Strecht Simulados · Guia de preparação



Guia de preparação DP-750

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.

4
capítulos por domínio

28
questões comentadas

70%
nota de corte

6
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 DP-750: 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 50 questões e 120 minutos, no formato do exame. Referência de aprovação: 70%.

Domínio (capítulo)	Peso	Neste guia
1. Configurar e ajustar um ambiente do Azure Databricks	15-20%	18 conceitos · 14 pegadinhas
2. Implantar e manter pipelines de dados e cargas de trabalho	30-35%	18 conceitos · 14 pegadinhas
3. Preparar e processar dados	30-35%	18 conceitos · 14 pegadinhas
4. Proteger e governar objetos do Unity Catalog	15-20%	17 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 · Configurar e ajustar um ambiente do Azure Databricks

A Mineradora Veio de Ouro usa o Unity Catalog e computação em modo de acesso standard, compartilhada por vários engenheiros. Uma tarefa precisa de um driver JDBC em JAR, guardado em um volume, e de uma biblioteca Maven de geoprocessamento. Ao instalar os dois no cluster, a operação é bloqueada. O que deve ser feito para liberar a instalação mantendo o modo standard?

A. Pedir ao administrador do metastore que inclua o JAR e a coordenada Maven na allowlist
B. Conceder ao grupo de engenheiros o privilégio READ VOLUME no volume onde está o JAR
C. Copiar o JAR para o DBFS root e instalá-lo a partir de lá com um init script global
D. Trocar o modo de acesso para No isolation shared, que aceita bibliotecas sem restrição de origem

D2 · Configurar e ajustar um ambiente do Azure Databricks

Na Cooperativa de Saúde Unimais, o grupo engenharia_risco precisa criar esquemas novos dentro do catálogo analytics para seus projetos, sem poder criar catálogos nem administrar privilégios do catálogo. Hoje o grupo não tem nenhum privilégio no catálogo. Quais privilégios você deve conceder no catálogo analytics?

A. CREATE CATALOG no metastore e USE SCHEMA
B. USE CATALOG e CREATE SCHEMA
C. CREATE SCHEMA, sem USE CATALOG no catálogo
D. MANAGE e ALL PRIVILEGES no catálogo

D3 · Configurar e ajustar um ambiente do Azure Databricks

A Consultoria Dados Certos vai migrar notebooks interativos de um cluster clássico para computação serverless. Um dos notebooks usa sc.parallelize e transformações com a API RDD, além de df.cache() para reaproveitar um DataFrame em várias células. O que acontecerá e o que você deve fazer?

A. O código falhará; adicione um init script de computação que habilite a API RDD
B. O código falhará; reescreva com a API DataFrame e remova o cache()
C. O código rodará, mas é preciso ativar spark.rdd.enabled na sessão
D. O código rodará sem mudanças, pois o serverless aceita APIs RDD

D4 · Implantar e manter pipelines de dados e cargas de trabalho

Na Mineradora Terra Firme, o job de produção roda notebooks copiados para uma pasta do workspace, e já houve execução de código não revisado, alterado direto na cópia. A equipe quer que o job execute exatamente o código aprovado no repositório, identificado por uma tag de release. O que você deve configurar?

A. Definir a fonte das tarefas como Git provider, apontando para a tag de release
B. Manter a cópia no workspace e restringir CAN EDIT na pasta ao time de plataforma
C. Exportar os notebooks como DBC a cada release e importá-los na pasta do job
D. Criar uma Git folder pessoal do engenheiro líder e apontar o job para ela

D5 · Implantar e manter pipelines de dados e cargas de trabalho

Na Editora Página Viva, uma engenheira trabalha na branch feature/precos em uma Git folder do Azure Databricks e usou a opção Merge para trazer a main para a sua branch. A Git folder acusou conflito no arquivo calcula_precos.py, alterado por um colega nas mesmas linhas. Ela quer manter parte das duas versões e concluir a operação sem sair do Azure Databricks. O que ela deve fazer?

A. Escolher Take all incoming changes e concluir a operação de merge
B. Mover a Git folder para a lixeira, clonar o repositório de novo e colar o arquivo
C. Executar Reset da branch para a main e refazer as próprias alterações do zero
D. Editar o arquivo, remover os marcadores, marcar como resolvido e continuar o merge

D6 · Implantar e manter pipelines de dados e cargas de trabalho

A Operadora de Telecom Conecta Mais já tem uma regra de alerta de pesquisa de logs no Azure Monitor que detecta falhas de jobs do Azure Databricks na tabela DatabricksJobs. Agora, quando o alerta disparar, o plantonista deve receber um SMS e um sistema de chamados deve ser acionado por um webhook. O que você deve configurar?

A. Uma pasta de trabalho do Azure Monitor com gráfico de falhas por dia
B. Uma configuração de diagnóstico que envie os logs para um Event Hub
C. Uma notificação de falha no próprio job com o número do plantonista
D. Um grupo de ações com ação de SMS e ação de webhook, associado à regra de alerta

D7 · Preparar e processar dados

Uma rede de clínicas quer trazer diariamente tabelas de incidentes e chamados do ServiceNow para o Unity Catalog. A equipe de dados é pequena, não quer manter código de paginação de API e precisa de cargas incrementais governadas, com agendamento e monitoramento. Qual ferramenta de ingestão você deve escolher?

A. Criar um pipeline de ingestão com o conector gerenciado do ServiceNow no Lakeflow Connect
B. Escrever um notebook que pagina a API REST do ServiceNow e grava arquivos JSON em um volume
C. Montar no DBFS o armazenamento exportado pelo ServiceNow e ler os arquivos com spark.read
D. Criar um catálogo estrangeiro do Lakehouse Federation apontando para a instância do ServiceNow

D8 · Preparar e processar dados

A Concessionária de Rodovias Via Livre lê eventos de pedágio do Azure Event Hubs pelo endpoint compatível com Kafka, em um notebook com Structured Streaming. Na primeira execução da consulta, a equipe quer processar também os eventos que ainda estão retidos no hub, e a política de segurança proíbe deixar a connection string em texto no código. Qual configuração atende aos dois requisitos?

