômico. Não há locks nem deadlock; se duas escritas conflitam, uma falha de forma controlada e a tabela não fica corrompida. Leituras usam isolamento de snapshot e escritas, write serializable por padrão.

Arquitetura medalhão

Bronze guarda o dado bruto como chegou (reprocessamento e auditoria), silver o dado limpo e validado, gold o dado enriquecido, agregado e dimensional para BI. A qualidade sobe a cada camada; o padrão é recomendado, não obrigatório, e não existe camada extra obrigatória (landing, platinum).

All-purpose compute

Computação interativa, que pode ser compartilhada por analistas. Tem auto termination por inatividade e, depois de encerrada, é excluída de vez 30 dias após o término, a menos que esteja fixada (pin). Não é a recomendação para jobs de produção.

Jobs compute

Computação efêmera, criada para a execução do job, isolada e encerrada no fim; é configurada na definição do job. Junto com serverless para jobs, é a recomendação para produção.

Serverless compute para notebooks e jobs

Recursos gerenciados pela Databricks, alocados sob demanda, com partida rápida e sem escolher instâncias. Exige workspace com Unity Catalog e roda sempre na versão mais recente do runtime, sem versão fixa. Limitações: sem R, só APIs do Spark Connect (sem RDD), sem Spark UI e sem logs do Spark, sem init scripts ou bibliotecas no escopo da computação e sem a maioria das configurações Spark.

Tipos de SQL warehouse

Serverless, pro e classic. O serverless reúne Photon, Predictive IO e Intelligent Workload Management e serve BI, análise exploratória e ETL com demanda variável. O classic tem só desempenho de entrada, leva minutos para iniciar e escala com menos agilidade.

Dimensionamento de SQL warehouse serverless

Comece com um warehouse maior e reduza se puder. Spill nas consultas pede cluster size maior; fila de consultas (gráfico Peak Queued Queries) pede mais clusters, configurando o máximo de clusters.

Modo de acesso standard

Antes chamado shared, isola as cargas de cada usuário com Lakeguard e é o recomendado para a maioria das cargas com vários usuários e Unity Catalog. Não suporta Databricks Runtime ML, GPU, APIs RDD nem R.

Modo de acesso dedicated

Antes chamado single user, é atribuído a um usuário ou grupo. Use quando a carga depende do que o standard não suporta, como Databricks Runtime ML com GPU, RDD ou R. No isolation shared é legado, sem isolamento e incompatível com a governança do Unity Catalog.

Photon

Motor vetorizado em C++ que acelera SQL e DataFrames sem mudança de código, com mais ganho em consultas longas sobre grandes volumes. Operações não suportadas voltam ao Spark de forma transparente. Já é o motor padrão de SQL warehouses e serverless; na computação clássica liga-se em Use Photon Acceleration.

Instance pools

Mantêm instâncias ociosas já criadas no provedor para clusters iniciarem e escalarem mais rápido; ao serem liberadas, voltam ao pool. Instância ociosa não gera DBU, mas o provedor de nuvem cobra por ela. Para cargas compatíveis, a Databricks recomenda serverless no lugar de pools.

Autoscaling

Define mínimo e máximo de workers e ajusta os nós à carga, ideal para tarefas longas com fases leves e pesadas. O driver continua ligado e cobrado mesmo com poucos workers.

Auto termination

Encerra a computação inteira depois de um período sem comandos (em minutos). Resolve all-purpose ligada à noite e nos fins de semana; não reduz workers durante uma execução.

system.billing.usage

Tabela de sistema de consumo para cobrança, usada para acompanhar DBUs por carga, inclusive serverless. Pode levar até 24 horas para refletir o consumo. Predictive optimization e data quality monitoring aparecem no SKU de serverless jobs.

Macetes para memorizar

Onde roda cada computação

Clássica em Casa (conta do cliente); Serverless na Sede (conta da Databricks).

Standard x dedicated

Se aparecer Runtime ML, GPU, RDD ou R, é dedicated; o resto, standard.

Warehouse lento

Spill = Size (maior); Fila = Frota (mais clusters).

Limitações do serverless

Sem RRISC: sem R, sem RDD, sem Init script, sem Spark UI, sem Configurações Spark.

Pool

Pool parado: DBU zero, VM cobrando.

Medalhão

Bronze bruto, Silver saneado, Gold gerencial. Só três camadas.

Desligar x encolher

Terminar a computação toda = auto termination; encolher workers = autoscaling.

Tabela de memorização: não confunda

Quando é...	...e quando é
Computação clássica Preciso escolher tipo de instância ou versão LTS do runtime, ou uso init scripts, R ou RDD; roda na conta de nuvem do cliente.	**Computação serverless** Quero partida rápida e zero gestão de infraestrutura para código compatível; roda no compute plane da conta da Databricks, com versão atualizada automaticamente.
All-purpose compute Análise interativa de analistas em notebooks, compartilhada durante o expediente.	**Jobs compute ou serverless para jobs** Jobs de produção: criado para a execução, isolado e encerrado ao final.

Modo standard Vários usuários em Python e SQL com Unity Catalog, custo menor e isolamento via Lakeguard.	**Modo dedicated** Databricks Runtime ML com GPU, APIs RDD ou R; atribuído a um usuário ou grupo.
SQL warehouse serverless Consultas concorrentes com picos imprevisíveis, partida em segundos e custo baixo quando ocioso.	**SQL warehouse classic** Só desempenho de entrada; leva minutos para iniciar e escala com menos agilidade.
Aumentar o cluster size do warehouse Consultas individuais com spill em disco no query profile.	**Aumentar o máximo de clusters do warehouse** Consultas enfileiradas, com Peak Queued Queries acima de zero.
Auto termination Computação inteira ociosa sem comandos (noite, fim de semana).	**Autoscaling** Carga que varia dentro da mesma execução; ajusta workers entre mínimo e máximo.
Instance pool Jobs clássicos que precisam de tipos de instância específicos e não podem ir para serverless.	**Serverless para jobs** Jobs compatíveis (sem R, RDD, init scripts) que querem partida rápida sem administrar instâncias.

Números que caem

Item	Valor
Exclusão de all-purpose encerrada sem pin	**30 dias após o término**
Atraso de system.billing.usage	**até 24 horas**
Consultas em que o Photon quase não muda o tempo	**abaixo de dois segundos**
Peso do domínio na prova	**6% (cerca de 3 das 45 questões)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Colocar a all-purpose em modo dedicated para ela desligar ao fim da sessão.
Como não cair: Modo de acesso define quem usa, não quando desliga. Desligar por inatividade é auto termination.

⚠ **Alerta de prova:** Fazer pin na computação para economizar à noite.
Como não cair: Pin só preserva a configuração de uma computação encerrada por mais de 30 dias; não desliga nada.

⚠ **Alerta de prova:** Autoscaling com mínimo de zero workers resolve o custo noturno.
Como não cair: Mesmo sem workers, o driver continua ligado e cobrado; o certo é auto termination.

⚠ **Alerta de prova:** Fixar a versão do serverless com init script ou no campo Databricks Runtime version do job.
Como não cair: Serverless não tem versão fixa: a Databricks atualiza o runtime para todos, e init scripts no escopo da computação não são suportados.

⚠ **Alerta de prova:** Spill nas consultas do warehouse se resolve com mais clusters.
Como não cair: Mais clusters atendem concorrência; dar mais recursos a cada consulta é cluster size maior.

⚠ **Alerta de prova:** A serverless roda na conta de nuvem do cliente e a clássica na conta da Databricks.
Como não cair: É o contrário: clássica na conta do cliente; serverless no compute plane da conta da Databricks.

⚠ **Alerta de prova:** Erro de modificação concorrente no Delta significa tabela corrompida e exige RESTORE.
Como não cair: O controle otimista falhou a escrita justamente para evitar corrupção; arquivos não confirmados não entram na tabela.

⚠ **Alerta de prova:** Delta bloqueia a tabela durante a escrita ou só aceita um escritor por vez.
Como não cair: Não há locks nem deadlock; várias escritas concorrentes são aceitas e só as conflitantes falham.

⚠ **Alerta de prova:** Pool ocioso não custa nada, ou mantém clusters inteiros cobrando DBU.
Como não cair: Pool guarda instâncias, não clusters: ociosas não geram DBU, mas o provedor cobra pela VM.

⚠ **Alerta de prova:** Databricks Runtime ML acelera joins e agregações em SQL.
Como não cair: Runtime ML traz bibliotecas de machine learning; quem acelera SQL e DataFrames sem mudar código é o Photon.

⚠ **Alerta de prova:** Usar o modo standard para treinar com Runtime ML em GPU.
Como não cair: Standard não suporta Runtime ML nem GPU: use dedicated.

⚠ **Alerta de prova:** Usar SQL warehouse como computação de notebooks Python de jobs.
Como não cair: SQL warehouse só executa tarefas SQL ou notebooks escritos inteiramente em SQL.

⚠ **Alerta de prova:** Um metastore por workspace, sincronizado por jobs que copiam permissões.
Como não cair: Um metastore pode ser ligado a vários workspaces da mesma região, com governança definida uma vez.

⚠ **Alerta de prova:** Usar a Spark UI ou o event log para investigar consulta lenta em serverless.
Como não cair: Serverless não tem Spark UI, logs do Spark nem event log de computação: use o query profile.

Como cai na prova

📝 **Como cai:** Cenário de carga com requisitos de custo, partida e gestão, pedindo a computação mais adequada.
Como resolver: Classifique a carga: interativa (all-purpose), produção (jobs compute ou serverless), SQL concorrente (SQL warehouse serverless). Depois cheque restrições: R, RDD ou init script tiram o serverless; Runtime ML, GPU, RDD ou R tiram o standard e pedem dedicated.

📝 **Como cai:** Pergunta sobre onde rodam control plane, compute plane clássico e serverless.
Como resolver: Control plane e serverless na conta da Databricks; computação clássica na conta de nuvem do cliente. Descarte as alternativas invertidas ou que juntam tudo no control plane.

📝 **Como cai:** Sintoma no SQL warehouse (spill ou fila) pedindo o ajuste.
Como resolver: Leia o sintoma: spill no query profile = cluster size; Peak Queued Queries alto = mais clusters. Trocar para classic ou reduzir tamanho nunca é a resposta.

📝 **Como cai:** Requisitos de rollback, auditoria e fonte única governada.
Como resolver: Procure Delta Lake (ACID, time travel, histórico) registrado no Unity Catalog; elimine CSV com cópias, Parquet puro e DataFrames em cache.

📝 **Como cai:** Afirmação sobre custo de pools, autoscaling ou auto termination.
Como resolver: Pool ocioso: sem DBU, com custo de VM. Autoscaling mantém o driver. Auto termination encerra tudo após inatividade.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Cluster ligado à noite e no fim de semana sem comandos	**Auto termination**
Dezenas de analistas SQL, picos imprevisíveis, partida em segundos	**SQL warehouse serverless**
Jobs Python curtos esperando minutos de partida, sem querer escolher instâncias	**Serverless compute para jobs**
Jobs de produção em all-purpose compartilhada com analistas	**Jobs compute ou serverless para jobs**
Runtime ML, GPU, RDD ou R em computação clássica	**Modo de acesso dedicated**
Vários usuários, custo menor e isolamento com Unity Catalog	**Modo de acesso standard (Lakeguard)**
Acelerar SQL e DataFrames pesados sem reescrever código	**Photon**
Fases leves e pesadas na mesma execução, pagando máquinas ociosas	**Autoscaling com mínimo e máximo**
Spill no query profile e Peak Queued Queries em zero	**Cluster size maior**
Voltar a tabela ao estado anterior, histórico de alterações, fonte única para BI e IA	**Delta Lake com time travel, governado pelo Unity Catalog**
Mesmos catálogos e permissões em dois workspaces da mesma região	**Um metastore ligado aos dois; nomes catalogo.esquema.tabela**
Consumo de negócio, agregados e modelos dimensionais	**Camada gold**
Custo por job serverless, cruzando DBUs	**system.billing.usage (atraso de até 24 horas)**
Exceção de modificação concorrente em duas escritas simultâneas	**Controle de concorrência otimista: falha controlada, sem corrupção**

🎯 Se você lembrar de só uma coisa deste capítulo...

- Serverless: sem versão fixa, sem Spark UI, sem R, RDD ou init scripts; diagnóstico pelo query profile.
- Standard é o recomendado para a maioria das cargas; dedicated quando houver Runtime ML com GPU, RDD ou R.
- Produção roda em jobs compute ou serverless; all-purpose compartilhada não é recomendada para jobs de produção.
- Conflito de escrita no Delta é falha controlada, nunca corrupção.

Essenciais para a prova (checklist)

- ☐ Sei dizer onde ficam control plane, compute plane clássico e serverless.
- ☐ Sei escolher entre all-purpose, jobs compute, serverless e SQL warehouse para um cenário.
- ☐ Sei as limitações do serverless e por que ele não tem Spark UI.
- ☐ Sei quando usar standard e quando usar dedicated.
- ☐ Sei diferenciar SQL warehouse serverless, pro e classic.
- ☐ Sei qual ajuste fazer no warehouse para spill e para fila de consultas.
- ☐ Sei o efeito de Photon, autoscaling, auto termination e pools no custo e no desempenho.

- ☐ Sei explicar como o Delta Lake garante ACID, time travel e histórico.
- ☐ Sei o papel de bronze, silver e gold e que não há camada extra obrigatória.
- ☐ Sei que um metastore atende vários workspaces da mesma região.

Questões de revisão do capítulo

R1.1 · Databricks Intelligence Platform

No SQL warehouse serverless da Rede de Varejo Global, consultas individuais de BI estão gravando spill em disco segundo o query profile, enquanto o gráfico Peak Queued Queries fica em zero. Qual ajuste a documentação indica?

A. Trocar o warehouse serverless por um classic de mesmo tamanho
B. Aumentar o tamanho do cluster (cluster size) do warehouse
C. Reduzir o tamanho do cluster para acelerar a partida das consultas
D. Aumentar o número máximo de clusters do warehouse

R1.2 · Databricks Intelligence Platform

A equipe de segurança da Siderúrgica Aço Forte pergunta onde rodam os recursos de computação no Databricks. O workspace usa tanto computação clássica quanto serverless compute para notebooks e jobs. Qual descrição está correta segundo a arquitetura da plataforma?

A. A serverless roda na conta de nuvem do cliente; a clássica roda na conta da Databricks
B. A computação clássica roda na conta de nuvem do cliente; a serverless roda em um compute plane na conta da Databricks
C. As duas rodam no control plane da Databricks, junto com a aplicação web e os serviços de backend
D. As duas rodam na conta de nuvem do cliente; só o control plane fica na conta da Databricks

R1.3 · Databricks Intelligence Platform

A Agroverde Cooperativa recalcula indicadores de safra várias vezes ao dia. Quando uma carga ruim entra, a equipe precisa voltar a tabela ao estado anterior; a auditoria exige o histórico das alterações; e os times de BI e de ciência de dados devem ler a mesma versão governada dos dados. Qual abordagem atende a esses requisitos?

A. Arquivos CSV em um volume, com cópias noturnas manuais para voltar a versões anteriores
B. DataFrames em cache na memória do cluster, recriados a cada atualização para os dois times
C. Tabelas Delta Lake com transações ACID e time travel, governadas pelo Unity Catalog
D. Arquivos Parquet no armazenamento de objetos, com consultas ad hoc para recuperar dados antigos

R1.4 · Databricks Intelligence Platform

A Cooperativa de Saúde Unimédica está organizando o lakehouse em camadas. Analistas de negócio e executivos devem consumir agregados e modelos dimensionais prontos, enquanto os dados brutos ficam preservados para reprocessamento e auditoria. Em qual camada da arquitetura medalhão ficam os dados que esses usuários devem consumir?

A. Silver, com dados limpos e validados, ainda sem as agregações de negócio
B. Bronze, com os dados no formato original e histórico completo da origem
C. Landing, uma quarta camada obrigatória antes da bronze para os arquivos
D. Gold, com modelagem dimensional e agregações prontas para BI e decisão

Respostas comentadas

R1.1. B) Aumentar o tamanho do cluster (cluster size) do warehouse

Por quê: Para serverless: comece com um warehouse maior e reduza se puder; aumente o cluster size se houver spill; configure o máximo de clusters para picos de concorrência, acompanhando Peak Queued Queries na aba de monitoramento.

✗ A) Incorreta. O tipo classic tem menos recursos de desempenho e não resolve o spill.

✗ C) Incorreta. Diminuir o cluster agravaria o spill.

✗ D) Incorreta. Mais clusters atendem mais consultas simultâneas; não dão mais recursos a cada consulta, e não há fila.

> **Dica de prova:** Spill em consultas = cluster size maior; fila de consultas = mais clusters.

R1.2. B) A computação clássica roda na conta de nuvem do cliente; a serverless roda em um compute plane na conta da Databricks

Por quê: A Databricks opera com um control plane (serviços gerenciados na conta da Databricks) e um compute plane, onde os dados são processados. O classic compute plane fica na conta de nuvem do cliente; o serverless compute plane, na conta da Databricks, com isolamento de rede por workspace.

✗ A) Incorreta. A descrição está invertida em relação à arquitetura.

✗ C) Incorreta. O control plane abriga serviços de backend e a aplicação web; o processamento de dados acontece no compute plane.

✗ D) Incorreta. Isso vale só para a computação clássica; a serverless roda na conta da Databricks.

> **Dica de prova:** Clássica = conta de nuvem do cliente; serverless = compute plane na conta da Databricks.

R1.3. C) Tabelas Delta Lake com transações ACID e time travel, governadas pelo Unity Catalog

Por quê: Delta Lake registra cada operação como uma nova versão no log de transações, o que viabiliza time travel, RESTORE e DESCRIBE HISTORY. Registrada no Unity Catalog, a mesma tabela vira a fonte única governada para BI e IA.

✗ A) Incorreta. Cópias manuais de CSV não têm transações ACID nem histórico confiável de operações e ainda ficam fora de uma governança única.

✗ B) Incorreta. DataFrames em memória são temporários: somem quando a sessão ou o cluster termina e não oferecem histórico nem trilha de auditoria.

✗ D) Incorreta. Parquet puro não tem log de transações; sem ele não há versões da tabela para restaurar nem histórico de operações.

> **Dica de prova:** Voltar a uma versão anterior + histórico de alterações + fonte única governada = Delta Lake com Unity Catalog.

R1.4. D) Gold, com modelagem dimensional e agregações prontas para BI e decisão

Por quê: Na arquitetura medalhão, a qualidade dos dados aumenta a cada camada: bronze (bruto, como chegou), silver (limpo e validado) e gold (enriquecido e agregado). Seguir esse padrão é recomendado, mas não obrigatório.

✗ A) Incorreta. A silver traz dados limpos e validados, usados por engenheiros, cientistas e analistas que precisam de detalhe, mas não as agregações finais.

✗ B) Incorreta. A bronze guarda os dados brutos para reprocessamento e auditoria e não é indicada para consumo de analistas.

✗ C) Incorreta. A arquitetura medalhão é formada por bronze, silver e gold; não há camada adicional obrigatória.

> **Dica de prova:** Consumo de negócio e BI = camada gold.

02

CAPÍTULO 2 · 15% DA PROVA

Governança e segurança

Vale 15% (cerca de 7 questões por prova) e gira em torno do Unity Catalog: tabelas gerenciadas e externas (criar, excluir, recuperar, converter), privilégios com GRANT e REVOKE para usuários, grupos e entidades de serviço, herança, propriedade e MANAGE, filtros de linha, máscaras de coluna, views dinâmicas e políticas ABAC com tags governadas. A prova pede o privilégio mínimo que resolve, o comando SQL exato ou o efeito de uma operação. Onde mais erro é achar que SELECT na tabela basta (faltam USE CATALOG e USE SCHEMA), que ALL PRIVILEGES inclui MANAGE e que DENY funciona no Unity Catalog: o guia oficial cita DENY, mas a documentação atual diz que o comando não é suportado no Unity Catalog; para tirar acesso, REVOKE.

Neste capítulo

17 tópicos-chave · 14 pegadinhas · 8 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Tabela gerenciada

Tipo padrão e recomendado: o Unity Catalog controla local e ciclo de vida dos arquivos (Delta ou Iceberg), com custo menor, predictive optimization e exclusão automática dos arquivos após o DROP (depois do período de recuperação). Clientes externos acessam por APIs abertas, não gravando direto no caminho.

Tabela externa

Indicada quando sistemas fora do Databricks leem ou gravam os arquivos diretamente ou o formato não é suportado em tabela gerenciada (gerenciadas usam Delta ou Iceberg; externas aceitam DELTA, CSV, JSON, AVRO, PARQUET, ORC e TEXT). O Unity Catalog governa o acesso, a auditoria e a linhagem, mas o DROP remove só os metadados. Criar exige CREATE EXTERNAL TABLE na external location e CREATE TABLE no esquema.

Managed storage location

Arquivos de tabelas gerenciadas ficam no managed storage location do esquema, do catálogo ou do metastore, valendo o nível mais específico. Para isolar um catálogo em bucket próprio mantendo as tabelas gerenciadas, defina o managed storage location no catálogo.

UNDROP e SET MANAGED

UNDROP TABLE recupera tabela gerenciada excluída dentro do período de recuperação, com metadados e privilégios (SHOW TABLES DROPPED lista; WITH ID resolve nomes repetidos; PK e FK não voltam). ALTER TABLE ... SET MANAGED converte externa Delta em gerenciada preservando nome, grants e histórico, com pouca parada e reversível; MOVE e COPY são só para tabelas estrangeiras.

Privilégios de uso

USE CATALOG e USE SCHEMA são portões: sem eles, SELECT na tabela não funciona. Ler tabela = USE CATALOG + USE SCHEMA + SELECT; alterar dados (INSERT, UPDATE, DELETE, MERGE) = MODIFY; criar tabela = CREATE TABLE no esquema.

Herança de privilégios

GRANT em catálogo ou esquema vale para os filhos atuais e futuros, sem cláusula especial. Exemplo: GRANT USE SCHEMA, SELECT ON SCHEMA prd.financeiro TO grupo; para todo o catálogo, GRANT USE CATALOG, USE SCHEMA, SELECT ON CATALOG. GRANT não aceita curinga em nomes nem WHERE.

Principais e sintaxe do GRANT

Principais são usuários, grupos e entidades de serviço; jobs não recebem privilégios. Entidade de serviço é identificada pelo applicationId entre crases. Não existe AS APPLICATION, TO JOB, SERVICE_PRINCIPAL nem WITH GRANT OPTION no Unity Catalog.

Propriedade e MANAGE

O dono tem todos os privilégios. Podem conceder: dono do objeto, dono do catálogo ou esquema que o contém, quem tem MANAGE e o admin do metastore. MANAGE delega administração de acesso e exclusão sem transferir propriedade e, em contêiner, vale para os filhos sem exigir USE SCHEMA. ALL PRIVILEGES não inclui MANAGE. Transferir dono: ALTER TABLE ... SET OWNER TO grupo.

REVOKE e DENY

REVOKE é idempotente: tem sucesso mesmo sem concessão prévia e não afeta outros privilégios; acesso herdado continua valendo se não for revogado onde foi concedido. DENY não é suportado no Unity Catalog (só no hive_metastore legado); negações explícitas existem apenas como ABAC DENY policies, em Beta.

BROWSE e auditoria de acesso

BROWSE no catálogo permite descobrir objetos e pedir acesso sem ler dados (recomendado para All account users). SHOW GRANTS ON TABLE lista concessões (quem não administra vê só as próprias). system.access.audit registra quem fez o quê e quando.

Volumes e external locations

Volume: READ VOLUME para ler e WRITE VOLUME para gravar e apagar arquivos. External location: READ FILES e WRITE FILES para acesso direto ao caminho, CREATE EXTERNAL TABLE e CREATE EXTERNAL VOLUME para registrar objetos. SELECT e MODIFY são só de tabelas.

Views e views dinâmicas

Em SQL warehouse, o usuário precisa de SELECT na view e o dono da view precisa de acesso às tabelas de base. Views dinâmicas fazem segurança por linha e coluna com CASE e is_account_group_member (grupos de conta); is_member avalia grupos do workspace e existe por compatibilidade.

Filtro de linha

UDF SQL que retorna BOOLEAN, avaliada em tempo de consulta; um filtro por tabela. Aplica-se com ALTER TABLE ... SET ROW FILTER função ON (coluna) ou já na criação com CREATE TABLE ... WITH ROW FILTER. Exige EXECUTE na função e ser dono (ou MANAGE e SELECT) na tabela; remova com DROP ROW FILTER antes de excluir a função.

Máscara de coluna

UDF SQL que recebe o valor e devolve o original ou mascarado, com tipo compatível com a coluna; uma máscara por coluna, que pode receber outras colunas. Aplica-se com ALTER TABLE ... ALTER COLUMN col SET MASK função ou na criação (col STRING MASK função). Padrão: CASE WHEN is_account_group_member('grupo') THEN col ELSE 'XXX' END.

Políticas ABAC

Aplicam filtros de linha e máscaras a todos os objetos que atendem às condições, inclusive os criados depois, a partir de tags governadas. Têm escopo (ON CATALOG, SCHEMA ou TABLE), principais (TO e EXCEPT), condição WHEN opcional, MATCH COLUMNS por has_tag e USING COLUMNS. Exigem Runtime 16.4 ou superior ou serverless, EXECUTE na UDF e MANAGE ou propriedade no escopo; donos de tabela não conseguem removê-las.

Tags governadas

Definidas na conta, com chaves e valores permitidos e controle de quem aplica; apoiam classificação, ABAC, custos e descoberta. Aplicar exige ser dono ou ter APPLY TAG (com USE CATALOG e USE SCHEMA) e a permissão de atribuição da tag. Na avaliação ABAC, esquemas e tabelas herdam tags do catálogo, mas colunas não.

Catálogo do workspace

Em workspace habilitado automaticamente, o catálogo com o nome do workspace tem os admins do workspace como donos; os usuários podem usá-lo e criar tabelas, volumes e funções no esquema default, sem SELECT padrão em todas as tabelas.

Macetes para memorizar

Ler uma tabela
Catálogo, Esquema, Select: três portas, três chaves.

Gerenciada x externa no DROP
Gerenciada leva os arquivos junto (após a recuperação); externa só esquece o nome.

Filtro x máscara
Filtro Fala sim ou não (BOOLEAN) para a linha; Máscara Mantém o tipo e troca o valor.

Poder de conceder
ALL PRIVILEGES usa; MANAGE administra; dono manda.

DENY
No Unity Catalog não se nega: se revoga.

Grupo de conta
Conta = is_account_group_member; is_member avalia grupos do workspace (compatibilidade).

Tags no ABAC
A tag desce do catálogo até a tabela, mas para na coluna.

Arquivos
Volume = READ/WRITE VOLUME; caminho cru = READ/WRITE FILES.

Tabela de memorização: não confunda

Quando é...	...e quando é
Tabela gerenciada Padrão recomendado: ciclo de vida pelo Unity Catalog, predictive optimization, DROP apaga os arquivos após a recuperação.	**Tabela externa** Outro sistema mantém ou lê os arquivos direto, ou formato não suportado; DROP remove só metadados.

Filtro de linha Esconder linhas: função BOOLEAN, SET ROW FILTER, um por tabela.	**Máscara de coluna** Esconder valores: função que devolve o mesmo tipo, SET MASK, uma por coluna.
Filtro ou máscara por tabela Regra pontual em uma tabela, gerenciada pelo dono ou por quem tem MANAGE.	**Política ABAC** Muitas tabelas, inclusive futuras, via tags governadas, que os donos não conseguem remover.
MANAGE Conceder e revogar acessos e excluir o objeto sem ser dono.	**ALL PRIVILEGES** Todos os privilégios de uso de dados, sem poder de conceder nem MANAGE.
BROWSE Descobrir objetos e ver metadados no Catalog Explorer, sem ler dados.	**SELECT** Ler dados (com USE CATALOG e USE SCHEMA).
READ VOLUME / WRITE VOLUME Arquivos dentro de um volume do Unity Catalog.	**READ FILES / WRITE FILES** Acesso direto ao caminho de uma external location.
REVOKE Retirar acesso concedido no Unity Catalog; idempotente.	**DENY** Só no hive_metastore legado; não suportado no Unity Catalog.

Números que caem

Item	Valor
Tags governadas por conta	**até 1.000**
Tags por objeto	**até 50**
Filtros de linha por tabela	**um**
Máscaras por coluna	**uma**
Políticas ABAC de filtro e máscara	**Databricks Runtime 16.4 ou superior, ou serverless**
Views em computação dedicated que exigem SELECT do usuário nas tabelas de base	**Databricks Runtime 15.3 ou anterior**
Peso do domínio na prova	**15% (cerca de 7 das 45 questões)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** SELECT na tabela basta para consultar.
Como não cair: Sem USE CATALOG e USE SCHEMA nos pais, a leitura falha.

⚠ **Alerta de prova:** ALL PRIVILEGES permite conceder acesso a outros.
Como não cair: ALL PRIVILEGES não inclui MANAGE; conceder exige MANAGE ou propriedade.

⚠ **Alerta de prova:** GRANT SELECT ... WITH GRANT OPTION para delegar concessões.
Como não cair: O Unity Catalog não usa GRANT OPTION; delegação é MANAGE ou propriedade.

⚠ **Alerta de prova:** GRANT OWNERSHIP ou GRANT MANAGE para trocar o dono.
Como não cair: Propriedade não é privilégio de GRANT; use ALTER TABLE ... SET OWNER TO.

⚠ **Alerta de prova:** DENY SELECT bloqueia o grupo e tem prioridade sobre GRANTs.
Como não cair: DENY não é suportado no Unity Catalog; use REVOKE.

⚠ **Alerta de prova:** REVOKE falha se o privilégio nunca foi concedido.
Como não cair: REVOKE tem sucesso mesmo sem concessão prévia.

⚠ **Alerta de prova:** DROP TABLE só remove metadados nas duas tabelas.
Como não cair: Isso vale para externa; na gerenciada os arquivos são apagados após o período de recuperação.

⚠ **Alerta de prova:** RESTORE TABLE ou DESCRIBE HISTORY recuperam tabela excluída.
Como não cair: Eles atuam em tabela existente; para excluída, UNDROP TABLE.

⚠ **Alerta de prova:** Recriar com CTAS para migrar externa em gerenciada.
Como não cair: CTAS perde histórico e permissões; use ALTER TABLE ... SET MANAGED (sem MOVE ou COPY).

⚠ **Alerta de prova:** Governança, auditoria, ACID e time travel são exclusivos de gerenciadas.
Como não cair: Externas Delta também têm; exclusivos são predictive optimization e limpeza automática após DROP.

⚠ **Alerta de prova:** Máscara que devolve STRING 'OCULTO' em coluna DATE, ou duas máscaras na coluna.
Como não cair: O retorno deve ser compatível com o tipo (NULL ou data) e há uma máscara por coluna.

⚠ **Alerta de prova:** is_member ou current_user() = 'grupo' para testar grupo de conta.
Como não cair: Use is_account_group_member; current_user devolve o usuário e não testa pertencimento.

⚠ **Alerta de prova:** Aplicar filtro de linha falha e falta MODIFY ou APPLY TAG.
Como não cair: Falta EXECUTE na função do filtro.

⚠ **Alerta de prova:** Política ABAC ROW FILTER para diretoria com EXCEPT account users.
Como não cair: Está invertido: TO account users e EXCEPT diretoria.

Como cai na prova

📝 **Como cai:** Usuário ou grupo não consegue fazer algo; qual concessão adicional, com menor privilégio?
Como resolver: Liste o que a ação exige (USE CATALOG, USE SCHEMA, SELECT, MODIFY, CREATE TABLE, EXECUTE, MANAGE) e compare com o que o enunciado diz que já existe. Descarte ALL PRIVILEGES e privilégios amplos quando algo pontual resolve.

📝 **Como cai:** Escolher o comando SQL exato entre variações de sintaxe.
Como resolver: Elimine sintaxes inventadas (WITH INHERITANCE, WITH CHILDREN, curinga, WHERE no GRANT, TO JOB, GRANT OWNERSHIP). Confira SET ROW FILTER ... ON (coluna), ALTER COLUMN ... SET MASK, SET OWNER TO e applicationId entre crases.

📝 **Como cai:** Restringir visibilidade por grupo em uma ou em muitas tabelas.
Como resolver: Linhas = filtro (BOOLEAN); valores = máscara (mesmo tipo). Uma tabela = SET ROW FILTER ou SET MASK; muitas, inclusive futuras, ou regra que o dono não pode remover = política ABAC com tags governadas.

📝 **Como cai:** Efeito de DROP, UNDROP, conversão ou escolha entre gerenciada e externa.
Como resolver: Pergunte quem controla os arquivos: Unity Catalog (gerenciada) ou outro sistema (externa). Aplique DROP, UNDROP e SET MANAGED conforme esse dono do ciclo de vida.

📝 **Como cai:** Pergunta sobre DENY ou sobre negar acesso no Unity Catalog.
Como resolver: DENY não é suportado no Unity Catalog; a resposta é REVOKE do privilégio onde foi concedido (ou ABAC DENY policy, em Beta, se o enunciado tratar de políticas).

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Arquivos mantidos ou lidos por sistema fora do Databricks	**Tabela externa**
Tabela gerenciada excluída por engano há dois dias	**UNDROP TABLE**
Externa Delta para gerenciada preservando nome, grants e histórico	**ALTER TABLE ... SET MANAGED**
Consulta falha mesmo com SELECT na tabela	**Falta USE CATALOG e/ou USE SCHEMA**
UPDATE e DELETE na tabela	**MODIFY**
Ler tabelas atuais e futuras de um esquema	**GRANT USE SCHEMA, SELECT ON SCHEMA**
Administrar acessos sem ser dono	**MANAGE**
Encontrar tabelas e pedir acesso sem ler dados	**BROWSE no catálogo**
Listar quem tem acesso à tabela	**SHOW GRANTS ON TABLE**
Quem acessou ou alterou objetos e quando	**system.access.audit**
Gerente vê só a própria região; auditoria vê tudo	**SET ROW FILTER com função booleana e is_account_group_member**
CPF em dezenas de tabelas, novas toda semana, exceto compliance	**Tag governada + política ABAC COLUMN MASK no catálogo**
Tabela deve nascer já filtrada	**CREATE TABLE ... WITH ROW FILTER**
Ler PDFs de um volume e só outro grupo grava	**READ VOLUME e WRITE VOLUME**

🎯 Se você lembrar de só uma coisa deste capítulo...

- Ler tabela = USE CATALOG + USE SCHEMA + SELECT; escrever = MODIFY.
- ALL PRIVILEGES não inclui MANAGE; dono do catálogo administra os filhos.
- DENY não é suportado no Unity Catalog; REVOKE é idempotente.
- Filtro de linha retorna BOOLEAN (um por tabela); máscara mantém o tipo (uma por coluna); ambos exigem EXECUTE na função.
- DROP em gerenciada apaga arquivos após a recuperação (UNDROP salva); em externa, só metadados.

Essenciais para a prova (checklist)

- ☐ Sei quando usar tabela gerenciada e quando usar externa e o efeito do DROP em cada uma.
- ☐ Sei recuperar tabela com UNDROP e converter externa em gerenciada com SET MANAGED.
- ☐ Sei os privilégios para ler, escrever, criar tabela, criar tabela externa e usar volumes.
- ☐ Sei como funciona a herança de privilégios em catálogo e esquema.
- ☐ Sei a diferença entre propriedade, MANAGE e ALL PRIVILEGES e quem pode conceder.
- ☐ Sei escrever GRANT e REVOKE para usuários, grupos e entidades de serviço.
- ☐ Sei por que DENY não vale no Unity Catalog.

- ☐ Sei criar e aplicar filtros de linha e máscaras de coluna e seus requisitos.
- ☐ Sei montar uma política ABAC com tags governadas, TO e EXCEPT.
- ☐ Sei usar BROWSE, SHOW GRANTS e system.access.audit.

Questões de revisão do capítulo

R2.1 · Governança e segurança

A Companhia Portuária Baía Azul vai migrar atracacoes, hoje externa (Delta) no esquema prd.ops, para o armazenamento gerenciado do Unity Catalog e assim usar predictive optimization, sem perder nome, grants e histórico de versões e com pouca parada. Qual comando é o recomendado?

A. ALTER TABLE prd.ops.atracacoes SET MANAGED MOVE
B. CREATE TABLE prd.ops.atracacoes_g AS SELECT * FROM prd.ops.atracacoes
C. ALTER TABLE prd.ops.atracacoes SET MANAGED
D. DROP TABLE prd.ops.atracacoes e CREATE TABLE com o mesmo nome

R2.2 · Governança e segurança

Um engenheiro da Rede de Farmácias Popular Mais tentou criar uma política ABAC de máscara de coluna no catálogo saude, usando a função prd.seg.mascara_cpf, e recebeu erro de permissão. Além de computação compatível, o que ele precisa ter?

A. CREATE TABLE no catálogo saude e APPLY TAG em cada coluna mascarada
B. Ser admin da conta, porque políticas ABAC só são criadas no nível da conta
C. MANAGE no catálogo saude (ou ser dono) e EXECUTE na função de máscara
D. SELECT nas tabelas do catálogo saude e MODIFY na função de máscara

R2.3 · Governança e segurança

Na Editora Livro Aberto, a tabela gerenciada prd.vendas.pedidos foi excluída por engano com DROP TABLE há dois dias. O catálogo e o esquema continuam existindo. Como recuperar a tabela com seus metadados e privilégios?

A. DESCRIBE HISTORY prd.vendas.pedidos
B. CREATE TABLE prd.vendas.pedidos LIKE prd.vendas.pedidos
C. RESTORE TABLE prd.vendas.pedidos TO VERSION AS OF 0
D. UNDROP TABLE prd.vendas.pedidos

R2.4 · Governança e segurança

A Rede Farmácias Vida tem duas tabelas no Unity Catalog: vendas_hist, gerenciada, e notas_parceiro, externa, apontando para um bucket que o parceiro também lê. Um analista vai executar DROP TABLE nas duas. Qual é o efeito sobre os arquivos de dados?

A. Os arquivos das duas tabelas continuam no armazenamento, porque DROP TABLE remove só os metadados
B. Os arquivos de vendas_hist são apagados depois do período de recuperação; os de notas_parceiro ficam no bucket
C. Os arquivos das duas tabelas são apagados na hora, porque o Unity Catalog governa as duas localizações
D. Os arquivos de notas_parceiro são apagados; os de vendas_hist ficam no local de armazenamento gerenciado

Respostas comentadas

R2.1. C) ALTER TABLE prd.ops.atracacoes SET MANAGED

Por quê: SET MANAGED copia dados e log de transações para o local gerenciado sem parar leitores e escritores e depois faz uma troca rápida. Acessos por caminho são redirecionados ao novo local, e a conversão pode ser desfeita.

✗ A) Incorreta. MOVE e COPY são usados só na conversão de tabelas estrangeiras; em tabela externa o comando falha com esse sufixo.
✗ B) Incorreta. CTAS cria outra tabela, reescreve todos os dados e não preserva histórico nem permissões; a Databricks recomenda SET MANAGED.
✗ D) Incorreta. Recriar a tabela perde histórico e permissões e deixa a tabela indisponível durante a recarga.

> **Dica de prova:** Externa Delta para gerenciada preservando tudo = ALTER TABLE ... SET MANAGED.

R2.2. C) MANAGE no catálogo saude (ou ser dono) e EXECUTE na função de máscara

Por quê: Requisitos de políticas ABAC de filtro e máscara: Databricks Runtime 16.4 ou superior, ou serverless; UDF com EXECUTE (ou função SQL definida na política); tags governadas nos objetos-alvo; e MANAGE ou propriedade para criar, editar ou excluir.

✗ A) Incorreta. Criar tabelas e aplicar tags não autorizam a criação da política.
✗ B) Incorreta. Políticas são anexadas a catálogo, esquema ou tabela (e, em Beta, ao metastore), sem exigir admin da conta.
✗ D) Incorreta. Ler dados não autoriza criar políticas, e MODIFY não é um privilégio de funções.

> **Dica de prova:** Criar política ABAC = MANAGE (ou dono) no escopo + EXECUTE na UDF.

R2.3. D) UNDROP TABLE prd.vendas.pedidos

Por quê: Após o DROP de uma tabela gerenciada, os arquivos ficam disponíveis durante o período de recuperação. SHOW TABLES DROPPED lista tabelas excluídas, e UNDROP TABLE WITH ID resolve nomes repetidos. Restrições de chave primária e estrangeira não são recuperadas.

✗ A) Incorreta. DESCRIBE HISTORY lista operações de uma tabela existente; não recupera a tabela.
✗ B) Incorreta. CREATE TABLE LIKE copia a definição de uma tabela existente e não traz os dados de volta.
✗ C) Incorreta. RESTORE volta uma tabela existente a uma versão anterior; não recupera uma tabela excluída.

> **Dica de prova:** Tabela excluída por engano = UNDROP TABLE dentro do período de recuperação.

R2.4. B) Os arquivos de vendas_hist são apagados depois do período de recuperação; os de notas_parceiro ficam no bucket

Por quê: Tabela gerenciada: o Unity Catalog controla o ciclo de vida dos arquivos e os exclui após o período de recuperação. Tabela externa: o DROP remove a definição do catálogo e os arquivos continuam no local informado.

✗ A) Incorreta. Isso vale só para a tabela externa; na gerenciada o Unity Catalog também remove os arquivos de dados.
✗ C) Incorreta. Em tabela externa o DROP remove apenas os metadados, e na gerenciada a exclusão dos arquivos ocorre depois do período de recuperação.
✗ D) Incorreta. O comportamento está invertido: quem perde os arquivos é a tabela gerenciada, não a externa.