A. startingOffsets = 'latest' e a connection string vinda de um widget preenchido pelo job
B. startingOffsets = 'earliest' e a connection string no notebook, protegido por permissão de leitura
C. kafka.group.id com um grupo novo e a connection string em texto em variável de ambiente
D. startingOffsets = 'earliest' e a connection string lida de um escopo de segredo no kafka.sasl.jaas.config

D9 · Preparar e processar dados

O cadastro de produtos de um e-commerce muda o preço de lista e a descrição várias vezes por mês. A área de negócio confirmou que precisa só do valor atual de cada atributo, pois os relatórios históricos usam os preços gravados nas vendas. Qual tipo de dimensão de alteração lenta (SCD) você deve implementar na dim_produto?

A. SCD tipo 1, sobrescrevendo os atributos alterados na linha existente
B. SCD tipo 2 com coluna is_current e versões anteriores
C. SCD tipo 2, com uma nova linha e datas de vigência a cada alteração de atributo
D. Snapshot diário completo da dimensão, particionado por data

D10 · Proteger e governar objetos do Unity Catalog

Analistas de uma cooperativa de crédito não entendem a tabela gold.inadimplencia nem a coluna dpd. A governança quer que as descrições apareçam no Catalog Explorer, ajudem a busca e o Genie e sejam gravadas por código no pipeline de implantação. O que você deve executar?

A. SET TAG ON TABLE gold.inadimplencia descricao = '...' e uma tag na coluna dpd com o texto
B. CREATE VIEW gold.inadimplencia_doc AS SELECT dpd AS dias_em_atraso FROM gold.inadimplencia
C. COMMENT ON TABLE gold.inadimplencia IS '...' e ALTER TABLE gold.inadimplencia ALTER COLUMN dpd COMMENT '...'
D. ALTER TABLE gold.inadimplencia SET TBLPROPERTIES ('descricao' = '...') para a tabela e outra para a coluna dpd

D11 · Proteger e governar objetos do Unity Catalog

No catálogo vendas_prd da Varejista Bom Preço, o esquema gold recebe tabelas novas toda semana. Os analistas do grupo analistas_bi precisam consultar as tabelas atuais e as futuras desse esquema, sem que o engenheiro tenha de emitir um GRANT a cada tabela criada. Quais concessões atendem com o menor privilégio?

A. GRANT SELECT em cada tabela do esquema gold, repetido por um job semanal de permissões
B. GRANT USE SCHEMA no esquema gold e SELECT ON ANY FILE ao grupo
C. GRANT USE CATALOG no catálogo, mais USE SCHEMA e SELECT no esquema gold, ao grupo
D. GRANT ALL PRIVILEGES no catálogo vendas_prd ao grupo analistas_bi

D12 · Proteger e governar objetos do Unity Catalog

Uma aplicação de integração da Transportadora Carga Leve roda fora do Azure Databricks e chama a API REST do workspace a cada 15 minutos para disparar jobs. Hoje ela usa um token pessoal (PAT) de um engenheiro que saiu da empresa, e o token expirou. A segurança quer uma identidade não humana e credenciais de curta duração geradas automaticamente. O que você deve configurar?

A. Um novo PAT de outro engenheiro com validade de 365 dias
B. O usuário e a senha do Microsoft Entra ID de uma conta de serviço compartilhada
C. Um PAT gerado para a entidade de serviço sem data de expiração
D. Uma entidade de serviço com segredo OAuth, usando o fluxo OAuth M2M

01

CAPÍTULO 1 · 15-20% DA PROVA

Configurar e ajustar um ambiente do Azure Databricks

Aqui eu preciso escolher a computação certa para cada carga (job compute, serverless, all-purpose, SQL warehouse, single node), o modo de acesso (standard ou dedicated), as bibliotecas e as permissões de computação, e ainda montar a hierarquia do Unity Catalog: catálogos, esquemas, volumes, tabelas gerenciadas e externas, views, materialized views, catálogos estrangeiros e instruções do Genie. O domínio vale 15-20% da prova. Os enunciados trazem um requisito de custo, de partida rápida ou de isolamento e pedem a opção mínima que atende. Onde mais erro: pool x serverless, standard x dedicated quando aparece RDD ou GPU, MANAGED LOCATION x LOCATION e o privilégio exato para criar catálogo ou esquema.

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

Tópicos-chave

Job compute x all-purpose

Job agendado em produção roda em job compute, criado no início da execução e encerrado no fim, ou em serverless para jobs. All-purpose é para trabalho interativo e tem tarifa de DBU mais cara; deixá-lo ligado o dia todo para os jobs só aumenta o custo.

Computação serverless

Partida rápida e infraestrutura gerenciada pelo Azure Databricks, sem escolher VM nem versão de runtime. Não roda RDD, R, init scripts nem cache(), e a maioria das configurações Spark não pode ser alterada. Dependências vão no painel Environment do notebook, que o job também usa, e não como biblioteca de cluster.

Modos de acesso standard e dedicated

Standard (antes chamado compartilhado) isola usuários e governa com Unity Catalog, mas não suporta APIs de RDD nem R. Dedicated (antes single user) atende RDD, R, Databricks Runtime ML e GPU; atribuído a um grupo, os comandos rodam com as permissões do grupo, não de cada pessoa. No isolation shared é legado, sem isolamento nem governança.

Bibliotecas

Dependência só de um notebook: %pip install e, se mexer em pacote já importado, dbutils.library.restartPython(). Para o cluster inteiro, requirements.txt nos workspace files como biblioteca de computação; wheel compartilhado vem de um volume do Unity Catalog, nunca do DBFS root. Remover biblioteca de cluster só vale após reiniciar o cluster.