> **Dica de prova:** DROP em gerenciada apaga dados (após a recuperação); em externa, só metadados.

03

CAPÍTULO 3 · 10% DA PROVA

Implementação de CI/CD

Vale 10% (4 ou 5 questões por prova) e se divide em duas partes: Git folders na interface do workspace e Declarative Automation Bundles (antes Databricks Asset Bundles) com a Databricks CLI. Preciso saber criar e trocar branch, fazer Commit & Push, abrir pull request no provedor, resolver conflitos e entender o que merge, rebase e reset fazem. Nos bundles, cai o ciclo validate, plan, deploy, run e destroy, variáveis por target, precedência das fontes de valor, modos development e production, permissões e autenticação em CI. Onde mais erro é confundir --var (deploy) com --params (run) e achar que reset ou rebase preservam o histórico.

Neste capítulo

16 tópicos-chave · 14 pegadinhas · 6 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Git folders

Cliente Git visual no workspace (antes Repos). Pelo diálogo Git é possível criar branch (Create Branch com nome e branch base), trocar de branch, pull, Commit & Push (commit e push juntos), merge, rebase e reset. O pull request e a revisão acontecem na interface do provedor Git, que aplica as proteções de branch.

Merge, rebase e reset no Git folder

Merge usa git merge e não reescreve o histórico (recomendado para iniciantes). Rebase reaplica commits e o Git folder faz push forçado. Reset equivale a reset --hard com push --force: local e remoto viram cópia do outro branch e as alterações, inclusive não commitadas, se perdem.

Conflitos e troca de branch

Conflito de merge se resolve na UI do Git folder, mantendo as linhas desejadas sem os marcadores, ou com Abort. Ao trocar de branch, as alterações não commitadas acompanham a troca se não conflitarem, e ativos ausentes no novo branch podem ser removidos.

Colaboração e Git folder de produção

O branch ativo é do folder, não do usuário: cada desenvolvedor deve ter seu Git folder e seu branch. Para produção que implanta só código, um Git folder criado por admin, atualizado por automação via API e com Can Run para os usuários; para implantar jobs e pipelines com o código, bundles.

Estrutura do bundle

Um, e somente um, databricks.yml na raiz; arquivos de recursos (como resources/*.yml) entram pelo mapeamento include. Targets representam ambientes; só um target pode ter default: true, usado quando -t não é informado.

Comandos do bundle

init cria projeto a partir de template; validate verifica a configuração (--output json mostra valores resolvidos); plan mostra o que seria criado, atualizado ou removido; deploy -t aplica no workspace; run executa job ou pipeline implantado; summary mostra identidade e recursos; destroy exclui de forma permanente o que foi implantado.

generate e deployment bind

databricks bundle generate job --existing-job-id ID --bind cria a configuração a partir de um job existente e o vincula, sem recriar nem perder histórico. deployment bind também vincula; deployment unbind só remove o vínculo e não apaga nada.

Variáveis de bundle

Declaradas em variables e usadas como ${var.nome}, com valor diferente em cada target. São string, a menos que type: complex (único valor válido do campo) para estruturas como definição de cluster. lookup resolve o ID de objetos pelo nome (warehouse, cluster, job, pipeline, service_principal etc.) e falha se não houver ou houver mais de um.

Precedência de valores de variáveis

Do maior para o menor: --var no comando, variáveis de ambiente BUNDLE_VAR_nome, arquivo .databricks/bundle/target/variable-overrides.json, valor no target e default. No CI, exporte BUNDLE_VAR_nome a partir do segredo.

Substituições

Recuperam valores do contexto: ${bundle.name}, ${bundle.target} (use no lugar de ${bundle.environment}), ${workspace.current_user.userName}, ${workspace.host}. Exemplo de root_path por usuário: /Workspace/Users/${workspace.current_user.userName}/.bundle/${bundle.name}/${bundle.target}.

--var x --params

--var e BUNDLE_VAR_ definem variáveis resolvidas no deploy. Para um valor só em uma execução, use databricks bundle run nome_do_job --params chave=valor, que chega às tarefas como job parameter.

mode: development

Prefixa os nomes com [dev usuário], pausa agendamentos e gatilhos, libera execuções concorrentes e desliga o bloqueio de deploy, para iteração rápida.

mode: production

Valida que o branch Git atual é o configurado no target, valida pipelines com development: false e, sem entidade de serviço, exige run_as e permissions explícitos. Recomenda run_as com entidade de serviço; run_as de nível superior e de target não aceita grupos.

Permissões no bundle

O mapeamento permissions de nível superior aplica a todos os recursos do bundle (por exemplo, group_name com level CAN_RUN). O mesmo principal não pode estar no permissions de nível superior e no de um recurso.

Autenticação em CI

Para deploy sem interação humana, OAuth M2M com entidade de serviço e credenciais em variáveis de ambiente do CI; U2M é para uso assistido local. No GitHub Actions, databricks/setup-cli instala a CLI e a federação de identidade de workload evita guardar segredos.

Job implantado por bundle

Fica com campos bloqueados na UI; o procedimento é alterar o bundle e reimplantar. Disconnect from source é saída de urgência, e as mudanças feitas na UI não voltam para o bundle até ele ser atualizado e reimplantado.

Macetes para memorizar

Precedência de variáveis

Comando, Ambiente, Arquivo, Target, Default: quem está mais perto do comando ganha.

Ciclo do bundle

Init, Valida, Planeja, Implanta, Roda, Destrói.

--var x --params

Var vai no deploy; Params vai no play (run).

Merge, rebase, reset

Merge mantém, Rebase reescreve, Reset arrasa.

Modos de deploy

Dev = prefixo e pausa; Prod = branch conferido e entidade de serviço.

Tipos de variável

Complex carrega estrutura; lookup troca nome por ID.

Tabela de memorização: não confunda

Quando é...	...e quando é
Merge Integrar o main sem reescrever histórico nem forçar push; indicado para iniciantes.	**Rebase** Reaplicar commits sobre o branch alvo, aceitando reescrever o histórico com push forçado.
Reset Descartar tudo do branch e alinhá-lo a outro, local e remoto.	**Merge** Trazer mudanças preservando o que já foi feito no branch.
databricks bundle validate Verificar se o YAML é válido e ver variáveis e substituições resolvidas.	**databricks bundle plan** Ver quais recursos o deploy criaria, atualizaria ou removeria, sem mudar nada.
--var no deploy Definir variável de bundle resolvida na implantação.	**--params no run** Passar job parameter para uma única execução.
mode: development Prefixo [dev usuário], agendamentos pausados, concorrência liberada.	**mode: production** Validação do branch, pipelines com development: false e run_as com entidade de serviço.
Variável type: complex Reaproveitar uma estrutura, como definição de cluster, em vários jobs.	**Variável com lookup** Obter o ID de um objeto pelo nome, como um SQL warehouse que muda por workspace.
Git folder de produção Implantar apenas código, com folder atualizado por automação e Can Run.	**Declarative Automation Bundles** Implantar jobs, pipelines e outros recursos junto com o código entre dev, test e prod.

Números que caem

Item	Valor
Targets com default: true em um bundle	**só um**
Arquivo databricks.yml por bundle	**um, na raiz**
Peso do domínio na prova	**10% (4 ou 5 questões por prova)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Rebase não reescreve o histórico e é o método indicado para iniciantes.
Como não cair: É o rebase que reescreve e força o push; para iniciantes, merge.

⚠ **Alerta de prova:** Reset mescla as duas versões ou afeta só o branch local.
Como não cair: Reset substitui o branch e força o push: local e remoto perdem as alterações.

⚠ **Alerta de prova:** Na troca de branch, o não commitado vai para stash ou é descartado.
Como não cair: Não há stash: as alterações acompanham a troca se não conflitarem.

⚠ **Alerta de prova:** Fazer Commit & Push com marcadores de conflito para o provedor resolver.
Como não cair: Os marcadores precisam ser removidos na UI antes de concluir, ou use Abort.

⚠ **Alerta de prova:** Commit direto no main protegido porque o workspace não aplica as regras.
Como não cair: O push passa pelo provedor, que aplica as proteções; use branch próprio e pull request.

⚠ **Alerta de prova:** BUNDLE_VAR_ vence --var, ou o target vence tudo.
Como não cair: Ordem: --var, BUNDLE_VAR_, variable-overrides.json, target, default.

⚠ **Alerta de prova:** Exportar DATABRICKS_CATALOGO para preencher a variável catalogo.
Como não cair: Só variáveis com prefixo BUNDLE_VAR_ preenchem variáveis do bundle.

⚠ **Alerta de prova:** Passar --var no bundle run para mudar o valor de uma execução.
Como não cair: Variáveis são resolvidas no deploy; valor por execução vai em --params.

⚠ **Alerta de prova:** Variável simples com a definição do cluster em uma string JSON.
Como não cair: Estruturas usam type: complex.

⚠ **Alerta de prova:** databricks bundle schema mostra os valores resolvidos do projeto.
Como não cair: schema exibe o JSON Schema genérico; valores resolvidos vêm de validate --output json.

⚠ **Alerta de prova:** deployment unbind apaga jobs e pipelines.
Como não cair: unbind só desfaz o vínculo; remover tudo é bundle destroy, de forma permanente.

⚠ **Alerta de prova:** run_as de nível superior com um grupo para quem pode executar os jobs.
Como não cair: run_as define a identidade de execução e não aceita grupos; quem pode executar vai em permissions com CAN_RUN.

⚠ **Alerta de prova:** Criar um databricks.yml por ambiente ou por subpasta.
Como não cair: Um único databricks.yml na raiz, com targets e include de resources/*.yml.

⚠ **Alerta de prova:** Deploy de CI com token pessoal ou U2M do líder técnico.
Como não cair: CI usa OAuth M2M com entidade de serviço.

Como cai na prova

📝 **Como cai:** Uma necessidade de CLI (validar, prever, implantar, executar, remover, adotar job existente) com quatro comandos bundle.
Como resolver: Associe o verbo ao comando: validar = validate; prever = plan; aplicar = deploy -t; executar = run; remover = destroy; adotar = generate --bind; começar = init. Desconfie de opções inventadas como --default, --prune ou --clean.

📝 **Como cai:** Variável definida em vários lugares e pergunta qual valor vale.
Como resolver: Aplique a ordem --var, BUNDLE_VAR_, variable-overrides.json, target, default e pegue a fonte mais alta presente no cenário.

📝 **Como cai:** Operação no Git folder (branch, conflito, troca, integração com main, PR).
Como resolver: Lembre que tudo é feito pelo diálogo Git, que PR é no provedor, que merge é o seguro e que reset e rebase forçam push.

📝 **Como cai:** Comportamento observado após o deploy em um target.
Como resolver: Prefixo e agendamentos pausados = development; falha por branch ou exigência de run_as = production; campos bloqueados na UI = recurso gerido pelo bundle.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Começar bundle novo com estrutura recomendada	**databricks bundle init**
Conferir YAML e variáveis resolvidas sem implantar	**databricks bundle validate --output json**
Ver o que seria criado, atualizado ou removido	**databricks bundle plan**
Implantar em produção com dev como default	**databricks bundle deploy -t prod**
Valor só para uma execução	**databricks bundle run --params**
Remover tudo o que o bundle implantou	**databricks bundle destroy (permanente)**
Trazer job existente para o bundle sem recriar	**bundle generate job --existing-job-id --bind**
Nome de catálogo diferente por ambiente	**${var.catalogo} com valor em cada target**
ID de warehouse que muda por workspace	**Variável com lookup pelo nome**
Segredo do CI como valor de variável	**BUNDLE_VAR_nome**
Pasta por desenvolvedor, nome do bundle e target	**${workspace.current_user.userName} + ${bundle.name} + ${bundle.target}**
Deploy falhou a partir de outro branch em prod	**mode: production valida o branch do target**
Nomes com [dev usuário] e agendamentos pausados	**mode: development**
Vários desenvolvedores no mesmo Git folder trocando branch	**Um Git folder e um branch por desenvolvedor**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- Precedência: --var > BUNDLE_VAR_ > variable-overrides.json > target > default.
- --var é de deploy; --params é de run.
- Merge preserva o histórico; rebase e reset forçam push, e reset perde alterações.
- Produção: mode: production, run_as com entidade de serviço e OAuth M2M no CI.

Essenciais para a prova (checklist)

- ☐ Sei criar, trocar e integrar branches no Git folder e abrir pull request no provedor.
- ☐ Sei o efeito de merge, rebase e reset e como resolver conflitos.
- ☐ Sei por que cada desenvolvedor deve ter seu Git folder.
- ☐ Sei estruturar um bundle com databricks.yml, include e targets.
- ☐ Sei usar init, validate, plan, deploy, run, summary, destroy e generate --bind.

- ☐ Sei declarar variáveis simples, complex e com lookup e usá-las por target.
- ☐ Sei a ordem de precedência das fontes de valor das variáveis.
- ☐ Sei o que muda entre mode: development e mode: production.
- ☐ Sei configurar permissions de nível superior e run_as com entidade de serviço.
- ☐ Sei qual autenticação usar em CI e em desenvolvimento local.

Questões de revisão do capítulo

R3.1 · Implementação de CI/CD

Um desenvolvedor da Empresa de Logística Ponte Aérea quer usar Reset no Git folder para alinhar seu branch feature-y ao main remoto. Antes de confirmar, qual efeito ele precisa conhecer?

A. Local e remoto de feature-y viram cópias do main, e as alterações dele se perdem
B. Só o branch local é alterado; o feature-y remoto mantém os commits até o próximo pull request
C. As alterações não commitadas são preservadas em stash e reaplicadas depois do reset
D. O reset cria um commit de merge entre feature-y e main, preservando os dois históricos

R3.2 · Implementação de CI/CD

No CI da Fundação Educar, o databricks.yml define a variável warehouse_id com default e com outro valor no target prod. O pipeline de CI também exporta BUNDLE_VAR_warehouse_id e chama databricks bundle deploy -t prod --var="warehouse_id=abc123". Qual valor será usado?

A. abc123, porque o valor passado em --var tem a maior precedência
B. O valor de BUNDLE_VAR_warehouse_id, porque variáveis de ambiente vencem --var
C. O default da variável, porque ele é lido antes das demais fontes
D. O valor definido no target prod, porque o target sobrepõe outras fontes

R3.3 · Implementação de CI/CD

No GitHub Actions da Agência de Dados Prisma, o mesmo workflow implanta o bundle em staging e prod, e o nome do catálogo vem de um segredo diferente para cada ambiente. Sem editar o YAML a cada execução, como o CI pode fornecer o valor da variável catalogo do bundle?

A. Gravar o valor do segredo no default da variável catalogo dentro do databricks.yml
B. Exportar BUNDLE_VAR_catalogo com o valor do segredo antes de rodar databricks bundle deploy
C. Passar o valor do segredo em --params no databricks bundle deploy de cada ambiente
D. Exportar DATABRICKS_CATALOGO com o valor do segredo antes de rodar o comando databricks bundle deploy

R3.4 · Implementação de CI/CD

No bundle da Editora Saber Mais, cada desenvolvedor deve implantar em uma pasta própria no workspace, e o caminho precisa incluir o nome do bundle e o target. Qual valor de workspace.root_path atende sem escrever o nome de ninguém no YAML?

A. /Workspace/Users/${var.usuario}/.bundle/${bundle.name}/${bundle.environment}
B. /Workspace/Shared/.bundle/${bundle.name}/${bundle.target}
C. /Workspace/Users/${workspace.host}/.bundle/${resources.jobs.id}
D. /Workspace/Users/${workspace.current_user.userName}/.bundle/${bundle.name}/${bundle.target}

Respostas comentadas

R3.1. A) Local e remoto de feature-y viram cópias do main, e as alterações dele se perdem

Por quê: Reset é útil quando as edições conflitam com o branch de origem e podem ser descartadas. Para integrar mudanças preservando o histórico, use merge.

✗ B) Incorreta. O reset no Git folder também força o push e altera o branch remoto.

✗ C) Incorreta. Não há stash: alterações não commitadas também são perdidas.

✗ D) Incorreta. Esse é o comportamento de um merge, não de um reset.

> **Dica de prova:** Reset no Git folder = reset --hard + push --force: perde tudo do branch.

R3.2. A) abc123, porque o valor passado em --var tem a maior precedência

Por quê: A ordem de precedência das variáveis de bundle é: --var no comando, variáveis de ambiente BUNDLE_VAR_, .databricks/bundle/target/variable-overrides.json, variables do target e default. Use a mesma forma no deploy e no run.

✗ B) Incorreta. Variáveis de ambiente BUNDLE_VAR_ vêm em segundo lugar, depois de --var.

✗ C) Incorreta. O default é a última opção, usada só quando nenhuma outra fonte define o valor.

✗ D) Incorreta. O valor do target fica abaixo de --var, das variáveis de ambiente e do arquivo variable-overrides.json.

> **Dica de prova:** Precedência: --var > BUNDLE_VAR_ > overrides.json > target > default.

R3.3. B) Exportar BUNDLE_VAR_catalogo com o valor do segredo antes de rodar databricks bundle deploy

Por quê: Valores de variáveis podem vir de --var, de BUNDLE_VAR_, de variable-overrides.json, do target ou do default, nessa ordem de precedência.

✗ A) Incorreta. Gravar segredos no YAML os expõe e exige editar o arquivo por ambiente.

✗ C) Incorreta. --params passa job parameters no bundle run, não variáveis no deploy.

✗ D) Incorreta. Só variáveis de ambiente com o prefixo BUNDLE_VAR_ preenchem variáveis do bundle.

> **Dica de prova:** Variável de bundle vinda do CI = BUNDLE_VAR_nome.

R3.4. D) /Workspace/Users/${workspace.current_user.userName}/.bundle/${bundle.name}/${bundle.target}

Por quê: Substituições recuperam valores do contexto da implantação, como nome do bundle, target, host e usuário atual. Use databricks bundle validate para conferir os valores resolvidos.

✗ A) Incorreta. Exigiria definir o usuário em uma variável, e a documentação orienta usar ${bundle.target} em vez de ${bundle.environment}.

✗ B) Incorreta. Uma pasta compartilhada faria todos os desenvolvedores implantarem no mesmo lugar.

✗ C) Incorreta. ${workspace.host} é o endereço do workspace, e a referência ao ID de job está incompleta.

> **Dica de prova:** Caminho por usuário e ambiente = ${workspace.current_user.userName} + ${bundle.target}.

04

CAPÍTULO 4 · 21% DA PROVA

Ingestão e carga de dados

Este é um dos domínios mais pesados (21%, cerca de 9 questões por prova), e eu preciso escolher a ferramenta certa de ingestão e conhecer as opções de COPY INTO e Auto Loader de cor. A prova cobra a decisão entre conector gerenciado do Lakeflow Connect, conector padrão, Auto Loader, COPY INTO, Lakehouse Federation, conexão JDBC e notebook com cliente REST, sempre a partir de um requisito (sem código, sem copiar, gravar de volta, milhões de arquivos, sem CDC na origem). Também cai muito o comportamento exato dos modos de evolução de esquema, de _rescued_data, de schemaHints, de includeExistingFiles e das opções force, mergeSchema e VALIDATE. Onde mais erro é trocar uma opção do Auto Loader por outra de nome parecido e esquecer que o COPY INTO é idempotente por padrão.

Neste capítulo

18 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Camadas de ingestão

Da mais customizável para a mais gerenciada: Structured Streaming, Lakeflow pipelines e conectores gerenciados. A orientação é começar pela camada mais gerenciada que atenda à fonte e descer só se ela não cumprir os requisitos; os gerenciados já tratam autenticação, CDC, evolução de esquema e novas tentativas.

Conectores gerenciados do Lakeflow Connect

Configurados sem código e gravando em streaming tables com governança do Unity Catalog. Há conectores SaaS (como Salesforce, Workday, ServiceNow), de banco de dados (MySQL, PostgreSQL, SQL Server, Oracle), de arquivos (como SharePoint e SFTP) e de streaming (Kafka e RabbitMQ). Kinesis, Pub/Sub e Pulsar usam Structured Streaming ou pipelines.

Conector gerenciado x conector padrão

O conector de arquivos gerenciado é configurado sem código de leitura. O conector padrão usa as APIs de leitura do Spark e SQL e dá controle fino, mas exige escrever a leitura.

Componentes do conector de banco de dados

Conexão do Unity Catalog (credenciais), ingestion gateway (computação clássica e contínua, que captura mudanças antes de os logs serem truncados), staging storage em volume (expurgo após 30 dias) e pipeline de ingestão serverless que aplica as mudanças às streaming tables. O gateway explica o custo clássico mesmo com o pipeline parado.

CDC e conector baseado em consulta

O conector de SQL Server usa change tracking ou CDC habilitados na origem (não suportados em réplicas de leitura). Sem binlog ou CDC, o conector baseado em consulta usa uma coluna cursor (como atualizado_em), roda por agendamento, gera mais carga na origem e captura só o estado mais recente das linhas alteradas.

History tracking (SCD type 2)

Nos conectores gerenciados que suportam, ligar History tracking (ou scd_type = SCD_TYPE_2) preserva o histórico de alterações. O padrão é SCD type 1, que sobrescreve as linhas.

Operação dos conectores gerenciados

Cada agendamento de um pipeline de ingestão cria um job com o pipeline como tarefa. Em falha, o conector repete com backoff exponencial e guarda a posição do cursor para retomar dali. O custo aparece em system.billing.usage com billing_origin_product = LAKEFLOW_CONNECT. Conexões podem ser criadas por automação com databricks connections create --json.

COPY INTO

Carrega arquivos de armazenamento de objetos ou volumes para tabela Delta e é idempotente: registra os arquivos carregados e ignora os já vistos, mesmo após DELETE na tabela. COPY_OPTIONS ('force' = 'true') recarrega; COPY_OPTIONS ('mergeSchema' = 'true') evolui o esquema; FORMAT_OPTIONS repassa opções ao leitor do formato.

Cláusulas do COPY INTO

VALIDATE (ALL ou n ROWS) testa parsing, esquema e restrições sem gravar. PATTERN usa glob para selecionar arquivos por nome; FILES lista nomes exatos (até 1000); os dois são mutuamente exclusivos. FROM (SELECT ... FROM 'caminho') converte tipos e escolhe colunas na carga. Com external location e READ FILES, não precisa de credencial inline.

Auto Loader (cloudFiles)

Ingestão incremental de arquivos de armazenamento de objetos e volumes, com descoberta que escala com o número de arquivos ingeridos, não de diretórios. Os metadados dos arquivos descobertos ficam em RocksDB no checkpoint, garantindo exatamente uma vez; cada stream precisa do próprio checkpoint, mesmo gravando na mesma tabela.

Esquema no Auto Loader

Fora de pipelines, cloudFiles.schemaLocation liga a inferência e a evolução do esquema; em Lakeflow pipelines, esquema e checkpoint são gerenciados sozinhos. JSON e CSV são inferidos como STRING por padrão; cloudFiles.inferColumnTypes = true infere tipos exatos. cloudFiles.schemaHints força o tipo de algumas colunas e mantém as demais inferidas.

schemaEvolutionMode

addNewColumns (padrão sem esquema informado): adiciona a coluna e falha com UnknownFieldException, retomando ao reiniciar. rescue: não para e manda o inesperado para _rescued_data. failOnNewColumns: para e exige intervenção. none (padrão com .schema informado): ignora a coluna nova. Há também addNewColumnsWithTypeWidening.

_rescued_data

Coluna adicionada quando o Auto Loader infere o esquema; guarda em JSON os campos que não couberam (como tipos incompatíveis) e o caminho do arquivo de origem. Registros corrompidos não vão para ela (com badRecordsPath, vão para esse caminho).

Detecção de arquivos

Directory listing é o padrão, sem permissões extras. File notification com file events habilitados na external location reduz custo e tempo de descoberta em buckets enormes; file events beneficiam todos os streams e gatilhos de chegada de arquivo no mesmo local e exigem ser dono ou ter MANAGE na external location.

Opções de lote e arquivos do Auto Loader

cloudFiles.maxFilesPerTrigger e cloudFiles.maxBytesPerTrigger limitam o micro-lote (maxBytes é flexível; um arquivo nunca é dividido). cloudFiles.includeExistingFiles = false ignora o histórico, mas só vale na primeira execução. cloudFiles.partitionColumns captura colunas de partição do caminho chave=valor.

trigger(availableNow=True)

Processa os arquivos disponíveis e termina, preservando o checkpoint para a próxima execução agendada. Em notebooks e jobs serverless só AvailableNow e Once (obsoleto) são suportados; sem trigger, o padrão roda sem parar.

Lakehouse Federation x conexão JDBC

Federation consulta bancos suportados sem copiar, com catálogo estrangeiro, governança por tabela e pushdown, mas não aceita escrita. A conexão JDBC do Unity Catalog, usada pela Spark Data Source API, serve para gravar no banco, usar driver específico ou acessar banco não suportado.

Semiestruturados e não estruturados

JSON variável e aninhado vai para coluna VARIANT com parse_json. JSON em STRING é lido com coluna:campo.subcampo::tipo; from_json com esquema gera STRUCT tipado. Imagens e PDFs entram com cloudFiles.format = binaryFile. JSON com registro em várias linhas pede multiline = true. Kafka entrega value binário: cast('string') e depois from_json.

Macetes para memorizar

Modos de evolução do Auto Loader

Add adiciona e cai; Rescue resgata e segue; Fail falha e espera; None nem vê.

Padrão do schemaEvolutionMode

Sem esquema = addNewColumns; com .schema = none.

COPY INTO

Force reforça a carga; Merge mexe no esquema; Validate vê sem gravar.

Federation x JDBC

Federation Faz só leitura; JDBC Joga dados de volta.

Escolha de ingestão

Desça a escada: gerenciado primeiro, pipeline depois, Structured Streaming por último.

Opções includeExistingFiles

Só na estreia: muda com checkpoint novo ou não muda.

Kafka e streaming gerenciado

Kafka e Rabbit têm casa (conector gerenciado); Kinesis, Pub/Sub e Pulsar fazem na mão.

Tabela de memorização: não confunda

Quando é...	...e quando é
COPY INTO Carga idempotente e agendada de arquivos em volume moderado, direto em SQL.	**Auto Loader** Diretórios com milhões de arquivos, streaming incremental, evolução de esquema e file notification.

Lakehouse Federation Consultar banco suportado sem copiar, com governança por tabela; só leitura.	**Conexão JDBC do Unity Catalog** Precisa gravar de volta no banco, usar driver específico ou acessar banco não suportado pela federação.
Conector gerenciado do Lakeflow Connect Fonte suportada (Salesforce, Workday, SharePoint, bancos): mínimo de manutenção, incremental e retries prontos.	**Notebook com cliente REST em job** API REST sem conector, volume pequeno: requests + spark.createDataFrame, agendado em Lakeflow Jobs.
schemaEvolutionMode rescue O stream nunca pode parar e o inesperado vai para _rescued_data sem mudar o esquema.	**schemaEvolutionMode failOnNewColumns** Toda coluna nova deve parar o stream até revisão humana do esquema.
cloudFiles.schemaHints Fixar o tipo de algumas colunas e inferir o resto; addNewColumns continua permitido.	**.schema(...)** Esquema completo informado: desliga a inferência e o modo padrão passa a ser none.
COPY_OPTIONS force = true Recarregar de propósito arquivos já carregados (de preferência com PATTERN ou FILES).	**COPY_OPTIONS mergeSchema = true** Adicionar automaticamente colunas novas que vieram nos arquivos.
Conector de banco com CDC A origem permite binlog, change tracking ou CDC; o gateway captura todas as mudanças.	**Conector baseado em consulta** Não é permitido habilitar CDC, mas existe coluna de alteração confiável usada como cursor.

Números que caem

Item	Valor
Upload de arquivo para volume pela interface	**até 5 GB**
Expurgo do staging do conector de banco	**após 30 dias**
Lista FILES do COPY INTO	**até 1000 nomes de arquivo**
COPY INTO VALIDATE	**Databricks SQL ou Databricks Runtime 10.4 LTS ou superior**
COPY INTO em tabela sem esquema	**Databricks Runtime 11.3 LTS ou superior**
Tipo VARIANT	**Databricks Runtime 15.4 ou superior**
Conector SFTP em computação clássica	**Databricks Runtime 17.3 ou superior**
maxFilesPerTrigger ajustado dinamicamente	**a partir do Databricks Runtime 18.0**
Peso do domínio na prova	**21% (cerca de 9 das 45 questões)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** COPY INTO reexecutado na mesma pasta duplica linhas.
Como não cair: É idempotente por padrão: arquivos já carregados são ignorados; só force = true recarrega.

⚠ **Alerta de prova:** Usar PATTERN ou mergeSchema para recarregar arquivos apagados por DELETE.
Como não cair: Arquivos já carregados continuam ignorados; o que permite recarregar é COPY_OPTIONS ('force' = 'true').

⚠ **Alerta de prova:** Testar a carga com FORMAT_OPTIONS ('mode' = 'FAILFAST') sem gravar.
Como não cair: FAILFAST só falha em registro malformado; se não houver erro, grava. Testar sem gravar é VALIDATE.

⚠ **Alerta de prova:** Informar o esquema completo com .schema(...) e esperar que a coluna nova vá para _rescued_data.
Como não cair: Com esquema informado o padrão é none: a coluna é ignorada, sem resgate se rescuedDataColumn não estiver definida.

⚠ **Alerta de prova:** No modo padrão, o stream segue rodando e grava a coluna nova em _rescued_data.
Como não cair: O padrão sem esquema é addNewColumns: inclui a coluna e falha com UnknownFieldException; retoma ao reiniciar.

⚠ **Alerta de prova:** Usar cloudFiles.inferColumnTypes com uma lista de tipos.
Como não cair: inferColumnTypes recebe true ou false; tipos por coluna vão em cloudFiles.schemaHints.

⚠ **Alerta de prova:** Aumentar maxFilesPerTrigger para listar mais rápido um bucket enorme.
Como não cair: Esse limite muda o tamanho do lote, não o custo de descoberta; use file notification com file events.

⚠ **Alerta de prova:** Trocar includeExistingFiles a cada reinício do stream.
Como não cair: A opção só vale na primeira execução, com checkpoint novo; depois o checkpoint manda.

⚠ **Alerta de prova:** Compartilhar o checkpointLocation entre dois streams que gravam na mesma tabela.
Como não cair: Cada stream precisa do próprio checkpoint; compartilhar mistura estados e não é suportado.

⚠ **Alerta de prova:** Federation para gravar o resumo diário de volta no banco.
Como não cair: Federation é só leitura; para escrita use conexão JDBC do Unity Catalog.

⚠ **Alerta de prova:** Apontar Auto Loader ou COPY INTO para URL de API, SharePoint ou servidor SFTP.
Como não cair: Ambos leem armazenamento de objetos e volumes; SharePoint e SFTP têm conectores próprios, API REST vai por notebook.

⚠ **Alerta de prova:** Conector de banco com CDC quando não é permitido habilitar binlog.
Como não cair: Sem CDC na origem, use o conector baseado em consulta com coluna cursor, aceitando mais carga na origem.

⚠ **Alerta de prova:** Rodar vários COPY INTO em paralelo para ganhar desempenho em diretório gigante.
Como não cair: Paralelizar não é técnica de desempenho; para milhões de arquivos, use Auto Loader ou streaming table com read_files.

⚠ **Alerta de prova:** Guardar JSON variável como STRING e extrair com get_json_object.
Como não cair: A recomendação é VARIANT com parse_json; em STRING já existente, use coluna:campo::tipo.

Como cai na prova

📝 **Como cai:** Cenário com fonte e requisitos (sem código, sem copiar, gravar de volta, sem CDC) pedindo o método de ingestão.
Como resolver: Identifique o tipo da fonte (SaaS, banco, arquivos em objeto, SFTP, API, barramento). Depois aplique o requisito decisivo: sem código = gerenciado; sem copiar = federation; gravar = JDBC; sem conector = notebook em job.

📝 **Como cai:** Stream de Auto Loader com uma configuração e chega uma coluna nova; o que acontece?
Como resolver: Veja primeiro se o esquema foi informado (.schema = none) ou inferido com schemaLocation (addNewColumns), depois se há schemaEvolutionMode explícito. Aplique o comportamento exato do modo.

📝 **Como cai:** Comando COPY INTO com um problema (duplicar, recarregar, coluna nova, filtrar arquivos, converter tipo).
Como resolver: Mapeie o verbo: recarregar = force; coluna nova = mergeSchema; testar = VALIDATE; filtrar nome = PATTERN; converter ou descartar coluna = FROM (SELECT ...).

📝 **Como cai:** Opção do cloudFiles com nome parecido entre as alternativas.
Como resolver: Associe cada opção ao seu único efeito: schemaLocation (esquema), schemaHints (tipos de algumas colunas), inferColumnTypes (tipos exatos), partitionColumns (partições do caminho), includeExistingFiles (histórico na estreia), maxFiles/maxBytes (tamanho do lote).

📝 **Como cai:** Dado semiestruturado ou binário a ser lido ou armazenado.
Como resolver: JSON variável = VARIANT; STRING JSON consultada = dois-pontos; STRUCT tipado = from_json; várias linhas = multiline; binário = binaryFile; Kafka = cast para string.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Fonte SaaS suportada com mínimo de manutenção	**Conector gerenciado do Lakeflow Connect (streaming tables)**
Custo clássico contínuo com o pipeline de ingestão parado	**Ingestion gateway clássico sempre ligado**
Diretório com milhões de arquivos e descoberta lenta	**Auto Loader com file notification e file events**
Testar a carga sem gravar nada	**COPY INTO ... VALIDATE**
Só arquivos cujo nome começa com um prefixo	**PATTERN com glob**
Parar e exigir revisão humana a cada coluna nova	**failOnNewColumns**
Nunca parar e guardar o inesperado sem mudar o esquema	**rescue**
Ler só arquivos que chegarem a partir de agora	**cloudFiles.includeExistingFiles = false na primeira execução**
Processar o disponível a cada 4 horas e terminar, em serverless	**trigger(availableNow=True)**
Consultar PostgreSQL sem copiar, com acesso por tabela	**Lakehouse Federation (catálogo estrangeiro)**
Gravar de volta no banco externo pelo Spark	**Conexão JDBC do Unity Catalog**
Fotos, áudio ou PDFs com conteúdo e metadados	**cloudFiles.format = binaryFile**
Coluna value do Kafka em bytes ilegíveis	**cast('string') e from_json**
Analistas SQL querendo ingestão incremental escalável	**CREATE OR REFRESH STREAMING TABLE ... FROM STREAM read_files(...)**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- COPY INTO é idempotente; force recarrega, mergeSchema evolui, VALIDATE não grava.
- Auto Loader: sem esquema o padrão é addNewColumns (falha e retoma); com .schema é none.
- Cada stream tem seu checkpoint; em Lakeflow pipelines checkpoint e esquema são automáticos.
- Federation só lê; para gravar no banco, conexão JDBC do Unity Catalog.
- Comece pela camada de ingestão mais gerenciada que atenda à fonte.

Essenciais para a prova (checklist)

- ☐ Sei escolher entre conector gerenciado, conector padrão, Auto Loader, COPY INTO, Federation, JDBC e notebook REST.
- ☐ Sei os componentes do conector de banco e por que o gateway gera custo contínuo.
- ☐ Sei quando usar o conector baseado em consulta e o que é history tracking.
- ☐ Sei o efeito de force, mergeSchema, VALIDATE, PATTERN, FILES e SELECT no COPY INTO.
- ☐ Sei o comportamento dos cinco modos de evolução de esquema e qual é o padrão em cada caso.
- ☐ Sei o que vai e o que não vai para _rescued_data.
- ☐ Sei para que servem schemaLocation, schemaHints, inferColumnTypes, partitionColumns, includeExistingFiles e maxFilesPerTrigger.
- ☐ Sei a diferença entre directory listing e file notification com file events.
- ☐ Sei ler JSON com VARIANT, dois-pontos, from_json e multiline, e Kafka com cast para string.
- ☐ Sei os privilégios para fazer upload em volume (WRITE VOLUME, USE SCHEMA, USE CATALOG).

Questões de revisão do capítulo

R4.1 · Ingestão e carga de dados

Na Rede de Postos Rota Verde, os CSVs de abastecimento trazem a coluna litros como texto e uma coluna de observações que não deve ir para a bronze. A equipe quer converter litros para DOUBLE e descartar a coluna extra já na carga com COPY INTO. Como fazer?

A. Usar PATTERN = 'litros*.csv' para que o COPY INTO carregue só a coluna litros como DOUBLE
B. Usar FROM (SELECT cast(litros AS DOUBLE) AS litros, ... FROM 'caminho') no COPY INTO
C. Usar COPY_OPTIONS ('mergeSchema' = 'true') para o COPY INTO converter o tipo de litros
D. Usar FORMAT_OPTIONS ('inferSchema' = 'true') e remover a coluna extra com DELETE depois

R4.2 · Ingestão e carga de dados

A Rede Lojas Ponto Certo quer ingerir uma tabela de um banco MySQL em que não é permitido habilitar binlog nem outra infraestrutura de CDC. A tabela tem a coluna atualizado_em, preenchida em toda inserção e atualização. Qual opção do Lakeflow Connect atende e qual é o cuidado?

A. Conector baseado em consulta, com atualizado_em como cursor e mais carga na origem
B. Auto Loader lendo o diretório de dados do MySQL; ele detecta as linhas alteradas pela coluna de data
C. Conector de banco com CDC e ingestion gateway; o gateway lê o binlog continuamente sem afetar a origem
D. Conector SaaS gerenciado apontando para o MySQL; ele sincroniza o banco sem consultar a origem

R4.3 · Ingestão e carga de dados

Na Empresa de Logística Caminho Seguro, um stream lê arquivos com spark.readStream.format("json").load(caminho) em uma pasta com milhões de arquivos espalhados em muitas subpastas, e a descoberta ficou lenta e cara. Qual troca a Databricks recomenda e por quê?

A. Usar Auto Loader (cloudFiles), cuja descoberta escala com o número de arquivos ingeridos e não com o de diretórios
B. Usar spark.read em lote a cada hora, que relê só os arquivos alterados desde a última leitura
C. Usar o mesmo file source com maxFilesPerTrigger menor, que reduz o custo de listar os diretórios
D. Usar COPY INTO agendado, que lista a pasta inteira a cada execução e mantém o estado na tabela de destino

R4.4 · Ingestão e carga de dados

A Cia. de Energia Ventos do Sul vai ingerir dados de uma fonte que tem conector gerenciado no Lakeflow Connect, mas um engenheiro sugere escrever Structured Streaming do zero para ter controle total. Qual é a orientação da Databricks para escolher a camada de ingestão?

A. Começar pela camada mais gerenciada e descer de camada só se ela não atender aos requisitos
B. Começar pelo Structured Streaming, que permite controlar cada detalhe da ingestão
C. Começar por COPY INTO, que oferece o maior nível de automação para fontes SaaS e bancos
D. Começar por Lakeflow pipelines e evitar conectores gerenciados em fontes corporativas

Respostas comentadas

R4.1. B) Usar FROM (SELECT cast(litros AS DOUBLE) AS litros, ... FROM 'caminho') no COPY INTO

Por quê: No SELECT do COPY INTO valem expressões comuns de SELECT, inclusive funções de janela; agregações só como agregados globais, sem GROUP BY.

✗ A) Incorreta. PATTERN filtra arquivos pelo nome; não seleciona colunas nem converte tipos.
✗ C) Incorreta. mergeSchema permite evoluir o esquema da tabela; não converte tipos nem descarta colunas.
✗ D) Incorreta. DELETE remove linhas, não colunas; e a inferência não garante o tipo desejado.

> **Dica de prova:** Converter e escolher colunas durante o COPY INTO = FROM (SELECT ... FROM origem).

R4.2. A) Conector baseado em consulta, com atualizado_em como cursor e mais carga na origem

Por quê: Os conectores baseados em consulta rodam por agendamento, gravam em streaming tables e capturam só o estado mais recente das linhas alteradas entre execuções. Suportam SCD tipo 1, SCD tipo 2 e APPEND_ONLY.

✗ B) Incorreta. Auto Loader lê arquivos em armazenamento de objetos, não arquivos internos de um banco.
✗ C) Incorreta. O conector com CDC depende justamente do binlog, que não pode ser habilitado.
✗ D) Incorreta. Conectores SaaS atendem aplicações como Salesforce e Workday, não bancos relacionais.

> **Dica de prova:** Sem CDC, mas com coluna de alteração confiável = conector baseado em consulta (cursor).

R4.3. A) Usar Auto Loader (cloudFiles), cuja descoberta escala com o número de arquivos ingeridos e não com o de diretórios

Por quê: Além de escalar melhor, o Auto Loader usa APIs nativas de listagem da nuvem e pode usar file notification para evitar listagens, reduzindo custo.

✗ B) Incorreta. Leitura em lote não acompanha quais arquivos já foram processados.
✗ C) Incorreta. Limitar arquivos por lote não muda o custo de descoberta do file source.
✗ D) Incorreta. Para diretórios com número muito grande de arquivos, a Databricks recomenda Auto Loader em vez de COPY INTO.

> **Dica de prova:** Muitos arquivos em streaming de arquivos = Auto Loader.

R4.4. A) Começar pela camada mais gerenciada e descer de camada só se ela não atender aos requisitos

Por quê: As três camadas de ingestão, da mais customizável para a mais gerenciada, são Structured Streaming, Lakeflow pipelines e conectores gerenciados. Os conectores gerenciados acrescentam autenticação específica da fonte, CDC, evolução automática de esquema e novas tentativas.

✗ B) Incorreta. Structured Streaming é a camada mais customizável, mas exige que a equipe trate autenticação, CDC, esquema e novas tentativas.