Allowlist do metastore

No modo standard, JAR, coordenadas Maven e init scripts ficam bloqueados até o administrador do metastore incluí-los na allowlist. Copiar para o DBFS root e usar init script global é prática legada e não contorna o bloqueio.

Autoscaling, auto termination e pools

Volume variável pede autoscaling com mínimo e máximo de workers; cluster ocioso gastando pede terminação automática. Pool com instâncias ociosas reduz a partida de clusters clássicos (paga a VM, não paga DBU ocioso), mas SQL warehouses não usam pools.

Instâncias spot

Use spot só nos workers, com driver sob demanda e reposição sob demanda quando faltar spot: se o driver for recolhido, a sessão Spark inteira cai. Capacity reservation group do Azure só funciona com instâncias sob demanda.

Photon

Acelera SQL e DataFrames pesados: MERGE INTO, CTAS de grandes volumes, joins e agregações sobre bilhões de linhas. Não acelera UDFs nem a API de RDD (que caem no motor Spark tradicional) e quase não ajuda consultas que já rodam em menos de dois segundos.

Versões e tipos de runtime e nó

Produção estável em computação clássica usa a versão LTS mais recente do Databricks Runtime; versões sem LTS ou beta não servem. ML com GPU e bibliotecas prontas: Databricks Runtime ML em dedicated. Dados pequenos e biblioteca não distribuída: Single node. Spill com CPU sobrando: nós com otimização de memória (série E); série F tem menos memória por núcleo.

SQL warehouse

BI e dashboards com muitos usuários SQL pedem SQL warehouse serverless. Consulta lenta: aumente o tamanho do cluster do warehouse; fila de usuários: aumente o máximo de clusters do escalonamento. Auto Stop só desliga o warehouse ocioso.

Permissões e políticas de computação

CAN ATTACH TO anexa notebooks e vê logs; CAN RESTART também inicia e reinicia; CAN MANAGE edita e exclui. No warehouse, CAN USE executa consultas e CAN MONITOR também vê o histórico e os perfis das consultas de todos. Para limitar quem cria cluster, retire o entitlement Unrestricted cluster creation e conceda CAN USE em uma política de computação.

Hierarquia e privilégios de criação no Unity Catalog

Cada região tem um único metastore por conta. Criar catálogo exige CREATE CATALOG no metastore; criar esquema exige USE CATALOG + CREATE SCHEMA no catálogo. Ambientes se separam com um catálogo por ambiente (dev_, hml_, prd_) com os mesmos esquemas, parametrizando o catálogo.

MANAGED LOCATION e workspace-catalog binding

Isolar fisicamente os dados de um catálogo ou de um esquema: CREATE CATALOG ... MANAGED LOCATION ou CREATE SCHEMA ... MANAGED LOCATION; LOCATION em esquema é sintaxe do Hive. Para restringir um catálogo de produção a certos workspaces, use workspace-catalog binding, com opção somente leitura.

Volumes, tabelas externas e SET MANAGED

Arquivos não tabulares governados vão para volumes (READ VOLUME e WRITE VOLUME); arquivos já existentes e usados por outro sistema pedem volume externo. Tabela com LOCATION exige external location com CREATE EXTERNAL TABLE mais CREATE TABLE no esquema. ALTER TABLE ... SET MANAGED converte externa em gerenciada mantendo nome e histórico (owner, Databricks Runtime 17.3 LTS ou serverless).

Views, materialized views e streaming tables

View comum recalcula a cada consulta; view temporária some com a sessão e a global temporária fica presa à computação que a criou; para compartilhar, CREATE VIEW em um esquema do Unity Catalog com SELECT. Agregação cara e lida muitas vezes: materialized view com SCHEDULE ou REFRESH MATERIALIZED VIEW (incremental pede row tracking na origem). Ingestão incremental em SQL: CREATE STREAMING TABLE ... STREAM read_files.

Lakehouse Federation e tipos de catálogo

Consultar banco externo sem copiar: CREATE CONNECTION (credenciais via secret()) e CREATE FOREIGN CATALOG; uso intenso e recorrente pede ingestão. Dados próprios ficam em catálogo padrão, banco externo em catálogo estrangeiro e dados recebidos por sharing em catálogo criado a partir do share, que é somente leitura.

DDL de manutenção

UNDROP TABLE recupera tabela gerenciada removida há menos de 7 dias; RESTORE só volta versão de tabela existente. DROP SCHEMA ... CASCADE remove esquema com objetos (RESTRICT é o padrão e falha se não estiver vazio). RENAME ou DROP COLUMN em Delta exige delta.columnMapping.mode = 'name'. SHALLOW CLONE testa sem copiar dados; DEEP CLONE faz cópia independente.

Genie Agent (antes Genie space)

Compartilhe com CAN RUN; o warehouse usa a credencial embutida da autora, mas os dados respeitam o SELECT de cada usuário. Regra que vale para todas as perguntas vai nas instruções de texto gerais; termos, valores e joins se corrigem com sinônimos, entity matching e join relationships no knowledge store; fórmula oficial e resposta verificada pedem consulta de exemplo parametrizada ou função SQL (trusted asset).

Macetes para memorizar

💡 Escolha da computação

Agenda = job compute; interativo = all-purpose; SQL e BI = warehouse; sem infra = serverless; pequeno = single node.

💡 Modos de acesso

Standard = todo mundo junto e isolado; dedicated = RDD, ML e GPU. No isolation = museu.

💡 Spot

Spot no braço (workers), nunca na cabeça (driver).

💡 Warehouse

Lenta = mais grande (tamanho); fila = mais clusters.

💡 Permissões de cluster

Attach olha, Restart liga, Manage mexe.

💡 Local de armazenamento

No Unity Catalog o local é MANAGED: MANAGED LOCATION em catálogo e esquema.