✗ C) Incorreta. COPY INTO carrega arquivos de armazenamento de objetos; não se conecta a aplicações SaaS nem a bancos.

✗ D) Incorreta. Os conectores gerenciados são justamente a camada recomendada quando a fonte é suportada.

> **Dica de prova:** Escolha de ingestão: comece pelo mais gerenciado; desça só se precisar.

05

CAPÍTULO 5 · 10% DA PROVA

Solução de problemas, monitoramento e otimização

Vale 10% (4 ou 5 questões por prova) e cobra diagnóstico: ler a Spark UI (skew, spill, shuffle, estágio com uma tarefa, lacunas na timeline), usar a UI de Lakeflow Jobs (aba Runs, matrix view, Timeline, status, Repair run), resolver falha de partida de cluster, conflito de biblioteca e falta de memória, e entender liquid clustering e predictive optimization. O enunciado dá um sintoma concreto e quatro diagnósticos plausíveis; a resposta é a que bate exatamente com o sintoma. Onde mais erro é confundir skew com estágio de uma tarefa só e achar que mais workers resolvem tudo.

Neste capítulo

17 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Roteiro na Spark UI

Comece pela timeline de jobs, vá ao estágio mais longo e registre Input (lido do armazenamento), Output (gravado), Shuffle Read e Shuffle Write (troca entre executores) e o número de tarefas. Depois verifique skew, spill e se o estágio é limitado por I/O; meça de novo após cada ajuste.

Skew

Nas Summary Metrics do estágio, Max muito maior que o percentil 75 de duração (e de Shuffle Read Size) indica skew: poucas tarefas sobrecarregadas e núcleos ociosos. Correções: conferir o tratamento de skew join do AQE, ajustar seus limites, salting ou pré-agregação, liquid clustering na origem ou hint REBALANCE na escrita.

Spill

Ocorre quando a memória de execução (shuffles, joins, sorts, agregações) se esgota e o Spark grava em disco; aparece nas estatísticas de spill no topo da página do estágio. Primeira ação: spark.sql.shuffle.partitions como auto ou instâncias com mais memória.

Estágio com uma única tarefa

Causas: janela sem PARTITION BY, UDF cara sobre poucos dados, arquivo não divisível como gzip, multiLine em JSON ou CSV, inferência de esquema em arquivo grande, repartition(1) ou coalesce(1). Mais workers não ajudam porque o Spark não consegue distribuir.

Lacunas na timeline

Minutos sem jobs Spark com workers ociosos e driver ocupado indicam código não Spark no driver (laço Python sobre lista coletada), planos complexos (withColumn em laço) ou driver sobrecarregado. Reescreva com Spark ou use serverless quando o código não Spark for necessário.

Hint REBALANCE

Equilibra partições de saída na escrita de tabelas, evitando arquivos grandes demais e pequenos demais. MERGE e BROADCAST são hints de join; COALESCE(1) concentra tudo em uma partição.

Falta de memória (OOM)

ExecutorLostFailure é genérico. Antes de suspeitar de memória, confira no event log se executores sumiram por autoscaling ou perda de instâncias spot. Para confirmar OOM, troque o worker por um com o dobro de memória por núcleo: o que importa é a razão memória por núcleo, não a memória total.

Falha de partida de cluster

O motivo está no event log da computação, pelo código de término. Causas comuns: capacidade de instâncias, cotas, IPs livres na sub-rede e rede. Para AWS_INSUFFICIENT_INSTANCE_CAPACITY_FAILURE: outra zona ou auto AZ, outro tipo de instância e flexible node types (habilitados pelo admin, não valem para GPU).

Conflito de bibliotecas

Evite init scripts e bibliotecas no escopo da computação; instale com %pip no notebook ou em environment, isolados na sessão. Depois de %pip pode ser preciso dbutils.library.restartPython(). Runtime ML só quando necessário e instalação a partir do DBFS é legado.

Repair run e status de execução

Repair run (jobs com duas ou mais tarefas) reexecuta só as tarefas que falharam e suas dependentes; cada uma começa do zero, então tarefas devem ser idempotentes. Status: Succeeded, Succeeded with failures (falhas no meio, folhas com sucesso; conta como sucesso para notificações), Failed, Skipped, Timed Out e Canceled.

Monitoramento na UI de Lakeflow Jobs

Aba Runs de Jobs & Pipelines: últimos 60 dias, filtros e gráfico com os 5 erros mais frequentes. Matrix view na aba Runs do job compara duração de cada execução e tarefa. Na página Job run details, as visões Graph, Timeline (duração e sobreposição, caminho crítico) e List.

Histórico longo e query profile

A UI guarda execuções por 60 dias; system.lakeflow (como job_run_timeline) guarda 365 dias e pode ser cruzada com cobrança. Para consultas em SQL warehouse, o query profile (Top operators) aberto pelo Query History mostra o gargalo por operador; exige ser dono ou CAN MONITOR, e consultas do cache não têm profile.

Limites de duração

Em Duration and streaming backlog thresholds: Warning dispara o evento de duração excedida usado pela notificação Duration warning sem parar o job; Timeout encerra a execução como Timed Out.

Liquid clustering

Substitui particionamento e ZORDER (não é compatível com eles) e permite redefinir chaves sem reescrever dados. Até quatro chaves, que precisam ter estatísticas coletadas (por padrão, das 32 primeiras colunas). OPTIMIZE agrupa os dados novos de forma incremental; OPTIMIZE ... FULL reclusteriza o que já existia.

Conversão e CLUSTER BY AUTO

ALTER TABLE ... CLUSTER BY habilita clustering em tabela não particionada; tabela particionada converte com ALTER TABLE ... REPLACE PARTITIONED BY WITH CLUSTER BY (Runtime 18.1 ou superior), de preferência com chaves parecidas com as partições. CLUSTER BY AUTO deixa a escolha de chaves para a predictive optimization em tabelas gerenciadas.

Predictive optimization

Roda OPTIMIZE, VACUUM e ANALYZE automaticamente em tabelas gerenciadas do Unity Catalog, cobrada no SKU de serverless jobs; não executa ZORDER nem cobre hive_metastore. Liga-se com ALTER CATALOG ou SCHEMA ... ENABLE PREDICTIVE OPTIMIZATION (dono ou MANAGE), com herança ENABLE, DISABLE ou INHERIT; DESCRIBE ... EXTENDED mostra o estado efetivo.

Arquivos pequenos

Leitura dominada por dezenas de milhares de arquivos pequenos pede OPTIMIZE, predictive optimization e revisão do layout (arquivos de pelo menos 8 MB; liquid clustering no lugar de partição de alta cardinalidade). VACUUM remove arquivos não referenciados e não compacta; ANALYZE coleta estatísticas.

Macetes para memorizar

Skew na Spark UI

Max longe do p75 = skew; uma tarefa só = janela, gzip ou coalesce.

Onde olhar cada coisa

Cluster não sobe ou executor some: event log. Consulta SQL lenta: query profile. Tarefa Spark lenta: Spark UI. Histórico de jobs: Runs, matrix ou system.lakeflow.

OOM

Memória por núcleo, não memória total.

Predictive optimization

OVA automático: Optimize, Vacuum, Analyze, só em gerenciadas do UC.

Liquid clustering

Quatro chaves no máximo, FULL para o passado, AUTO para delegar.

Visões da execução

Matrix compara execuções; Timeline mostra uma execução por dentro.

Bibliotecas

Conflito de pacote se cura com %pip no notebook, não com init script.

Tabela de memorização: não confunda

Quando é...	...e quando é
Skew Muitas tarefas, poucas muito mais lentas (Max muito acima do p75): AQE skew join, salting, REBALANCE.	**Estágio com uma única tarefa** O estágio inteiro tem uma tarefa: janela sem PARTITION BY, gzip, multiLine ou coalesce(1).

Spill Estatísticas de spill no estágio sem sinal de desequilíbrio: shuffle partitions auto ou mais memória.	**Skew** Uma tarefa muito acima das demais em duração e shuffle read.
Matrix view Comparar duração de execuções e tarefas de um job ao longo do tempo.	**Visão Timeline do Job run details** Ver, em uma execução, quais tarefas demoraram e como se sobrepuseram.
UI de Lakeflow Jobs Histórico de até 60 dias, com filtros e erros mais frequentes.	**Tabelas de sistema system.lakeflow** Histórico de 365 dias, consultável em SQL e cruzável com cobrança.
OPTIMIZE Clusterizar de forma incremental os dados novos.	**OPTIMIZE ... FULL** Reclusterizar também os dados gravados antes de habilitar o clustering.
Repair run Recuperar a execução refazendo só a tarefa que falhou e as dependentes.	**Run now** Começar uma execução nova que refaz todas as tarefas.
Duration Warning Avisar quando passar do limite sem interromper.	**Timeout** Encerrar a execução ao atingir o limite.

Números que caem

Item	Valor
Histórico de execuções na UI de Lakeflow Jobs	**60 dias**
Histórico em system.lakeflow	**365 dias**
Erros mais frequentes no gráfico da aba Runs	**5**
Chaves de liquid clustering	**até quatro**
Colunas com estatísticas coletadas por padrão no Delta	**as 32 primeiras**
Tamanho mínimo sugerido de arquivo	**8 MB**
Retenção do VACUUM (delta.deletedFileRetentionDuration)	**7 dias por padrão, mínimo de 7 dias**
Predictive optimization habilitada por padrão	**contas criadas a partir de 11/11/2024**
REPLACE PARTITIONED BY WITH CLUSTER BY	**Databricks Runtime 18.1 ou superior**
Peso do domínio na prova	**10% (4 ou 5 questões por prova)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Estágio com uma tarefa e núcleos ociosos é skew; aplicar salting.
Como não cair: Skew são poucas tarefas lentas entre muitas; uma tarefa só vem de janela sem PARTITION BY, gzip ou coalesce(1).

⚠ **Alerta de prova:** Adicionar workers acelera a tarefa lenta.
Como não cair: Mais workers não dividem uma tarefa sobrecarregada nem um estágio de tarefa única.

⚠ **Alerta de prova:** Reduzir spark.sql.shuffle.partitions corrige o skew.
Como não cair: Menos partições concentram mais dados; o caminho é AQE skew join e salting.

⚠ **Alerta de prova:** Cache do DataFrame ou shuffle partitions = 1 evitam spill.
Como não cair: Cache consome mais memória e uma partição agrava o spill; use auto ou mais memória.

⚠ **Alerta de prova:** Dobrar workers do mesmo tipo testa a hipótese de OOM.
Como não cair: Mantém a mesma memória por núcleo; o teste é dobrar a memória por núcleo.

⚠ **Alerta de prova:** Executores removidos em vermelho já provam falta de memória.
Como não cair: Primeiro o event log: autoscaling ou perda de spot também removem executores.

⚠ **Alerta de prova:** Cluster não sobe: olhar a Spark UI ou a matrix view.
Como não cair: Sem cluster não há aplicação Spark; o motivo está no event log pelo código de término.

⚠ **Alerta de prova:** Instalar bibliotecas no escopo da computação para todos usarem as mesmas versões.
Como não cair: Isso cria divergência entre jobs; isole com %pip ou environment e evite init scripts.

⚠ **Alerta de prova:** Succeeded with failures significa que a folha falhou ou que houve retries.
Como não cair: Significa falhas no meio com todas as folhas bem-sucedidas; folha com falha dá Failed e retry bem-sucedido dá Succeeded.

⚠ **Alerta de prova:** VACUUM com retenção zero resolve arquivos pequenos.
Como não cair: VACUUM só remove arquivos não referenciados e respeita no mínimo 7 dias; compactar é OPTIMIZE.

⚠ **Alerta de prova:** Particionar também por hora reduz arquivos por partição.
Como não cair: Mais colunas de partição pioram arquivos pequenos; use liquid clustering.

⚠ **Alerta de prova:** ALTER TABLE ... CLUSTER BY em tabela já particionada.
Como não cair: Clustering não é compatível com partição; use REPLACE PARTITIONED BY WITH CLUSTER BY.

⚠ **Alerta de prova:** Usar seis chaves de liquid clustering ou OPTIMIZE ZORDER BY AUTO.
Como não cair: O limite é quatro chaves, e ZORDER não tem modo automático nem é compatível com clustering; o automático é CLUSTER BY AUTO.

⚠ **Alerta de prova:** Predictive optimization roda sem custo, também no hive_metastore, com ZORDER.
Como não cair: Só tabelas gerenciadas do Unity Catalog, OPTIMIZE, VACUUM e ANALYZE, cobrada como serverless jobs.

Como cai na prova

📝 **Como cai:** Métricas da Spark UI descritas em números e quatro diagnósticos com correções.
Como resolver: Classifique o sintoma: Max x p75 (skew), estágio de uma tarefa (janela, gzip, coalesce), estatística de spill (memória), lacuna sem jobs (driver). Depois escolha a correção que ataca essa causa; mais workers raramente é a resposta.

📝 **Como cai:** Falha de job ou cluster pedindo onde investigar e o que fazer.
Como resolver: Partida ou executores removidos: event log primeiro. Falha no meio de job multitarefa: Repair run. Conflito de pacote: %pip ou environment. OOM: dobrar memória por núcleo.

📝 **Como cai:** Monitoramento: onde ver uma informação de execuções de jobs.
Como resolver: Comparar execuções do mesmo job = matrix view; dentro de uma execução = Timeline; visão do workspace e erros frequentes = aba Runs; mais de 60 dias = system.lakeflow.

📝 **Como cai:** Layout de tabela e manutenção automática.
Como resolver: Partição ruim ou filtros que mudam = liquid clustering; tabela particionada = REPLACE PARTITIONED BY; dados antigos = OPTIMIZE FULL; deixar escolher = CLUSTER BY AUTO; manutenção automática = predictive optimization em gerenciadas.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Max de 14 minutos e percentil 75 de 25 segundos no estágio	**Skew: AQE skew join e salting**
Estágio mais longo com uma única tarefa e row_number() OVER (ORDER BY ...)	**Janela sem PARTITION BY**
Estatísticas de spill no shuffle de uma agregação	**shuffle partitions auto ou mais memória**
Escrita com arquivos grandes demais e pequenos demais	**Hint REBALANCE**
Minutos sem jobs Spark, driver ocupado, workers ociosos	**Código não Spark no driver**
ExecutorLostFailure sem spot nem autoscaling no event log	**Testar o dobro de memória por núcleo**
Cluster de job falha ao iniciar	**Event log da computação e código de término**
INSUFFICIENT_INSTANCE_CAPACITY com spot	**Outra zona ou auto AZ, outro tipo de instância, flexible node types**
Tarefa 4 falhou, 1 a 3 deram certo, 5 e 6 dependem da 4	**Repair run**
Comparar duração com a de oito meses atrás	**system.lakeflow (365 dias)**
Falhas das últimas 48 horas e erros mais frequentes do workspace	**Aba Runs de Jobs & Pipelines**
Avisar após 1 hora sem parar e encerrar em 3 horas	**Warning + Duration warning e Timeout**
Clustering habilitado e consultas lentas nos dados antigos	**OPTIMIZE ... FULL**
Databricks deve escolher as chaves de clustering sozinho	**CLUSTER BY AUTO com predictive optimization**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- Skew = Max muito acima do p75; uma tarefa só = janela sem PARTITION BY, gzip ou coalesce(1).
- Event log antes de concluir OOM; OOM se testa com o dobro de memória por núcleo.
- Repair run refaz só falhas e dependentes; tarefas devem ser idempotentes.
- Liquid clustering: até 4 chaves, OPTIMIZE FULL para dados antigos, CLUSTER BY AUTO com predictive optimization.
- Predictive optimization = OPTIMIZE, VACUUM e ANALYZE em gerenciadas do Unity Catalog.

Essenciais para a prova (checklist)

- ☐ Sei identificar skew, spill e shuffle pelas métricas de estágio da Spark UI.
- ☐ Sei as causas de um estágio com uma única tarefa e de lacunas na timeline.

- ☐ Sei diagnosticar falha de partida de cluster pelo event log e códigos de término.
- ☐ Sei resolver conflito de bibliotecas e testar suspeita de OOM.
- ☐ Sei usar Repair run e interpretar os status de execução.
- ☐ Sei onde ficam aba Runs, matrix view, Timeline e system.lakeflow e o que cada um mostra.
- ☐ Sei configurar Warning e Timeout de duração com notificação.
- ☐ Sei habilitar, converter e reclusterizar com liquid clustering.
- ☐ Sei o que a predictive optimization faz, onde vale e como ligá-la por catálogo.
- ☐ Sei tratar arquivos pequenos com OPTIMIZE e layout adequado.

Questões de revisão do capítulo

R5.1 · Solução de problemas, monitoramento e otimização

Um job da Rede de Varejo Moda Já grava uma tabela a partir de uma consulta, e a Spark UI mostra que o estágio de escrita tem algumas tarefas muito maiores que as outras, gerando arquivos grandes demais e pequenos demais. O AQE está ligado. Qual hint a documentação sugere para a escrita?

A. MERGE
B. BROADCAST
C. REBALANCE
D. COALESCE(1)

R5.2 · Solução de problemas, monitoramento e otimização

Em um cluster clássico compartilhado da Fábrica de Software Pixel, um init script instala versões de pacotes Python que entram em conflito com as de um notebook novo, quebrando outros jobs. Qual prática a Databricks recomenda para evitar esse tipo de conflito?

A. Instalar as bibliotecas no escopo da computação para que os notebooks compartilhem as mesmas versões
B. Evitar init scripts e bibliotecas no escopo da computação; instalar dependências com %pip no notebook ou em environment
C. Trocar para o Databricks Runtime ML, que já traz versões compatíveis com os pacotes mais usados
D. Copiar os pacotes para o DBFS e instalar a partir dele em cada inicialização do cluster

R5.3 · Solução de problemas, monitoramento e otimização

A gerente de dados da Seguradora Vida Plena quer ver, em um só lugar, quantas execuções de jobs e pipelines do workspace falharam nas últimas 48 horas e quais foram os tipos de erro mais frequentes. Onde ela encontra isso?

A. Na aba Tasks de cada job, que mostra o total de falhas do workspace nas últimas 48 horas
B. Na aba Runs de Jobs & Pipelines, com o gráfico de execuções concluídas e os 5 erros mais frequentes
C. No query profile do SQL warehouse, que lista os erros de todos os jobs e pipelines do workspace
D. Na Spark UI de cada computação, filtrando a aba Jobs pelos status com falha das últimas 48 horas

R5.4 · Solução de problemas, monitoramento e otimização

A diretoria da Rede de Hotéis Estrela quer comparar a duração das execuções de um job de hoje com as de oito meses atrás, mas a UI de Lakeflow Jobs não mostra execuções tão antigas. Onde buscar esse histórico?

A. Nas tabelas de sistema system.lakeflow, como job_run_timeline, que guardam 365 dias
B. Na matrix view do job, ativando a opção de histórico estendido de execuções
C. No event log da computação, que guarda a duração de cada execução de job
D. Na Spark UI do cluster, que mantém o histórico de jobs desde a criação

Respostas comentadas

R5.1. C) REBALANCE

Por quê: Para skew na fonte, há outras opções: tratamento de skew join pelo AQE, salting ou pré-agregação e liquid clustering na tabela de origem.

✗ A) Incorreta. MERGE é hint de join (sort merge join).
✗ B) Incorreta. BROADCAST é hint de join, não de partições de saída.
✗ D) Incorreta. COALESCE(1) concentra tudo em uma partição e um arquivo.

> **Dica de prova:** Partições de saída desequilibradas na escrita = hint REBALANCE.

R5.2. B) Evitar init scripts e bibliotecas no escopo da computação; instalar dependências com %pip no notebook ou em environment

Por quê: Bibliotecas com escopo de notebook ficam isoladas na sessão do notebook e não afetam os demais. Depois de instalar ou atualizar um pacote com %pip, pode ser necessário dbutils.library.restartPython().

✗ A) Incorreta. Bibliotecas no escopo da computação criam divergência de ambiente entre jobs, o que as boas práticas pedem para evitar.
✗ C) Incorreta. O Runtime ML instala muitas bibliotecas que podem conflitar com as dependências do projeto; só deve ser usado quando necessário.
✗ D) Incorreta. Instalar bibliotecas do DBFS é legado e mantém o mesmo problema de instalação compartilhada.

> **Dica de prova:** Conflito de bibliotecas = isolar com %pip ou environment; evitar init scripts.

R5.3. B) Na aba Runs de Jobs & Pipelines, com o gráfico de execuções concluídas e os 5 erros mais frequentes

Por quê: A lista de execuções guarda os últimos 60 dias e aceita filtros por nome, tipo, run as, período, status e código de erro. O gráfico aparece quando o filtro está em Jobs ou Pipelines.

✗ A) Incorreta. A aba Tasks mostra a configuração das tarefas de um job.
✗ C) Incorreta. O query profile analisa uma consulta específica.
✗ D) Incorreta. A Spark UI mostra jobs Spark de uma computação, não execuções de Lakeflow Jobs do workspace.

> **Dica de prova:** Visão geral de falhas e erros mais comuns = aba Runs de Jobs & Pipelines.

R5.4. A) Nas tabelas de sistema system.lakeflow, como job_run_timeline, que guardam 365 dias

Por quê: A Databricks mantém o histórico de execuções na UI por até 60 dias; para preservar resultados, exporte-os. As tabelas de sistema de jobs podem ser cruzadas com as de cobrança para monitorar custo e desempenho.

✗ B) Incorreta. A UI guarda execuções por 60 dias, e não existe histórico estendido na matrix view.
✗ C) Incorreta. O event log registra eventos da computação, não o histórico de execuções de jobs.
✗ D) Incorreta. A Spark UI mostra a aplicação Spark de uma computação, não o histórico de execuções do job.

> **Dica de prova:** Histórico de jobs além de 60 dias = system.lakeflow (365 dias).

06

CAPÍTULO 6 · 16% DA PROVA

Trabalho com Lakeflow Jobs

Este domínio vale 16% (cerca de 7 questões por prova) e é muito de configuração: DAG de tarefas, Run if, If/else, For each, task values, parâmetros, tipos de tarefa e gatilhos. A prova descreve um fluxo (tarefa de limpeza que sempre roda, alerta quando alguma extração falha, lista dinâmica de lojas, arquivos que chegam em rajadas) e pede a configuração exata. Eu preciso saber de cor as seis condições de Run if, as referências dinâmicas mais usadas, a diferença entre gatilho por tempo e por dados e os limites (For each sem aninhar, pipeline uma atualização por vez, Run Job com até três níveis). Onde mais erro é usar Run if para avaliar valor (isso é If/else) e confundir Minimum time between triggers com Wait after last change.

Neste capítulo
17 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

DAG e Depends on

As dependências formam um grafo acíclico dirigido: tarefas dependentes esperam as anteriores e as independentes rodam em paralelo, seguindo as dependências e não a ordem de criação. Depends on e Run if definem a ordem e a regra de execução de cada tarefa.

Condições Run if

All succeeded (padrão), At least one succeeded, None failed, All done, At least one failed e All failed. Avaliam o resultado (sucesso ou falha) das dependências, nunca o valor de parâmetros. Quando a condição não é atendida, a tarefa fica Excluded.

Tarefas desabilitadas

Desabilitar uma tarefa a faz ser pulada sem perder configuração nem histórico. Ela termina como Disabled e não conta como sucesso para All succeeded nem como falha para At least one failed ou All failed; para a dependente rodar, use None failed.

If/else condition

Cria ramos true e false no DAG comparando job parameters, task parameters ou task values. Operandos devem ser número, string ou booleano. == e != comparam como texto (12.0 é diferente de 12); operadores de ordem comparam números.

For each

Roda a tarefa aninhada para cada elemento de um array JSON digitado, de um task value ou de um job parameter; o valor da iteração é {{input}} (ou {{input.campo}}) e Concurrency define iterações simultâneas. A tarefa aninhada não pode ser outro For each; uma tarefa Pipeline aninhada roda uma iteração por vez.

Task values

dbutils.jobs.taskValues.set(key, value) em notebook Python publica valores representáveis em JSON de até 48 KiB. A recomendação é referenciá-los nas tarefas seguintes por {{tasks.NOME.values.CHAVE}}, em vez de taskValues.get. Variáveis Python e widgets não passam valores entre tarefas.

Job parameters e parâmetros de tarefa

Job parameters são pares chave-valor repassados às tarefas; o valor padrão pode ser sobreposto em Run now with different parameters ou pela API (não é controle de segurança). Notebook lê parâmetro com dbutils.widgets.get('nome'); script Python recebe como argumento de linha de comando; tarefa SQL como parâmetro da consulta.

Dynamic value references

Usam chaves duplas e são resolvidas na execução: {{job.start_time.iso_date}} (data de início), {{job.run_id}}, {{job.trigger.type}}, {{job.parameters.nome}}, {{tasks.X.values.Y}}, {{tasks.X.execution_count}}, {{job.trigger.file_arrival.location}} e {{job.trigger.table_update.updated_tables}}. Os valores resolvidos aparecem em Parameters na página da execução.

Gatilho Scheduled

Simple define intervalo e unidade a partir da configuração, sem escolher horário. Advanced define período, horário e fuso, com sintaxe Quartz cron opcional; para horas absolutas sem horário de verão, UTC. Use para entregas em horário fixo.

Gatilho File arrival

Orientado a dados: monitora volume ou external location do Unity Catalog, em regra a cada minuto, inclusive subpastas. Exige leitura no local e CAN MANAGE no job; file events na external location melhoram o desempenho. Minimum time between triggers limita a frequência; Wait after last change espera a rajada terminar.

Gatilho Table update

Monitora tabelas Delta e Iceberg gerenciadas, externas Delta, materialized views, streaming tables e views, com Any table ou All tables. A identidade do Run as precisa de SELECT na tabela, USE CATALOG e USE SCHEMA; change data feed não é exigido.

Gatilho Continuous

Mantém uma execução ativa por vez, reiniciando com backoff exponencial em falhas; não aceita dependências de tarefas nem retries (opcional Task retry mode On failure). Indicado para streaming sempre ativo. Pipeline contínuo recomendado = tarefa Pipeline em job contínuo, com Pipeline mode triggered.

Pause, Retries, Timeout e fila

Pause no gatilho impede novas execuções pelo gatilho, mantém as ativas e não bloqueia Run now. Retries repetem tarefas que falharam e o Timeout vale para cada tentativa. Maximum concurrent runs = 1 com Queue enfileira a próxima execução em vez de pulá-la.

Notificações

Configuráveis no job ou na tarefa para início, sucesso, falha, duração excedida e backlog de streaming, com até três destinos de sistema por evento. Notificações do job não são enviadas a cada retry; para aviso a cada tentativa, configure Failure na tarefa (com opção Mute notifications until the last retry).

Tipos de tarefa

Notebook (serverless é o padrão quando habilitado; em SQL warehouse só se for 100% SQL), SQL (Query, Alert ou File, em warehouse serverless ou pro), Dashboard (dashboard publicado, assinantes, PDF), Pipeline (uma atualização por execução agendada; opção full refresh), Run Job (reaproveita job inteiro; até três níveis, sem ciclos).

Computação e bibliotecas em jobs

Job cluster compartilhado entre tarefas da mesma execução reduz partidas (escopo de uma execução, não entre jobs). No serverless, bibliotecas vão no painel Environment do notebook ou %pip; init scripts não são suportados. Na clássica, use Dependent libraries.

Identidade e Git

Por padrão o criador é IS OWNER e Run as; para produção, troque o Run as por uma entidade de serviço (exige CAN MANAGE ou IS OWNER). Com Git provider, só um repositório remoto vale para todas as tarefas do job, com caminhos relativos.

Macetes para memorizar

Run if x If/else

Run if olha o Resultado; If/else olha o Valor.

Condições Run if

Limpeza = All done; alarme = At least one failed; consolidar parcial = At least one succeeded; desabilitada no meio = None failed.

Gatilho de arquivo

Minimum = Máximo de uma por intervalo; Wait = Espera a rajada acabar.

Referências dinâmicas

job.* fala do job e do gatilho; tasks.* fala das tarefas; input fala da iteração do For each.

Limites de aninhamento

For each não abraça For each; Run Job empilha no máximo três.

Notificação por retry

Quer saber de cada tropeço? Avise na tarefa, não no job.

Pipeline contínuo

Job contínuo carregando pipeline triggered.

Tabela de memorização: não confunda

Quando é...	...e quando é
Run if Decidir se a tarefa roda pelo resultado (sucesso ou falha) das dependências.	**If/else condition** Desviar o fluxo pelo valor de um parâmetro ou task value.
All done Limpeza que roda sempre ao final, com sucesso ou falha das dependências.	**At least one failed** Alerta ou compensação que só roda quando ao menos uma dependência falhou.
Minimum time between triggers Arquivos chegam quase sem parar e aceito no máximo uma execução a cada X segundos.	**Wait after last change** Arquivos chegam em rajadas e cada rajada deve gerar uma execução depois que termina de chegar.
Gatilho Scheduled Entrega em horário fixo com dados já prontos, com fuso definido.	**Gatilho File arrival ou Table update** Chegadas irregulares: rodar quando houver arquivo novo ou tabela atualizada.

Repair run Reexecutar só as tarefas que falharam e suas dependentes em uma execução existente.	**Run now with different parameters** Iniciar uma execução nova uma única vez com outro valor de parâmetro, sem mudar o job.
For each Mesma tarefa para cada item de uma lista, com Concurrency.	**Run Job** Reaproveitar um job inteiro como etapa de outro, passando job parameters.
Notificação no job Aviso do resultado final da execução do job.	**Notificação na tarefa** Aviso a cada falha de tentativa de uma tarefa com retries.

Números que caem

Item	Valor
Tamanho máximo de um task value	**48 KiB**
Níveis de Run Job aninhados	**no máximo três**
Destinos de sistema por evento de notificação	**até três**
Verificação do gatilho File arrival	**em regra, a cada minuto**
Fila habilitada por padrão	**jobs criados pela UI depois de 15/04/2024**
Condições de Run if	**seis (All succeeded é o padrão)**
Peso do domínio na prova	**16% (cerca de 7 das 45 questões)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Run if = At least one failed para rodar o reprocessamento quando há rejeições.
Como não cair: A tarefa que conta rejeições termina com sucesso; desvio por valor é If/else com task value.

⚠ **Alerta de prova:** Run if = All succeeded com o parâmetro ambiente = prod.
Como não cair: Run if não lê parâmetros; use If/else com {{job.parameters.ambiente}} == prod.

⚠ **Alerta de prova:** Repair run para reprocessar outra data.
Como não cair: Repair reexecuta tarefas que falharam; outra data uma vez é Run now with different parameters.

⚠ **Alerta de prova:** If/else com dicionário Python como operando.
Como não cair: Operando só pode ser número, string ou booleano; listas e dicionários servem para For each.

⚠ **Alerta de prova:** == no If/else compara números, então 12.0 == 12 é verdadeiro.
Como não cair: == e != comparam como texto; use operadores de ordem para números ou grave no formato esperado.

⚠ **Alerta de prova:** For each aninhado em For each para filial e produto.
Como não cair: Não é permitido; gere a lista de pares antes ou aninhe via Run Job.

⚠ **Alerta de prova:** Concurrency 10 em For each com tarefa Pipeline roda dez atualizações juntas.
Como não cair: O pipeline roda uma atualização por vez; a iteração fica limitada a uma.

⚠ **Alerta de prova:** Gatilho Continuous para esperar arquivos sem cluster ligado.
Como não cair: Continuous mantém execução sempre ativa; chegada irregular de arquivos é File arrival.

⚠ **Alerta de prova:** Wait after last change = 1800 para arquivos que chegam sem parar.
Como não cair: Cada arquivo novo reinicia a espera e o job quase nunca dispara; use Minimum time between triggers.

⚠ **Alerta de prova:** File arrival no caminho de armazenamento de tabelas Delta.
Como não cair: Para tabelas existe Table update; File arrival é para volumes e external locations.

⚠ **Alerta de prova:** Gatilho Table update que não dispara exige change data feed ou MODIFY para o dono.
Como não cair: Falta SELECT (com USE CATALOG e USE SCHEMA) para a identidade do Run as.

⚠ **Alerta de prova:** Notificação de Failure no job avisa a cada retry.
Como não cair: Notificações do job não são enviadas nos retries; configure-as na tarefa.

⚠ **Alerta de prova:** Mudar Pipeline mode para continuous para rodar sempre.
Como não cair: O modo contínuo nativo é desencorajado e não suporta modos de desempenho serverless; use job contínuo com pipeline triggered.

⚠ **Alerta de prova:** Tarefa SQL do tipo File em SQL warehouse classic.
Como não cair: A tarefa SQL exige warehouse serverless ou pro.

Como cai na prova

📝 **Como cai:** DAG descrito em texto com uma tarefa final e uma regra de quando ela deve rodar.
Como resolver: Traduza a regra para resultado das dependências: sempre (All done), só com falha (At least one failed ou All failed), se alguma deu certo (At least one succeeded), se nada falhou mesmo com puladas (None failed). Se a regra fala de valor, troque para If/else.

📝 **Como cai:** Passar valores entre tarefas ou para uma lista de execuções.
Como resolver: Valor calculado = task value com {{tasks.X.values.Y}}; lista = For each com Inputs e {{input}}; parâmetro de entrada em notebook = dbutils.widgets.get.

📝 **Como cai:** Escolher o gatilho a partir do padrão de chegada dos dados.
Como resolver: Horário fixo = Scheduled (Advanced para horário exato). Arquivos irregulares = File arrival (ajuste Minimum ou Wait). Tabelas = Table update (Any ou All). Sempre ativo = Continuous.

📝 **Como cai:** Problema operacional (execução pulada, retries sem aviso, timeout, job de outra pessoa).
Como resolver: Pulada = concurrent runs + Queue; aviso por tentativa = notificação na tarefa; timeout conta por tentativa; identidade = Run as com entidade de serviço.

📝 **Como cai:** Escolher o tipo de tarefa e a computação.
Como resolver: Arquivo .sql = tarefa SQL File em warehouse serverless ou pro; dashboard = tarefa Dashboard; pipeline = tarefa Pipeline (full refresh para reprocessar); job inteiro = Run Job; notebook Python = serverless ou job compute.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Limpeza que deve rodar sempre ao final	**Run if All done**
Abrir chamado se pelo menos uma extração falhar	**Run if At least one failed**
Consolidar se ao menos um provedor responder	**Run if At least one succeeded**
Dependência desabilitada e a tarefa seguinte parou de rodar	**Run if None failed**
Rodar ramo só quando um valor calculado for maior que zero	**If/else condition com {{tasks.X.values.Y}}**
Mesmo notebook para uma lista de tabelas com no máximo N simultâneas	**For each com {{input}} e Concurrency**
Valor calculado em uma tarefa usado pelas seguintes	**taskValues.set + {{tasks.tarefa.values.chave}}**
Data em que a execução começou como parâmetro	**{{job.start_time.iso_date}}**
Quais tabelas dispararam o job	**{{job.trigger.table_update.updated_tables}}**
Rodar só quando duas tabelas tiverem sido atualizadas	**Table update com All tables**
Execuções às 6h e às 18h ou dias úteis às 7h	**Scheduled Advanced (cron e fuso)**
Execução seguinte aparece como Skipped e não quero paralelismo	**Maximum concurrent runs 1 + Queue**
Job de produção roda como o criador que vai sair	**Run as com entidade de serviço**
Tarefas sequenciais com partidas lentas de job cluster	**Job cluster compartilhado na execução**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- Run if avalia sucesso ou falha; valor de parâmetro ou task value é If/else.
- Desabilitada não conta como sucesso: use None failed na dependente.
- For each não aninha For each, e tarefa Pipeline em For each roda uma por vez.
- Timeout vale por tentativa; notificação a cada retry só na tarefa.
- Gatilho Table update precisa de SELECT para o Run as.

Essenciais para a prova (checklist)

- ☐ Sei as seis condições de Run if e quando usar cada uma.
- ☐ Sei configurar If/else com job parameters e task values e o cuidado com ==.
- ☐ Sei configurar For each com Inputs, {{input}} e Concurrency e suas restrições.
- ☐ Sei passar valores entre tarefas com task values e dynamic value references.
- ☐ Sei ler parâmetros em notebook com dbutils.widgets.get e rodar com outros parâmetros.
- ☐ Sei escolher entre Scheduled, File arrival, Table update e Continuous.
- ☐ Sei ajustar Minimum time between triggers e Wait after last change.
- ☐ Sei configurar retries, timeout, notificações, fila e concorrência.
- ☐ Sei os requisitos das tarefas SQL, Dashboard, Pipeline, Run Job e notebook.
- ☐ Sei por que usar entidade de serviço no Run as e job cluster compartilhado.

Questões de revisão do capítulo

R6.1 · Trabalho com Lakeflow Jobs

O job de reprocessamento da Cooperativa Agrícola Vale do Sol tem o job parameter data_ref com valor padrão do dia atual. Hoje a equipe precisa rodá-lo uma vez para 2026-09-30 sem alterar a definição do job. Como fazer pela UI?

A. Usar Run now with different parameters e informar data_ref=2026-09-30
B. Usar Repair run na última execução e informar o novo data_ref no diálogo
C. Criar um job novo com data_ref fixo e excluí-lo depois da execução
D. Editar o valor padrão de data_ref, executar o job e restaurar o valor depois

R6.2 · Trabalho com Lakeflow Jobs

No job de faturamento da Elétrica Vale Norte, a tarefa validar_notas grava a quantidade de notas rejeitadas com dbutils.jobs.taskValues.set(key="rejeitadas", value=total). A tarefa de reprocessamento deve rodar apenas quando houver rejeições. Como montar essa lógica no job?

A. Criar uma tarefa For each sobre {{tasks.validar_notas.values.rejeitadas}} que rode o reprocessamento por nota
B. Configurar Run if = At least one failed no reprocessamento para que rode quando validar_notas tiver rejeições
C. Adicionar uma tarefa If/else condition com {{tasks.validar_notas.values.rejeitadas}} > 0 e ligar o reprocessamento ao ramo true
D. Definir um gatilho de atualização de tabela na tabela de notas para disparar o reprocessamento ao surgir uma rejeição

R6.3 · Trabalho com Lakeflow Jobs

No job da Rede de Supermercados Família Feliz, uma tarefa de notebook escrita em Python com algumas células SQL foi configurada para rodar em um SQL warehouse serverless e não pôde ser salva assim. Por quê?

A. SQL warehouses só rodam notebooks escritos inteiramente em SQL, com SQL como linguagem padrão
B. SQL warehouses só rodam notebooks Python quando o warehouse é do tipo pro
C. Notebooks com mais de uma linguagem exigem uma tarefa If/else antes de rodar
D. Notebooks em warehouses precisam estar em um Git folder conectado ao job

R6.4 · Trabalho com Lakeflow Jobs

Na Indústria Gráfica Papel Nobre, a tarefa de notebook de um job recebe o parâmetro data_ref, configurado como chave e valor na própria tarefa. Como o código Python do notebook lê esse valor?

A. dbutils.widgets.get("data_ref")
B. spark.conf.get("data_ref")
C. os.environ["data_ref"]
D. dbutils.jobs.taskValues.get("data_ref")

Respostas comentadas

R6.1. A) Usar Run now with different parameters e informar data_ref=2026-09-30
Por quê: Job parameters são pares chave-valor definidos no job e repassados às tarefas. O valor estático funciona como padrão, que pode ser sobreposto em Run now with different parameters ou pela API. Isso não é um controle de segurança: quem tem CAN MANAGE RUN pode alterá-lo.

✗ B) Incorreta. Repair reexecuta tarefas malsucedidas de uma execução anterior; não inicia uma nova execução completa para outra data.
✗ C) Incorreta. Duplicar o job é desnecessário e perde o histórico no job original.
✗ D) Incorreta. Alterar e restaurar a definição do job é arriscado e afeta outras execuções no intervalo.

> **Dica de prova:** Executar uma vez com outro valor = Run now with different parameters.

R6.2. C) Adicionar uma tarefa If/else condition com {{tasks.validar_notas.values.rejeitadas}} > 0 e ligar o reprocessamento ao ramo true

Por quê: Task values passam informações entre tarefas e podem ser referenciados por {{tasks.NOME_DA_TAREFA.values.CHAVE}}. Combinados com a tarefa If/else condition, criam ramificações no DAG do job.

✗ A) Incorreta. For each executa uma tarefa em laço sobre uma lista de entradas; aqui o valor é uma contagem e a necessidade é um desvio condicional.
✗ B) Incorreta. Run if avalia o resultado (sucesso ou falha) das tarefas anteriores; validar_notas termina com sucesso mesmo quando registra rejeições.
✗ D) Incorreta. Gatilhos iniciam o job inteiro, não controlam o fluxo entre tarefas dentro de uma execução.

> **Dica de prova:** Desvio por valor calculado em uma tarefa anterior = If/else condition + task value.

R6.3. A) SQL warehouses só rodam notebooks escritos inteiramente em SQL, com SQL como linguagem padrão

Por quê: Escolha a computação conforme o tipo de tarefa: notebooks e scripts Python em serverless para jobs (ou job compute clássico); tarefas SQL, dbt e dashboards em SQL warehouse serverless ou pro.

✗ B) Incorreta. Nenhum tipo de SQL warehouse executa Python.
✗ C) Incorreta. If/else controla fluxo entre tarefas; não muda a computação.
✗ D) Incorreta. A origem do notebook não é o motivo da restrição.

> **Dica de prova:** Notebook em SQL warehouse = só se for 100% SQL.

R6.4. A) dbutils.widgets.get("data_ref")

Por quê: Cada tipo de tarefa recebe parâmetros de um jeito: notebooks por dbutils.widgets, scripts Python como argumentos de linha de comando e tarefas SQL como parâmetros da consulta.