💡 Legado na prova

DBFS root, montagem, init script no DBFS e Hive metastore aparecem só como alternativa errada.

Tabela de memorização: não confunda

Quando é...	...e quando é
Pool de instâncias Quero reduzir a partida de clusters clássicos e continuo administrando VM, tamanho e runtime.	**Computação serverless** Quero partida rápida sem gerir infraestrutura nenhuma; só não posso ter RDD, init script, R ou cache().
Modo standard Vários usuários com isolamento, Unity Catalog e cada um com suas permissões.	**Modo dedicated** Precisa de RDD, Databricks Runtime ML ou GPU; atribuído a um grupo, roda com as permissões do grupo.

Aumentar o tamanho do warehouse Uma consulta individual está lenta (latência).	**Aumentar o máximo de clusters** Muitos usuários em fila ao mesmo tempo (concorrência).
Volume Arquivos não tabulares (imagens, PDFs, wheels) governados por READ VOLUME e WRITE VOLUME.	**Tabela externa** Dados tabulares em caminho fixo usado por outro sistema, registrados com CREATE EXTERNAL TABLE sobre external location.
Lakehouse Federation Consulta ocasional ao banco externo sem cópia.	**Ingestão** Uso intenso e recorrente, que pede dados no lakehouse.
SHALLOW CLONE Teste rápido (inclusive MERGE) sem copiar terabytes.	**DEEP CLONE** Cópia independente e durável dos dados.
Materialized view Agregações e joins sobre tabelas que mudam, com resultado guardado e atualizado por agendamento ou REFRESH.	**Streaming table** Fonte só de acréscimo processada de forma incremental, como arquivos chegando em um volume.

Números que caem

Item	Valor
Recuperação de tabela gerenciada removida (UNDROP)	**7 dias por padrão**
Metastores por região na conta	**1**
Runtime mínimo para ALTER TABLE ... SET MANAGED	**Databricks Runtime 17.3 LTS ou serverless (leitores e gravadores em 15.4 LTS ou superior)**
Consultas que quase não ganham com Photon	**as que já rodam em menos de dois segundos**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Criar um instance pool e associá-lo ao SQL warehouse para acelerar as consultas.
Como não cair: SQL warehouses não usam pools; pools servem a clusters clássicos de notebooks e jobs.

⚠ **Alerta de prova:** Ativar spark.rdd.enabled para o código com RDD rodar no serverless.
Como não cair: Não existe configuração que habilite RDD no serverless: reescreva com a API DataFrame e tire o cache().

⚠ **Alerta de prova:** Copiar o JAR ou o wheel para o DBFS root e instalar com init script global.
Como não cair: DBFS root e init scripts no DBFS são legado; o JAR bloqueado no standard vai para a allowlist do metastore, e o wheel vem de um volume.

⚠ **Alerta de prova:** Usar instâncias spot também no driver para economizar mais.
Como não cair: Spot só nos workers; driver sob demanda, senão a sessão Spark inteira cai quando a instância é recolhida.

⚠ **Alerta de prova:** CREATE SCHEMA ... LOCATION para isolar o armazenamento do esquema.
Como não cair: No Unity Catalog é MANAGED LOCATION; LOCATION é sintaxe do Hive. E não existe CREATE EXTERNAL CATALOG com LOCATION.

⚠ **Alerta de prova:** RESTORE TABLE para recuperar uma tabela gerenciada que sofreu DROP.
Como não cair: RESTORE volta versão de tabela existente; tabela removida há menos de 7 dias se recupera com UNDROP TABLE.

⚠ **Alerta de prova:** DROP SCHEMA ... RESTRICT força a remoção dos objetos.
Como não cair: RESTRICT é o padrão e falha quando o esquema não está vazio; quem remove tudo é CASCADE.

⚠ **Alerta de prova:** CTAS para renomear coluna ou para converter tabela externa em gerenciada.
Como não cair: CTAS cria outra tabela, sem histórico e sem permissões. Renomear: column mapping 'name' + RENAME COLUMN; converter: ALTER TABLE ... SET MANAGED.

⚠ **Alerta de prova:** Revogar USE CATALOG dos grupos do ws-dev para isolar o catálogo de produção.
Como não cair: Privilégios valem no metastore inteiro; isolar por workspace é workspace-catalog binding.

⚠ **Alerta de prova:** Criar outro metastore na mesma região só para um projeto ou um workspace.
Como não cair: A conta tem um único metastore por região; separe com catálogos, MANAGED LOCATION e binding.

⚠ **Alerta de prova:** Conceder CAN ATTACH TO para quem precisa religar o cluster.
Como não cair: CAN ATTACH TO não inicia nem reinicia; o mínimo para anexar e religar sem editar é CAN RESTART.

⚠ **Alerta de prova:** Dar CAN USE no warehouse a cada gerente porque o Genie roda com a credencial deles.
Como não cair: O warehouse usa a credencial embutida da autora; o usuário precisa de CAN RUN no Genie Agent e SELECT nas tabelas.

⚠ **Alerta de prova:** Habilitar Photon para acelerar treino de deep learning ou transformações com RDD.
Como não cair: Photon acelera SQL e DataFrames pesados; RDD e UDF caem no Spark tradicional, e treino com GPU pede Runtime ML.

⚠ **Alerta de prova:** Usar uma view no catálogo de desenvolvimento para testar MERGE sobre dados de produção.
Como não cair: View é somente leitura; o teste rápido sem copiar dados é SHALLOW CLONE.

Como cai na prova

📝 **Como cai:** Cenário com custo alto ou partida lenta e quatro tipos de computação.
Como resolver: Identifique se é job agendado, interativo, SQL ou dado pequeno; descarte all-purpose ligado o dia todo para jobs; pool só se a computação continuar clássica; serverless se o pedido é não gerir infraestrutura.

📝 **Como cai:** Código que usa RDD, R, init script ou cache() vai migrar para serverless ou standard.
Como resolver: Esses recursos não rodam lá: reescreva com DataFrame ou use dedicated; nenhuma configuração Spark libera RDD.

📝 **Como cai:** Usuário não consegue criar catálogo, esquema ou tabela externa.
Como resolver: Mapeie o objeto ao privilégio no nível pai: catálogo = CREATE CATALOG no metastore; esquema = USE CATALOG + CREATE SCHEMA; tabela com LOCATION = CREATE EXTERNAL TABLE na external location + CREATE TABLE no esquema.

📝 **Como cai:** Painel lento sobre agregação de bilhões de linhas.
Como resolver: View comum e temporária recalculam; streaming table é para fonte só de acréscimo; a resposta é materialized view com SCHEDULE ou REFRESH.

📝 **Como cai:** Genie respondendo errado ou usuários sem acesso.
Como resolver: Regra global vai em instruções de texto gerais; termos e joins no knowledge store; fórmula oficial em consulta de exemplo ou função SQL; acesso = CAN RUN + SELECT do usuário.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
job agendado de produção, pagar só durante a execução	**Job compute definido no job (ou serverless para jobs)**
sem gerir infraestrutura, partida rápida	**Computação serverless**
RDD, Runtime ML ou GPU com vários usuários	**Modo dedicated atribuído a um grupo**
JAR ou Maven bloqueado em cluster standard	**Allowlist do metastore**
cluster demora a subir	**Pool com mínimo de instâncias ociosas**
clusters de desenvolvimento ociosos gastando	**Terminação automática**
spill com CPU sobrando	**Nós com otimização de memória (série E)**
dados pequenos e biblioteca que roda só no driver	**Single node**
Power BI, dashboards, muitos usuários SQL	**SQL warehouse serverless**
consultar banco externo sem copiar	**CREATE CONNECTION + CREATE FOREIGN CATALOG**
agregação cara, lida muitas vezes, pode atrasar um pouco	**Materialized view com SCHEDULE**
catálogo de produção só visível em um workspace	**Workspace-catalog binding**
tabela gerenciada apagada por engano há poucos dias	**UNDROP TABLE**
Genie confunde termos, valores ou joins	**Sinônimos, entity matching e join relationships**

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

- Serverless não roda RDD, R, init scripts nem cache(); dependências vão no Environment.
- Spot só nos workers; driver sempre sob demanda.
- Unity Catalog usa MANAGED LOCATION; LOCATION em esquema é Hive.
- Um metastore por região; ambientes se separam por catálogo.
- DBFS root, montagens e init scripts no DBFS são sempre alternativa errada.

Essenciais para a prova (checklist)

- ☐ Sei escolher entre job compute, all-purpose, serverless, SQL warehouse e single node para um cenário.
- ☐ Sei quando usar standard e quando usar dedicated, e o efeito de atribuir dedicated a um grupo.
- ☐ Sei instalar dependências com %pip, requirements.txt, wheel em volume e Environment do serverless.
- ☐ Sei configurar autoscaling, auto termination, pools e spot com driver sob demanda.
- ☐ Sei diferenciar tamanho do warehouse de máximo de clusters.
- ☐ Sei as permissões CAN ATTACH TO, CAN RESTART, CAN MANAGE, CAN USE e CAN MONITOR.

- ☐ Sei os privilégios para criar catálogo, esquema, tabela externa e catálogo estrangeiro.
- ☐ Sei usar MANAGED LOCATION, workspace-catalog binding, UNDROP, SET MANAGED e DROP SCHEMA CASCADE.
- ☐ Sei quando usar view, materialized view e streaming table.
- ☐ Sei configurar e compartilhar um Genie Agent e melhorar suas respostas.

Questões de revisão do capítulo

R1.1 · Configurar e ajustar um ambiente do Azure Databricks

Uma equipe de BI de uma rede de academias quer, só com SQL em um SQL warehouse, ingerir de forma incremental os arquivos JSON que chegam ao volume /Volumes/acad/bronze/checkins para uma tabela do Unity Catalog, processando cada arquivo uma vez e atualizando a cada hora. O que você deve criar?

A. CREATE OR REFRESH STREAMING TABLE checkins SCHEDULE EVERY 1 HOUR AS SELECT * FROM STREAM read_files('/Volumes/acad/bronze/checkins', format => 'json')
B. CREATE VIEW checkins AS SELECT * FROM json.`/Volumes/acad/bronze/checkins`
C. CREATE TABLE checkins AS SELECT * FROM read_files('/Volumes/acad/bronze/checkins', format => 'json') e um job de hora em hora
D. CREATE MATERIALIZED VIEW checkins SCHEDULE EVERY 1 HOUR AS SELECT * FROM read_files('/Volumes/acad/bronze/checkins', format => 'json', multiLine => true)

R1.2 · Configurar e ajustar um ambiente do Azure Databricks

Há dois dias, um engenheiro executou por engano DROP TABLE na tabela gerenciada prd_rh.folha.pagamentos. As configurações de recuperação estão no padrão. O que você deve fazer?

A. Executar UNDROP TABLE prd_rh.folha.pagamentos, já que tabelas gerenciadas são recuperáveis por 7 dias
B. Recriar a tabela com CREATE TABLE ... LOCATION apontando para a pasta antiga dos arquivos gerenciados
C. Executar RESTORE TABLE prd_rh.folha.pagamentos TO VERSION AS OF com a última versão antes do DROP
D. Abrir um chamado no suporte, porque o DROP de tabela gerenciada apaga os arquivos na mesma hora

R1.3 · Configurar e ajustar um ambiente do Azure Databricks

Os gestores da Rede de Postos Rota 66 usam um Genie Agent (antes chamado Genie space) sobre as tabelas abastecimentos e postos. Eles perguntam por litros vendidos em São Paulo e faturamento por bandeira, mas o Genie erra: a coluna de estado guarda siglas, a coluna de volume se chama qtd_l e os joins entre as duas tabelas saem errados. O que você deve configurar no knowledge store do Genie Agent?