✗ B) Incorreta. spark.conf lê configurações do Spark, não parâmetros da tarefa.
✗ C) Incorreta. Parâmetros de tarefa de notebook não chegam como variáveis de ambiente.
✗ D) Incorreta. taskValues lê valores gravados por outra tarefa e exige o nome da tarefa de origem.

> **Dica de prova:** Parâmetro de tarefa de notebook = dbutils.widgets.get("nome").

07

CAPÍTULO 7 · 22% DA PROVA

Transformação e modelagem de dados

É o maior domínio (22%, cerca de 10 questões por prova) e mistura PySpark, SQL e Lakeflow pipelines. Preciso limpar a bronze para a silver (nulos, tipos, datas, deduplicação), combinar DataFrames (inner, left, broadcast, várias chaves, cross, union e union all), manipular colunas e linhas, agregar, escolher o objeto gold certo (tabela, view, materialized view, streaming table) e aplicar qualidade com expectations e constraints. A prova costuma mostrar quatro trechos de código quase iguais e só um faz exatamente o pedido; o detalhe está em 'any' x 'all', & x and, explode x explode_outer, UNION x UNION ALL. Onde mais erro é a semântica sutil de funções parecidas e a diferença entre materialized view e streaming table.

Neste capítulo
19 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Tratamento de nulos no DataFrame

na.drop() remove linhas com qualquer nulo ('any' é o padrão); na.drop('all') só quando todas as colunas avaliadas são nulas; subset limita as colunas. na.fill(valor, subset=[...]) preenche nulos. na.replace troca valores existentes, como string vazia, que não é nulo.

coalesce, try_cast e ANSI

Aritmética com NULL resulta em NULL, então use valor - coalesce(desconto, 0). Com ANSI ativado (padrão no serverless), cast inválido gera erro; try_cast devolve NULL para valores malformados. O operador :: é cast comum e falha com ANSI.

Conversão de tipos e datas

withColumn('col', col('col').cast('double')) converte mantendo o nome. to_date(coluna, 'dd/MM/yyyy') converte texto fora do padrão ISO; sem formato, to_date só aceita yyyy-MM-dd ou yyyy-MM-dd HH:mm:ss. date_format faz o caminho inverso (data para STRING).

Operações de coluna

withColumnRenamed(antigo, novo) renomeia mantendo o resto; drop('c1', 'c2') remove colunas; select mantém só as listadas; alias nomeia expressões; selectExpr aceita expressões SQL como 'cod AS codigo_obra'. split(coluna, '-') transforma texto delimitado em array.

Filtros com várias condições

Em PySpark, & é AND e | é OR, cada comparação entre parênteses por causa da precedência do Python. and e or do Python não funcionam com colunas Spark.

Tipos de join

O padrão do PySpark é inner, que descarta linhas sem correspondência. left mantém tudo da esquerda com nulos do outro lado; left_anti devolve só as linhas sem correspondência; crossJoin ou CROSS JOIN gera o produto cartesiano. Várias chaves: on=['k1', 'k2'] ou condições combinadas com &.

Broadcast join e autoBroadcastJoinThreshold

No broadcast join a tabela pequena é copiada para cada executor e a grande não sofre shuffle. spark.sql.autoBroadcastJoinThreshold é o limite do broadcast automático (-1 desliga); o hint BROADCAST força a transmissão mesmo acima do limite. Hints de join: BROADCAST, MERGE, SHUFFLE_HASH e SHUFFLE_REPLICATE_NL.

Adaptive query execution (AQE)

Ligado por padrão, reotimiza o plano em execução: troca sort merge por broadcast hash join quando a tabela fica pequena, junta partições pequenas após o shuffle, trata skew em joins e propaga relações vazias. DataFrame.explain() compara plano inicial e atual.

Operadores de conjunto

UNION, INTERSECT e EXCEPT usam DISTINCT por padrão; UNION ALL mantém duplicatas legítimas. As colunas são casadas por posição: mesmo número de colunas e tipos compatíveis, nomes não importam.

explode, explode_outer e PIVOT

explode gera uma linha por elemento de ARRAY ou par de MAP e descarta arrays nulos ou vazios; explode_outer mantém a linha com NULL. O operador .* expande STRUCT, não ARRAY. PIVOT gira linhas em colunas; UNPIVOT faz o inverso.

Deduplicação

distinct() remove linhas inteiramente iguais. Para o registro mais recente por chave, use row_number() OVER (PARTITION BY chave ORDER BY data DESC) com QUALIFY = 1; funções de janela não podem ir no WHERE.

Agregações

groupBy(...).agg(func(...).alias('nome')) equivale ao GROUP BY. count conta não nulos, inclusive repetidos; count_if(condição) conta verdadeiros; approx_count_distinct(expr, relativeSD) estima distintos com HyperLogLog++, e relativeSD menor dá mais precisão e mais custo. dbutils.data.summarize(df) mostra estatísticas resumidas no notebook.

Escrita: INSERT, MERGE e CTAS

INSERT INTO acrescenta; INSERT OVERWRITE substitui o conteúdo em uma transação, mantendo time travel. MERGE INTO com WHEN MATCHED THEN UPDATE SET * e WHEN NOT MATCHED THEN INSERT * faz upsert atômico. CREATE TABLE ... AS SELECT tira uma foto fixa do resultado.

Objetos gold

View recalcula a cada consulta e não guarda dados (uso esporádico). Materialized view guarda o resultado, reflete inserções, atualizações e exclusões da origem na próxima atualização, recalculando o que for preciso (muitos leitores, joins e agregados). Streaming table processa cada linha uma vez, para fontes só de inserção. Temporary view vale só no notebook ou consulta que a criou.

Materialized views e streaming tables standalone

Podem ser criadas em SQL warehouse ou notebook serverless. CREATE MATERIALIZED VIEW ... SCHEDULE EVERY 1 HOUR agenda a atualização; TRIGGER ON UPDATE atualiza quando a origem muda. Em Python, @dp.table declara streaming table e @dp.materialized_view, materialized view, com from pyspark import pipelines as dp.

Expectations

Warn (padrão: grava o registro inválido e só registra métricas; @dp.expect), drop (ON VIOLATION DROP ROW ou @dp.expect_or_drop) e fail (ON VIOLATION FAIL UPDATE ou @dp.expect_or_fail, derruba a atualização). expect_all, expect_all_or_drop e expect_all_or_fail aplicam um dicionário de regras. Métricas de warn e drop na aba Data quality e no event log.

Constraints e imposição de esquema

NOT NULL e CHECK são impostas em tabelas Delta e valem para qualquer escrita; PRIMARY KEY, FOREIGN KEY e UNIQUE são informativas. Schema enforcement rejeita INSERT com coluna inexistente; para adicionar colunas use ALTER TABLE ADD COLUMN ou mergeSchema.

AUTO CDC

Aplica feed de CDC com inserções, atualizações, exclusões e eventos fora de ordem em uma streaming table, com chaves, coluna de sequência e SCD tipo 1 ou 2. Em SQL, AUTO CDC ... INTO; em Python, create_auto_cdc_flow; AUTO CDC FROM SNAPSHOT quando só há snapshots. Substituiu APPLY CHANGES.

Parâmetros básicos de ajuste

spark.sql.shuffle.partitions (padrão 200) define as partições de shuffle de joins e agregações; auto deixa o Spark calcular pelo volume. Não fixe spark.executor.memory nem dynamicAllocation (não suportado; o equivalente é autoscaling); dimensione pela instância. Driver sobrecarregado com workers ociosos pede driver maior e menos concorrência. Depois de cada ajuste, meça de novo na Spark UI.

Macetes para memorizar

na.drop

Any é o padrão e Arranca qualquer nulo; All só quando Ausente em Tudo.

Expectations

Warn só avisa (grava e mede), Drop Descarta a linha, Fail Falha a atualização inteira.

Objeto gold

Raro = view; muito lido = materialized view; só cresce = streaming table; foto fixa = CTAS.

Operadores PySpark

No Spark é & e |, e cada condição ganha seu parêntese.

Decoradores de pipeline

@dp.table é Tabela de streaming; @dp.materialized_view é a MV.

Constraints

Só NOT NULL e CHECK mordem; PK e FK só latem.

Medalhão e semântica

Bronze em streaming, silver limpa e deduplica, gold em lote com materialized view incremental.

Tabela de memorização: não confunda

Quando é...	...e quando é
na.drop('any') Remover a linha se qualquer coluna avaliada for nula (padrão).	**na.drop('all')** Remover só quando todas as colunas avaliadas forem nulas.
na.fill Preencher valores nulos.	**na.replace** Trocar valores existentes, como string vazia ou código inválido.
UNION Empilhar removendo duplicatas (equivale a UNION DISTINCT).	**UNION ALL** Empilhar mantendo todas as linhas, inclusive duplicatas legítimas.
explode Uma linha por elemento, aceitando perder linhas com array nulo ou vazio.	**explode_outer** Uma linha por elemento mantendo as linhas sem array, com NULL.
Materialized view Agregados e joins lidos por muitos consumidores, com origem que recebe updates e deletes; o resultado precisa ser correto.	**Streaming table** Fonte só de inserções, cada linha processada uma vez com baixa latência; joins não são recalculados se a dimensão mudar.
View Uso raro ou etapa leve; recalcula a cada acesso sem gastar armazenamento.	**Materialized view** Muitas leituras do mesmo resultado caro; guarda o resultado e exige refresh.
Expectation (pipeline) Regra de qualidade dentro de um pipeline, com warn, drop ou fail e métricas.	**CHECK constraint (tabela Delta)** Regra que vale para qualquer processo que escreva na tabela; a transação que viola falha.

Números que caem

Item	Valor
Padrão de spark.sql.shuffle.partitions	**200**
spark.sql.autoBroadcastJoinThreshold que desliga o broadcast automático	**-1**
Parâmetro precise de dbutils.data.summarize	**Databricks Runtime 10.4 LTS ou superior**
Peso do domínio na prova	**22% (cerca de 10 das 45 questões)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** df.na.drop(subset=[...]) remove só quando as duas colunas são nulas.
Como não cair: Sem o primeiro argumento vale 'any'; para exigir todas nulas, use na.drop('all', subset=[...]).

⚠ **Alerta de prova:** na.fill troca string vazia por um valor padrão.
Como não cair: String vazia não é nula; para trocá-la use na.replace([''], ['NAO_INFORMADO'], subset=[...]).

⚠ **Alerta de prova:** coalesce(valor - desconto) trata o desconto nulo.
Como não cair: A subtração já virou nulo antes; aplique coalesce no operando: valor - coalesce(desconto, 0).

⚠ **Alerta de prova:** peso_txt::DOUBLE tolera valores inválidos.
Como não cair: :: é cast comum e falha com ANSI; para receber NULL use try_cast.

⚠ **Alerta de prova:** Combinar condições com and e or do Python.
Como não cair: Colunas Spark exigem & e |, com parênteses em cada comparação.

⚠ **Alerta de prova:** Join sem how mantém todas as linhas da esquerda.
Como não cair: O padrão é inner; para manter todas as linhas use how='left' com o lado certo à esquerda.

⚠ **Alerta de prova:** Definir autoBroadcastJoinThreshold como -1 para forçar broadcast.
Como não cair: -1 desliga o broadcast automático; para forçar em uma consulta use o hint BROADCAST.

⚠ **Alerta de prova:** SHUFFLE_HASH ou MERGE transmitem a tabela pequena.
Como não cair: SHUFFLE_HASH embaralha os dois lados e MERGE é sort merge join; quem transmite é BROADCAST.

⚠ **Alerta de prova:** UNION perdeu linhas por causa de nulos ou fez interseção.
Como não cair: UNION sem qualificador remove duplicatas; interseção é INTERSECT; UNION ALL mantém tudo.

⚠ **Alerta de prova:** UNION ALL falhou porque os nomes das colunas diferem.
Como não cair: Nomes não importam; o erro vem de número de colunas diferente ou tipos incompatíveis por posição.

⚠ **Alerta de prova:** Streaming table para agregado gold com origem que sofre updates e deletes.
Como não cair: Streaming table processa cada linha uma vez e não reflete mudanças; use materialized view.

⚠ **Alerta de prova:** Expectation sem ON VIOLATION descarta o registro inválido.
Como não cair: Sem ação, a expectation só avisa (warn) e grava o registro; descartar é DROP ROW, derrubar é FAIL UPDATE.

⚠ **Alerta de prova:** PRIMARY KEY impede nulos ou valores inválidos.
Como não cair: Chaves são informativas; impostas são só NOT NULL e CHECK.

⚠ **Alerta de prova:** Usar spark.dynamicAllocation.enabled = true para o Databricks escalar.
Como não cair: Essas propriedades não são suportadas; o equivalente é o autoscaling, e configurações fixas de memória devem ser removidas.

Como cai na prova

📝 **Como cai:** Quatro trechos de PySpark quase iguais para uma regra de limpeza.
Como resolver: Leia a regra palavra por palavra (todas x qualquer, nulo x vazio, preencher x remover) e confira o argumento exato de cada método. Elimine métodos que operam em linhas quando o pedido é sobre colunas, e vice-versa.

📝 **Como cai:** Join ou filtro com várias condições em código.
Como resolver: Cheque o tipo de join (padrão inner), qual DataFrame está à esquerda, se as condições usam & ou |, e se cada comparação está entre parênteses.

📝 **Como cai:** Escolher o objeto gold ou o tipo de dataset do pipeline.
Como resolver: Pergunte: a origem só cresce? (streaming table). Há updates, deletes, joins e muitos leitores? (materialized view). Uso raro? (view). Resultado não pode mudar? (CTAS).

📝 **Como cai:** Regra de qualidade com consequência descrita (descartar, falhar, só medir, valer para qualquer escrita).
Como resolver: Mapeie a consequência: medir = expect; descartar = drop; derrubar = fail; qualquer escrita na tabela = CHECK ou NOT NULL.

📝 **Como cai:** Plano de join ou parâmetro de ajuste (shuffle, broadcast, memória).
Como resolver: Broadcast para dimensão pequena (hint se acima do limite), shuffle partitions auto para volume variável, não fixar memória de executor, e sempre medir de novo depois do ajuste.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Remover só quando todas as colunas indicadas forem nulas	**na.drop('all', subset=[...])**
Valores inválidos devem virar NULL sem falhar com ANSI	**try_cast**
Data em texto dd/MM/yyyy	**to_date(coluna, 'dd/MM/yyyy')**
Dimensão pequena com fato de bilhões de linhas e shuffle pesado	**Hint BROADCAST na tabela pequena**
Plano virou broadcast sozinho durante a execução	**AQE**
Empilhar tabelas com linhas idênticas legítimas	**UNION ALL**
Uma linha por item do array, mantendo pedidos sem array	**explode_outer**
Uma coluna por trimestre com a soma	**PIVOT**
Último evento de cada chave	**row_number() OVER (PARTITION BY ... ORDER BY ... DESC) + QUALIFY = 1**
Contar linhas que atendem a uma condição	**count_if(condição)**
Clientes distintos sobre bilhões de eventos, erro aceitável	**approx_count_distinct (relativeSD menor para mais precisão)**
Atualizar existentes e inserir novos em uma operação atômica	**MERGE INTO**
Feed de CDC com histórico em pipeline	**AUTO CDC com SCD tipo 2 em streaming table**
Lote inválido deve derrubar a atualização inteira	**expect_or_fail / ON VIOLATION FAIL UPDATE**

🎯 Se você lembrar de só uma coisa deste capítulo...

- na.drop usa 'any' por padrão; string vazia não é nulo.
- Join padrão é inner; UNION padrão é DISTINCT.
- Materialized view reflete mudanças da origem; streaming table processa cada linha uma vez.
- Expectation sem ação só avisa; NOT NULL e CHECK são as únicas constraints impostas.
- BROADCAST força broadcast; -1 no autoBroadcastJoinThreshold desliga.

Essenciais para a prova (checklist)

- ☐ Sei usar na.drop, na.fill e na.replace com subset e how corretos.
- ☐ Sei converter tipos com cast, try_cast e to_date com formato.
- ☐ Sei escrever joins inner, left, left_anti, cross e com várias chaves em PySpark.
- ☐ Sei quando aplicar o hint BROADCAST e o que o AQE muda em execução.
- ☐ Sei a diferença entre UNION, UNION ALL, INTERSECT e EXCEPT e a regra de posição das colunas.
- ☐ Sei usar explode, explode_outer, split, PIVOT e selectExpr.
- ☐ Sei deduplicar com distinct e com row_number + QUALIFY.
- ☐ Sei usar groupBy com agg, count_if e approx_count_distinct com relativeSD.
- ☐ Sei escolher entre tabela, view, materialized view, streaming table e temporary view.
- ☐ Sei aplicar expectations (warn, drop, fail), CHECK, NOT NULL e AUTO CDC.

Questões de revisão do capítulo

R7.1 · Transformação e modelagem de dados

Em um Lakeflow pipeline da Distribuidora Água Viva, a streaming table silver_entregas não deve receber registros com cep nulo, mas o pipeline precisa continuar rodando, e a equipe quer acompanhar quantos registros foram descartados. Qual definição atende?

A. CONSTRAINT cep_valido EXPECT (cep IS NOT NULL) ON VIOLATION FAIL UPDATE
B. CONSTRAINT cep_valido EXPECT (cep IS NOT NULL) ON VIOLATION DROP ROW
C. CONSTRAINT cep_valido EXPECT (cep IS NOT NULL)
D. CONSTRAINT cep_valido CHECK (cep IS NOT NULL)

R7.2 · Transformação e modelagem de dados

Na Rede Escolar Saber Mais, df_notas tem as colunas escola, disciplina e nota. A coordenação quer a média de nota por escola, com a coluna de resultado chamada media_nota. Considerando from pyspark.sql.functions import avg, qual código atende?

A. df_notas.select(avg("nota").alias("media_nota"), "escola")
B. df_notas.groupBy("escola").agg(avg("nota").alias("media_nota"))
C. df_notas.groupBy("escola").avg("media_nota")
D. df_notas.withColumn("media_nota", avg("nota")).groupBy("escola")

R7.3 · Transformação e modelagem de dados

Na Plataforma de Cursos Aprender+, os eventos de acesso dos alunos chegam a cada minuto como arquivos novos, nunca são alterados e só crescem. A equipe quer uma tabela silver que processe cada evento uma única vez, com baixa latência, em Lakeflow pipelines. Qual tipo de dataset usar?

A. Tabela externa em CSV, atualizada por um job que regrava a pasta de arquivos a cada minuto
B. Materialized view, que recalcula o resultado inteiro a cada atualização do pipeline
C. View temporária, que materializa os eventos em disco a cada atualização do pipeline
D. Streaming table, que processa cada registro uma vez a partir de uma fonte só de inserções

R7.4 · Transformação e modelagem de dados

Na Rede de Clínicas Odonto Sorriso, a silver de agendamentos não deve conter linhas com nulo em nenhuma coluna, porque todas são obrigatórias no sistema de destino. Qual código PySpark remove essas linhas?

A. df.dropDuplicates()
B. df.na.drop("all")
C. df.na.fill("")
D. df.na.drop()

Respostas comentadas

R7.1. B) CONSTRAINT cep_valido EXPECT (cep IS NOT NULL) ON VIOLATION DROP ROW

Por quê: Expectations têm três ações: warn (padrão, mantém o registro), drop (EXPECT ... ON VIOLATION DROP ROW) e fail (EXPECT ... ON VIOLATION FAIL UPDATE). As métricas de warn e drop aparecem na aba Data quality do pipeline e no event log.

✗ A) Incorreta. FAIL UPDATE faz a atualização falhar ao encontrar um registro inválido; o pipeline não continuaria.
✗ C) Incorreta. Sem ação definida a expectation só avisa: os registros inválidos são gravados no destino.
✗ D) Incorreta. CHECK é uma restrição de tabela Delta: a transação que a viola falha, e não há descarte linha a linha com métricas.

> **Dica de prova:** Descartar inválidos e seguir com métricas = EXPECT ... ON VIOLATION DROP ROW.

R7.2. B) df_notas.groupBy("escola").agg(avg("nota").alias("media_nota"))

Por quê: groupBy(colunas).agg(funções) equivale ao GROUP BY do SQL. O método alias é a forma prática de nomear colunas agregadas.

✗ A) Incorreta. Misturar uma agregação com uma coluna não agregada sem groupBy gera erro de análise.
✗ C) Incorreta. Calcula a média de uma coluna media_nota que não existe em df_notas.
✗ D) Incorreta. Agregações não são usadas em withColumn sem janela, e groupBy sem agg não produz resultado.

> **Dica de prova:** Agregar e nomear = groupBy(...).agg(func(...).alias("nome")).

R7.3. D) Streaming table, que processa cada registro uma vez a partir de uma fonte só de inserções

Por quê: Uma streaming table é uma tabela Delta gerenciada pelo pipeline, com suporte a processamento incremental. Se a origem tiver atualizações e exclusões, use AUTO CDC em vez de append.