A. Descrições e sinônimos nas colunas, entity matching no estado e um join relationship entre as tabelas
B. A permissão CAN EDIT para os gestores, para que eles corrijam as consultas geradas pelo Genie
C. Um SQL warehouse maior para o Genie Agent, com mais clusters para responder melhor às perguntas
D. Uma instrução de texto geral pedindo ao Genie para fazer o join pelas chaves corretas das tabelas

R1.4 · Configurar e ajustar um ambiente do Azure Databricks

Um laboratório já tem 2 TB de imagens de exames em um contêiner do ADLS Gen2 coberto por uma external location, e um sistema PACS fora do Azure Databricks também lê esses arquivos. A equipe quer governar o acesso às imagens pelo Unity Catalog sem copiá-las nem mudar o caminho. O que você deve criar?

A. Uma montagem do DBFS do contêiner com dbutils.fs.mount e ACLs na pasta montada
B. Um volume externo (CREATE EXTERNAL VOLUME) com LOCATION apontando para o caminho existente
C. Uma tabela externa em formato binaryFile apontando para o contêiner das imagens
D. Um volume gerenciado no esquema e uma cópia dos 2 TB de imagens para dentro do volume recém-criado

Respostas comentadas

R1.1. A) CREATE OR REFRESH STREAMING TABLE checkins SCHEDULE EVERY 1 HOUR AS SELECT * FROM STREAM read_files('/Volumes/acad/bronze/checkins', format => 'json')

Por quê: Streaming tables são o objeto SQL para ingestão incremental de fontes só de acréscimo, como arquivos em volumes ou mensagens; materialized views servem para transformações e agregações recalculadas.

✗ B) Incorreta. A view lê os arquivos a cada consulta e não materializa dados no Unity Catalog.
✗ C) Incorreta. CTAS lê todos os arquivos a cada execução e recria a tabela; não é incremental.
✗ D) Incorreta. A materialized view recalcula o resultado sobre a fonte; não é o objeto indicado para ingestão incremental de arquivos só de acréscimo.

> **Dica de prova:** Ingestão incremental em SQL: CREATE STREAMING TABLE ... STREAM read_files.

R1.2. A) Executar UNDROP TABLE prd_rh.folha.pagamentos, já que tabelas gerenciadas são recuperáveis por 7 dias

Por quê: DROP de tabela gerenciada marca a tabela como removida e apaga os arquivos depois do período de recuperação (7 dias por padrão, configurável no catálogo ou esquema). DROP de tabela externa remove só os metadados; os arquivos continuam no armazenamento.

✗ B) Incorreta. Não se cria tabela externa sobre o armazenamento gerenciado, e a tabela perderia privilégios e o vínculo de gerenciada.

✗ C) Incorreta. RESTORE volta uma tabela existente a uma versão anterior; não recupera uma tabela que foi removida.

✗ D) Incorreta. Os arquivos de uma tabela gerenciada só são apagados depois do período de recuperação.

> **Dica de prova:** Tabela gerenciada removida há menos de 7 dias: UNDROP TABLE.

R1.3. A) Descrições e sinônimos nas colunas, entity matching no estado e um join relationship entre as tabelas

Por quê: O knowledge store do Genie Agent reúne metadados locais (descrições, sinônimos, colunas ocultas), prompt matching (format assistance e entity matching), join relationships e expressões SQL. Ele melhora a precisão sem alterar o Unity Catalog.

✗ B) Incorreta. Dar CAN EDIT aos consumidores transfere a curadoria para eles e não melhora as respostas para os demais.

✗ C) Incorreta. O tamanho do warehouse afeta desempenho e concorrência, não a interpretação semântica das perguntas.

✗ D) Incorreta. Instruções gerais servem a contexto global; relações de join são definidas de forma estruturada no knowledge store, o que dá resultados mais consistentes.

> **Dica de prova:** Genie confunde termos, valores e joins: sinônimos, entity matching e join relationships no knowledge store.

R1.4. B) Um volume externo (CREATE EXTERNAL VOLUME) com LOCATION apontando para o caminho existente

Por quê: Volumes gerenciados guardam arquivos no local gerenciado do esquema e são apagados com o volume. Volumes externos apontam para um caminho existente coberto por uma external location, útil quando outros sistemas usam os mesmos arquivos.

✗ A) Incorreta. Montagens do DBFS são legado e não usam a governança do Unity Catalog.

✗ C) Incorreta. Tabelas são para dados tabulares; imagens são arquivos não tabulares, caso típico de volume.

✗ D) Incorreta. O volume gerenciado guarda os arquivos no armazenamento gerenciado do Unity Catalog, o que exigiria copiar e mudaria o caminho usado pelo PACS.

> **Dica de prova:** Arquivos já existentes e usados por outro sistema: volume externo.

02

CAPÍTULO 2 · 30-35% DA PROVA

Implantar e manter pipelines de dados e cargas de trabalho

Este é o outro domínio de maior peso (30-35%). Preciso dominar Lakeflow Jobs (tarefas, dependências, Run if, If/else, For each, task values, parâmetros, gatilhos, retries, timeout, fila, notificações, Repair run), saber quando usar notebook ou Lakeflow Spark Declarative Pipelines, e cuidar do ciclo de vida com Git folders, pull requests, testes e Declarative Automation Bundles (antes Databricks Asset Bundles). Também cai monitoramento e otimização: Spark UI (skew, spill, shuffle), cache, OPTIMIZE e VACUUM, tabelas de sistema de faturamento e Log Analytics com alertas do Azure Monitor. Onde mais erro: Run if x If/else, Repair run x Run now, retries x timeout x contínuo e os comandos de bundle.

Neste capítulo

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

Tópicos-chave

Dependências e Run if

Ordem se garante com dependências, nunca com horários: dimensões antes da fato, bronze antes de silver antes de gold; tarefas paralelas não dependem entre si e a junção depende de todas. Run if avalia o estado das dependências: All succeeded (padrão, pulada não conta), None failed (tolera puladas), All done (roda de qualquer jeito), At least one failed (compensação), At least one succeeded.

If/else condition e For each

If/else condition compara valores (task values, parâmetros de job) com ==, != e maior que e cria ramos true e false. For each executa a tarefa aninhada uma vez por item de uma lista (estática, de parâmetro ou de task value), com concorrência configurável. Job dentro de job: tarefa Run Job.

Task values, parâmetros e widgets

Valor de uma tarefa para outra: dbutils.jobs.taskValues.set e get; variáveis globais não passam, pois cada tarefa roda em processo próprio. Parâmetros de job valem para todas as tarefas e aceitam {{job.start_time.iso_date}} e {{job.run_id}} ({{start_date}} é obsoleta); parâmetro de tarefa fica restrito a ela. No notebook, dbutils.widgets.text('nome', 'padrao') define o padrão sobrescrito pelo orquestrador.

dbutils.notebook.run x %run

dbutils.notebook.run(caminho, timeout_em_segundos, {parametros}) executa outro notebook isolado, com tempo limite e retorno via dbutils.notebook.exit. %run inclui funções e variáveis no mesmo contexto, sem timeout e sem retorno. %pip não instala notebook.

Retries, timeout e idempotência

Por tarefa: número de novas tentativas, intervalo mínimo, timeout e retry on timeout. Só é seguro com tarefa idempotente: troque INSERT INTO por MERGE por chave. Se o código captura a exceção, registre e relance com raise; tarefa que não falha não aciona retry nem alerta.

Gatilhos

Cron do Lakeflow Jobs é Quartz: segundos primeiro e ? em dia do mês ou da semana, com fuso horário. File arrival monitora volumes ou external locations do Unity Catalog. Table update observa tabelas do Unity Catalog (Any ou All tables updated, Wait after last change). Contínuo mantém uma execução sempre ativa, reinicia com recuo exponencial e não aceita retry policy.

Controle de execução

Sem paralelismo e sem perder disparos: máximo de execuções simultâneas 1 com fila habilitada. Suspender job agendado: Pause no agendamento; job contínuo: cancelar reinicia, para parar use Pause. Execução errada em andamento: Cancel e Run now with different parameters.

Repair run

Reexecuta a tarefa com falha e suas dependentes dentro da mesma execução, permitindo alterar parâmetros, sem refazer o que já deu certo. Run now cria execução nova com todas as tarefas desde o início.

Notificações e alertas

Notificações de job usam e-mail, Teams, Slack e webhooks (não SMS), com opções para silenciar execuções puladas, canceladas ou até a última tentativa. Duration warning e Streaming backlog avisam sem parar; timeout encerra. Pipelines declarativos têm e-mail para falha de fluxo. Avisar quando o resultado de uma consulta cruza um limite: alerta SQL.

Notebook x Lakeflow Spark Declarative Pipelines

Pipeline declarativo (antes Delta Live Tables) quando a plataforma deve cuidar de dependências, incremental, novas tentativas e qualidade com streaming tables, materialized views e expectations. Notebook em job para API externa, laços e desvios procedurais. No job, use tarefa Pipeline e faça as tarefas seguintes dependerem dela.

Operação do pipeline declarativo

Modo produção reinicia a computação e repete a atualização em erros recuperáveis; desenvolvimento é para depurar, sem retries. Fonte com retenção curta em streaming table: pipelines.reset.allowed = false bloqueia full refresh. Recalcular uma tabela: full_refresh_selection em POST /api/2.0/pipelines/{pipeline_id}/updates. No editor: Run file ou Run table testa parte; Dry run valida sem gravar. Configuração por ambiente: chave em configuration lida com spark.conf.get.

Git folders

Uma Git folder por desenvolvedor, na pasta de usuário, com branch própria. Fluxo: pull, branch, commit e push, pull request e aprovação no provedor (GitHub, Azure DevOps), onde ficam as proteções de branch. Conflito: editar, Mark As Resolved e Continue Merge; Keep all current changes ou Take all incoming changes aceitam um lado inteiro; Abort desfaz. Merge traz a main sem reescrever histórico; Rebase exige push forçado; Reset descarta alterações.

Código de produção e segredos no Git

Job de produção roda o código aprovado: fonte Git na tarefa apontando para tag ou commit, ou implantação por bundle. Atualizar a Git folder de produção pelo CI/CD: PATCH /api/2.0/repos/{repo_id} com branch ou tag. Senha sai do código para escopo de segredo, e .databricks e arquivos locais de teste entram no .gitignore.

Declarative Automation Bundles

Um único databricks.yml na raiz; targets por ambiente com overrides de variables e resources; include junta resources/*.yml; artifacts constrói o wheel; run_as define quem executa e permissions quem vê ou gerencia. Variável no deploy: --var='nome=valor' ou BUNDLE_VAR_nome. mode: development prefixa [dev usuário], pausa agendas e deixa pipelines em development: true.

Comandos do bundle e CI/CD

validate verifica, deploy implanta, run executa um recurso, summary lista o que foi implantado com links, destroy desfaz e init cria projeto a partir de modelo. CI típico: validate e deploy -t prod; integração: deploy -t hml e run do job de testes. Autenticação no CI: entidade de serviço com workload identity federation (OIDC), sem PAT nem client secret guardado.

Testes

Teste de unidade: funções em módulos .py, pytest e DataFrames pequenos sintéticos, comparando com pyspark.testing.assertDataFrameEqual. Integração: bundle em homologação. Fluxo completo validado pelo negócio: ponta a ponta com UAT em homologação e dados representativos, nunca em produção.