✗ A) Incorreta. Regravar arquivos a cada minuto não é incremental e perde as garantias de uma tabela Delta.
✗ B) Incorreta. Materialized views servem a transformações e agregações cujo resultado precisa refletir o estado atual da origem; não são feitas para baixa latência.
✗ C) Incorreta. Views em pipelines não são persistidas e servem a etapas intermediárias.

> **Dica de prova:** Fonte que só cresce + processar cada linha uma vez = streaming table.

R7.4. D) df.na.drop()

Por quê: na.drop(how) aceita "any" (padrão) ou "all" e o parâmetro subset para limitar as colunas avaliadas.

✗ A) Incorreta. dropDuplicates remove linhas repetidas, não linhas com nulos.
✗ B) Incorreta. Com "all", só saem as linhas em que todas as colunas são nulas.
✗ C) Incorreta. na.fill preenche os nulos de colunas de texto com string vazia; não remove as linhas.

> **Dica de prova:** Remover linhas com qualquer nulo = na.drop() (any é o padrão).

DIA DA PROVA

Estratégia de prova

A prova tem 45 questões pontuadas de múltipla escolha com resposta única, em 90 minutos (podem vir questões extras não pontuadas, com o tempo já incluído), e cada enunciado é um cenário de empresa com um requisito decisivo escondido no texto (menor privilégio, sem copiar, sem código, sem parar o stream, só uma execução). As alternativas erradas costumam ser funcionalidades reais aplicadas ao problema errado, opções de nome parecido (schemaHints x inferColumnTypes, Minimum time x Wait after last change) ou sintaxes inventadas. O guia oficial vigente usa os nomes atuais (Lakeflow Jobs, Lakeflow Spark Declarative Pipelines, Declarative Automation Bundles, Git folders), mas as questões de exemplo dele vêm de uma versão anterior e ainda trazem termos legados, como high-concurrency cluster; reconheça o nome atual e o antigo de cada recurso (AUTO CDC/APPLY CHANGES, standard/shared, dedicated/single user). Código de dados vem em SQL sempre que possível e, nos demais casos, em Python. Transformação (22%), Ingestão (21%), Lakeflow Jobs (16%) e Governança (15%) somam quase três quartos da prova.

Como responder

✓ Grife o requisito decisivo do enunciado (menor privilégio, sem copiar, só leitura, uma única execução, sem parar) antes de ler as alternativas.

✓ Com cerca de 2 minutos por questão, responda de primeira as de regra direta (Run if, modos do Auto Loader, comandos bundle) e guarde tempo para as de código.

✓ Em alternativas de código, compare só o que muda entre elas ('any' x 'all', & x and, explode x explode_outer, UNION x UNION ALL) e teste cada uma contra o pedido.

✓ Elimine primeiro sintaxes inventadas (WITH INHERITANCE, --default, GRANT OWNERSHIP, ZORDER BY AUTO, TO JOB); se o enunciado usar um nome antigo (Repos, Databricks Asset Bundles, APPLY CHANGES, shared, single user), traduza para o nome atual antes de julgar.

✓ Em perguntas de privilégio, liste o que a ação exige e o que já foi concedido; a resposta é a diferença, sem privilégios amplos.

✓ Em perguntas de ingestão e computação, comece pela opção mais gerenciada que atende (conector gerenciado do Lakeflow Connect, serverless) e só desça se algum requisito impedir (fonte sem conector, R, RDD, init script).

✓ Em diagnóstico de desempenho, identifique o sintoma exato (Max x p75, uma tarefa só, spill, lacuna) antes de olhar as correções; mais workers raramente é a resposta.

✓ Distribua o estudo pelos pesos 6/21/22/16/10/10/15 e não deixe Plataforma, com só 3 questões, consumir tempo demais.

Erros que derrubam candidatos

✗ Usar Run if para decidir por valor de parâmetro ou task value em vez de If/else condition.

✗ Esquecer USE CATALOG e USE SCHEMA ao conceder SELECT, ou achar que ALL PRIVILEGES dá poder de conceder.

✗ Escolher DENY para bloquear acesso no Unity Catalog, onde o comando não é suportado.

✗ Confundir os modos de evolução de esquema do Auto Loader e qual é o padrão com e sem esquema informado.

✗ Achar que COPY INTO duplica dados ao rodar de novo na mesma pasta (é idempotente).

✗ Trocar --var (deploy) por --params (run) nos comandos de bundle.

✗ Diagnosticar skew quando o estágio tem uma tarefa só, ou propor mais workers para uma tarefa sobrecarregada.

Na semana da prova

- ☐ Refaça os simulados focando nas explicações das alternativas erradas dos domínios de maior peso: Transformação, Ingestão, Lakeflow Jobs e Governança.
- ☐ Monte uma folha única com as tabelas que mais caem: condições de Run if, modos do Auto Loader, opções do COPY INTO, precedência de variáveis de bundle e privilégios do Unity Catalog.
- ☐ Revise os pares de nome atual e antigo (Lakeflow Jobs/Databricks Workflows, Lakeflow pipelines/Delta Live Tables, AUTO CDC/APPLY CHANGES, Declarative Automation Bundles/Databricks Asset Bundles, Git folders/Repos, standard/shared, dedicated/single user), porque os dois podem aparecer no enunciado.
- ☐ Treine leitura de código PySpark e SQL em voz alta, conferindo argumentos padrão como how do join, 'any' do na.drop e DISTINCT do UNION.
- ☐ Confirme o agendamento, o modo de aplicação (online ou em centro de testes) e lembre que não é permitido consultar nada durante a prova; a nota de corte não é divulgada, então mire 70% ou mais nos simulados.
- ☐ Na véspera, só revise a folha única e os pontos de lembrar de cada domínio; durma bem.

REVISÃO FINAL

Folha de cola

Leia na véspera e na manhã da prova: os gatilhos e as pegadinhas mais importantes de cada domínio.

1. Databricks Intelligence Platform

‣ Cluster ligado à noite e no fim de semana sem comandos → **Auto termination**

‣ Dezenas de analistas SQL, picos imprevisíveis, partida em segundos → **SQL warehouse serverless**

‣ Jobs Python curtos esperando minutos de partida, sem querer escolher instâncias → **Serverless compute para jobs**

‣ Jobs de produção em all-purpose compartilhada com analistas → **Jobs compute ou serverless para jobs**

‣ Runtime ML, GPU, RDD ou R em computação clássica → **Modo de acesso dedicated**

⚠ Modo de acesso define quem usa, não quando desliga. Desligar por inatividade é auto termination.

⚠ Pin só preserva a configuração de uma computação encerrada por mais de 30 dias; não desliga nada.

⚠ Mesmo sem workers, o driver continua ligado e cobrado; o certo é auto termination.

2. Governança e segurança

‣ Arquivos mantidos ou lidos por sistema fora do Databricks → **Tabela externa**

‣ Tabela gerenciada excluída por engano há dois dias → **UNDROP TABLE**

‣ Externa Delta para gerenciada preservando nome, grants e histórico → **ALTER TABLE ... SET MANAGED**

‣ Consulta falha mesmo com SELECT na tabela → **Falta USE CATALOG e/ou USE SCHEMA**

‣ UPDATE e DELETE na tabela → **MODIFY**

⚠ Sem USE CATALOG e USE SCHEMA nos pais, a leitura falha.

⚠ ALL PRIVILEGES não inclui MANAGE; conceder exige MANAGE ou propriedade.

⚠ O Unity Catalog não usa GRANT OPTION; delegação é MANAGE ou propriedade.

3. Implementação de CI/CD

‣ Começar bundle novo com estrutura recomendada → **databricks bundle init**

‣ Conferir YAML e variáveis resolvidas sem implantar → **databricks bundle validate --output json**

‣ Ver o que seria criado, atualizado ou removido → **databricks bundle plan**

‣ Implantar em produção com dev como default → **databricks bundle deploy -t prod**

‣ Valor só para uma execução → **databricks bundle run --params**

⚠ É o rebase que reescreve e força o push; para iniciantes, merge.

⚠ Reset substitui o branch e força o push: local e remoto perdem as alterações.

⚠ Não há stash: as alterações acompanham a troca se não conflitarem.

4. Ingestão e carga de dados

‣ Fonte SaaS suportada com mínimo de manutenção → **Conector gerenciado do Lakeflow Connect (streaming tables)**

‣ Custo clássico contínuo com o pipeline de ingestão parado → **Ingestion gateway clássico sempre ligado**

‣ Diretório com milhões de arquivos e descoberta lenta → **Auto Loader com file notification e file events**

‣ Testar a carga sem gravar nada → **COPY INTO ... VALIDATE**

‣ Só arquivos cujo nome começa com um prefixo → **PATTERN com glob**

⚠ É idempotente por padrão: arquivos já carregados são ignorados; só force = true recarrega.

⚠ Arquivos já carregados continuam ignorados; o que permite recarregar é COPY_OPTIONS ('force' = 'true').

⚠ FAILFAST só falha em registro malformado; se não houver erro, grava. Testar sem gravar é VALIDATE.

5. Solução de problemas, monitoramento e otimização

‣ Max de 14 minutos e percentil 75 de 25 segundos no estágio → **Skew: AQE skew join e salting**

‣ Estágio mais longo com uma única tarefa e row_number() OVER (ORDER BY ...) → **Janela sem PARTITION BY**

‣ Estatísticas de spill no shuffle de uma agregação → **shuffle partitions auto ou mais memória**

‣ Escrita com arquivos grandes demais e pequenos demais → **Hint REBALANCE**

‣ Minutos sem jobs Spark, driver ocupado, workers ociosos → **Código não Spark no driver**

⚠ Skew são poucas tarefas lentas entre muitas; uma tarefa só vem de janela sem PARTITION BY, gzip ou coalesce(1).

⚠ Mais workers não dividem uma tarefa sobrecarregada nem um estágio de tarefa única.

⚠ Menos partições concentram mais dados; o caminho é AQE skew join e salting.

6. Trabalho com Lakeflow Jobs

‣ Limpeza que deve rodar sempre ao final → **Run if All done**

‣ Abrir chamado se pelo menos uma extração falhar → **Run if At least one failed**

‣ Consolidar se ao menos um provedor responder → **Run if At least one succeeded**

‣ Dependência desabilitada e a tarefa seguinte parou de rodar → **Run if None failed**

‣ Rodar ramo só quando um valor calculado for maior que zero → **If/else condition com {{tasks.X.values.Y}}**

⚠ A tarefa que conta rejeições termina com sucesso; desvio por valor é If/else com task value.

⚠ Run if não lê parâmetros; use If/else com {{job.parameters.ambiente}} == prod.

⚠ Repair reexecuta tarefas que falharam; outra data uma vez é Run now with different parameters.

7. Transformação e modelagem de dados

‣ Remover só quando todas as colunas indicadas forem nulas → **na.drop('all', subset=[...])**

‣ Valores inválidos devem virar NULL sem falhar com ANSI → **try_cast**

- Data em texto dd/MM/yyyy → **to_date(coluna, 'dd/MM/yyyy')**
- Dimensão pequena com fato de bilhões de linhas e shuffle pesado → **Hint BROADCAST na tabela pequena**
- Plano virou broadcast sozinho durante a execução → **AQE**

⚠ Sem o primeiro argumento vale 'any'; para exigir todas nulas, use na.drop('all', subset=[...]).

⚠ String vazia não é nula; para trocá-la use na.replace([''], ['NAO_INFORMADO'], subset=[...]).

⚠ A subtração já virou nulo antes; aplique coalesce no operando: valor - coalesce(desconto, 0).

GABARITO

Respostas do diagnóstico

D1. A) Autoscaling com mínimo e máximo de workers, ajustando os nós à carga → revise o capítulo 1

Por quê: As boas práticas de computação clássica recomendam autoscaling para tarefas longas com carga variável, além de instance pools para reduzir partidas e políticas de computação para padronizar configurações.

✗ B) Incorreta. Auto termination encerra a computação inteira após um período sem comandos; não reduz workers durante a execução.

✗ C) Incorreta. Mais workers fixos aumentam o custo nas fases leves.

✗ D) Incorreta. Um único nó não daria conta das fases pesadas, que precisam de paralelismo.

> **Dica de prova:** Carga variável na mesma execução = autoscaling (mínimo e máximo).

D2. D) Usar jobs compute (ou serverless para jobs), criado para a execução e encerrado ao final → revise o capítulo 1

Por quê: All-purpose compute atende análise interativa e pode ser compartilhada. Jobs compute é efêmero, isolado por execução e configurado na definição do job; serverless para jobs elimina a gestão de infraestrutura.

✗ A) Incorreta. SQL warehouse só serve de computação para tarefas SQL ou notebooks inteiramente em SQL.

✗ B) Incorreta. Um all-purpose ocioso continua cobrando DBU e instâncias durante a inatividade, e os jobs seguem misturados ao uso interativo.

✗ C) Incorreta. Pools reduzem a partida de clusters, mas o problema é usar all-purpose compartilhado para produção.

> **Dica de prova:** Produção = jobs compute ou serverless; all-purpose é para uso interativo.

D3. D) O serverless não tem versão fixa: a Databricks atualiza o runtime automaticamente para todos → revise o capítulo 1

Por quê: Workloads serverless rodam sempre na versão mais recente do runtime, divulgada nas release notes. O serverless exige workspace com Unity Catalog; workspaces legados sem Unity Catalog não têm acesso a ele.

✗ A) Incorreta. Init scripts no escopo da computação não são suportados no serverless.

✗ B) Incorreta. O serverless não fica preso a uma versão escolhida na criação do workspace.

✗ C) Incorreta. Em serverless não se escolhe a versão do Databricks Runtime como na computação clássica.

> **Dica de prova:** Serverless = sem versão fixa, atualizado automaticamente pela Databricks.

D4. A) ALTER TABLE prd.cad.beneficiarios ALTER COLUMN telefone SET MASK prd.seg.mascara_tel → revise o capítulo 2

Por quê: Máscaras de coluna são UDFs SQL que recebem o valor da coluna e devolvem o original ou uma versão mascarada, com tipo compatível. Cada coluna tem uma máscara. Para aplicar a muitas tabelas, a Databricks recomenda políticas ABAC.

✗ B) Incorreta. UPDATE grava o valor mascarado na tabela e destrói o dado original para todos, inclusive para o atendimento.

✗ C) Incorreta. Filtro de linha usa função booleana para esconder linhas; não mascara valores de uma coluna.

✗ D) Incorreta. Concessões no Unity Catalog são em nível de objeto; o mascaramento por usuário é feito com máscara de coluna.

> **Dica de prova:** Esconder valor de coluna por grupo = ALTER COLUMN ... SET MASK função.

D5. C) GRANT SELECT ON TABLE prd.frota.viagens TO `7c1e9a2b-55d0-4c1f-9e3a-2f6b8d0c4a11` → revise o capítulo 2

Por quê: Principais no Unity Catalog são usuários, grupos e entidades de serviço. Para jobs de produção, a Databricks recomenda Run as com entidade de serviço e concessões para essa identidade (ou para um grupo que a contenha).

✗ A) Incorreta. A sintaxe AS APPLICATION não existe no GRANT.

✗ B) Incorreta. Privilégios são concedidos a usuários, grupos ou entidades de serviço, não a jobs.

✗ D) Incorreta. Não existe a palavra-chave SERVICE_PRINCIPAL no GRANT; a entidade é identificada pelo applicationId.

> **Dica de prova:** GRANT para entidade de serviço = applicationId entre crases.

D6. A) EXECUTE na função prd.seg.f_regiao → revise o capítulo 2

Por quê: A função do filtro precisa ser uma UDF SQL registrada no Unity Catalog; lógica em Python ou Scala deve ser encapsulada por uma UDF SQL. Remova o filtro (DROP ROW FILTER) antes de excluir a função, para não deixar a tabela inacessível.

✗ B) Incorreta. MODIFY só seria exigido se a mesma instrução também alterasse o esquema da tabela.

✗ C) Incorreta. REFRESH serve para atualizar materialized views.

✗ D) Incorreta. APPLY TAG permite aplicar tags; não autoriza usar a função.

> **Dica de prova:** Aplicar filtro ou máscara = EXECUTE na função + dono (ou MANAGE e SELECT) na tabela.

D7. D) Declarar uma variável com type: complex contendo a definição e usar ${var.cluster} em new_cluster → revise o capítulo 3

Por quê: Uma variável é string, a menos que type seja complex, que é o único valor válido desse campo. A validação falha se type for complex e o default for um valor único.

✗ A) Incorreta. Um ID de job não é uma definição de cluster.

✗ B) Incorreta. Variáveis simples são strings; estruturas devem usar uma variável complexa.

✗ C) Incorreta. O lookup devolve o ID de um cluster existente, não uma definição de job cluster por job.

> **Dica de prova:** Estrutura reaproveitável no bundle = variável type: complex.

D8. B) Resolver na UI do Git folder, mantendo as linhas desejadas sem os marcadores, ou clicar em Abort para desfazer → revise o capítulo 3

Por quê: Conflitos surgem quando o Git não consegue conciliar mudanças nas mesmas linhas durante pull, merge ou rebase. Para quem está começando, a Databricks recomenda merge em vez de rebase, porque merge não reescreve o histórico.

✗ A) Incorreta. Reset substitui o branch pelo estado de outro e descarta alterações; não mescla versões.

✗ C) Incorreta. Os marcadores de conflito precisam ser removidos antes de concluir a operação.

✗ D) Incorreta. A UI do Git folder mostra os arquivos em conflito e as opções de resolução.

> **Dica de prova:** Conflito no Git folder = resolver na UI ou Abort.

D9. A) Mapeamento permissions de nível superior com group_name operacao e level CAN_RUN → revise o capítulo 3

Por quê: Permissões não podem se sobrepor: o mesmo usuário, grupo ou entidade de serviço não pode estar no permissions de nível superior e no de um recurso. Para evitar edições após o deploy, use uma pasta imutável.

✗ B) Incorreta. Lookup resolve IDs de objetos, não permissões.

✗ C) Incorreta. Declarar por job repete a configuração, e CAN_MANAGE permitiria editar.

✗ D) Incorreta. run_as define a identidade de execução, não quem pode executar; e o run_as de nível superior não aceita grupos.

> **Dica de prova:** Permissão para todos os recursos do bundle = permissions de nível superior.

D10. A) VALIDATE ALL → revise o capítulo 4

Por quê: VALIDATE está disponível no Databricks SQL e no Databricks Runtime 10.4 LTS ou superior. É útil para testar arquivos de uma fonte nova antes da primeira carga.

✗ B) Incorreta. PATTERN apenas seleciona arquivos; a carga seria executada.

✗ C) Incorreta. FAILFAST faz a leitura falhar em registros malformados, mas o comando continuaria gravando se não houvesse erro.

✗ D) Incorreta. force = false é o comportamento padrão (idempotente) e grava os dados normalmente.

> **Dica de prova:** Testar uma carga do COPY INTO sem gravar = VALIDATE.

D11. D) Os metadados dos arquivos descobertos ficam em um RocksDB no checkpoint, e o stream retoma de onde parou → revise o capítulo 4

Por quê: Não é preciso manter estado manualmente para obter tolerância a falhas: o checkpoint guarda o progresso. Cada fonte gravada em uma mesma tabela precisa de um checkpoint próprio.

✗ A) Incorreta. O Auto Loader não reprocessa a pasta inteira; ele continua a partir do estado guardado.
✗ B) Incorreta. Por padrão (cleanSource desligado), o Auto Loader não apaga nem move arquivos da origem.
✗ C) Incorreta. O Delta Lake não deduplica linhas automaticamente.

> **Dica de prova:** Exatamente uma vez no Auto Loader = estado no checkpoint (RocksDB).

D12. D) Executar databricks connections create com --json pela Databricks CLI → revise o capítulo 4

Por quê: A conexão é um objeto protegível do Unity Catalog que guarda os detalhes de autenticação da fonte. Conectores que só aceitam OAuth pelo navegador (U2M) não podem ter a conexão criada de forma programática.

✗ A) Incorreta. Variáveis de bundle não substituem a conexão e expor credenciais em texto é inseguro.
✗ B) Incorreta. As credenciais ficam na conexão do Unity Catalog, não em arquivos de volume.
✗ C) Incorreta. Widgets recebem parâmetros de notebook; não criam objetos de conexão no Unity Catalog.

> **Dica de prova:** Criar conexão do Unity Catalog por automação = databricks connections create --json.

Agora, treine no formato real

Faça as 6 provas completas do curso Databricks Data Engineer Associate: 270 questões inéditas, tempo do exame, explicação de cada alternativa e link da documentação para revisar.

Ver as 6 provas completas e o cupom do mês →

Até o dia da prova, uma revisão por dia no seu feed

Instagram @strecht.simulados
Um post de prova por dia: questão comentada, pegadinhas e o "não confunda" de cada certificação.
Seguir no Instagram →

YouTube Strecht Simulados
Shorts diários com pegadinhas de prova e, aos sábados, um simulado comentado.
Inscrever-se no canal →

Material independente de preparação, sem afiliação ou endosso de Databricks. As questões são inéditas e não reproduzem a prova real.