Diagnóstico de desempenho

Poucas tarefas muito mais lentas que a mediana: skew (AQE com skew join ou salting). Spill em todas as tarefas: mais partições de shuffle ou mais memória. Milhares de tarefas minúsculas: deixe o AQE juntar partições. Tabela grande com dimensão pequena: broadcast. UDF Python lenta: funções nativas. OOM no driver: procure collect() e toPandas(). Lentidão intermitente com spot e autoscaling: log de eventos da computação.

Manutenção de tabelas e custo

Muitos arquivos pequenos: OPTIMIZE; arquivos antigos não referenciados: VACUUM (prévia com DRY RUN); predictive optimization faz OPTIMIZE, VACUUM e ANALYZE em gerenciadas (ALTER CATALOG ... ENABLE PREDICTIVE OPTIMIZATION, dono ou MANAGE). Custo: system.billing.usage com custom_tags e list_prices, junto com system.compute.clusters; serverless por projeto: serverless usage policies (antes budget policies). Log Analytics exige plano Premium e configuração de diagnóstico; falha de job: DatabricksJobs com ActionName runFailed, regra de alerta de logs e grupo de ações.

Macetes para memorizar

💡 Run if

Pulada pode, falha não: None failed. Roda de qualquer jeito: All done. Só em erro: At least one failed.

💡 Valor x estado

Valor decide no If/else; estado decide no Run if.

💡 Corrigir job

Repair conserta o que quebrou; Run now começa tudo de novo.

💡 Timeout x retry

Travou: timeout. Falhou: retry. Os dois na tarefa.

💡 Alerta de log no Azure Monitor

Frequência = quando roda; janela = quanto tempo olha; limite = quando dispara; grupo de ações = quem avisar.

💡 Bundle

include junta, artifacts constrói, targets separa, variables troca, run_as executa.

💡 OPTIMIZE x VACUUM

OPTIMIZE arruma a casa; VACUUM leva o lixo antigo.

💡 Notebook x pipeline

Fala com o mundo (API, laço, desvio) = notebook; tabela vira tabela = pipeline.

Tabela de memorização: não confunda

Quando é...	...e quando é

Run if Decidir pela situação das tarefas anteriores (sucesso, falha, pulada).	**If/else condition** Decidir pelo valor de um dado (task value ou parâmetro maior que 1000).
Repair run Refazer só a tarefa com falha e dependentes, com parâmetro corrigido.	**Run now** Começar execução nova do zero.
dbutils.notebook.run Executar outro notebook isolado com parâmetros, timeout em segundos e retorno.	**%run** Reaproveitar funções e variáveis na mesma sessão.
Novas tentativas e timeout Tarefa agendada com falha transitória ou que trava.	**Gatilho contínuo** Streaming 24x7 que deve reiniciar sozinho (recuo exponencial, sem retry policy).
Gatilho file arrival Arquivos chegando em volume ou external location em horário incerto.	**Gatilho table update** Rodar quando tabelas do Unity Catalog forem atualizadas.
Skew Poucas tarefas muito mais lentas: AQE skew join ou salting.	**Spill generalizado** Todas as tarefas derramando em disco: mais partições ou mais memória.
Modo desenvolvimento do pipeline Depurar, sem retries.	**Modo produção do pipeline** Reiniciar a computação e repetir a atualização em erros recuperáveis.

Números que caem

Item	Valor
Unidade do timeout de dbutils.notebook.run	**segundos (600 = 10 minutos)**
Retenção padrão do VACUUM	**7 dias (abaixo disso exige desligar a verificação de segurança)**
Exemplo de cron Quartz dias úteis às 6h30	**'0 30 6 ? * MON-FRI' com fuso America/Sao_Paulo**
Endpoint para acompanhar uma execução disparada pela API	**GET /api/2.2/jobs/runs/get com o run_id**
Endpoint para sobrescrever a definição inteira de um job	**POST /api/2.2/jobs/reset (jobs/update é alteração parcial)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Run if All succeeded nas duas tarefas depois da contagem para escolher o ramo.
Como não cair: Run if olha estado, não valor; as duas rodariam. Ramificar por valor é If/else condition.

⚠ **Alerta de prova:** All succeeded para rodar mesmo com dependência pulada.
Como não cair: Pulada não conta como sucesso; tolerar pulada e barrar falha é None failed.

⚠ **Alerta de prova:** Aumentar as novas tentativas para corrigir parâmetro errado.
Como não cair: A tentativa repete o mesmo parâmetro; use Repair run informando o valor correto.

⚠ **Alerta de prova:** Novas tentativas recuperam execução pulada por limite de concorrência.
Como não cair: Pulada não é falha; mantenha concorrência 1 e habilite a fila.

⚠ **Alerta de prova:** dbutils.notebook.run com timeout 10 para esperar dez minutos.
Como não cair: O timeout é em segundos: dez minutos são 600; e o retorno vem de dbutils.notebook.exit, não de print.

⚠ **Alerta de prova:** Passar o valor entre tarefas por variável global ou spark.conf.
Como não cair: Cada tarefa roda em processo próprio; use dbutils.jobs.taskValues (entre tarefas) ou widgets (notebook chamado).

⚠ **Alerta de prova:** Gatilho contínuo para recuperar tarefa agendada que travou.
Como não cair: Contínuo é para cargas sempre ativas e não aceita retry policy; use timeout + retries.

⚠ **Alerta de prova:** Gatilho file arrival monitorando um servidor SFTP ou table update sobre volume.
Como não cair: File arrival monitora volumes e external locations do Unity Catalog; table update observa tabelas.

⚠ **Alerta de prova:** Cron '0 30 6 * * MON-FRI' com dia do mês e da semana preenchidos.
Como não cair: No Quartz um dos dois precisa ser ?: '0 30 6 ? * MON-FRI', com o fuso correto.

⚠ **Alerta de prova:** VACUUM RETAIN 0 HOURS junta os arquivos pequenos.
Como não cair: VACUUM não compacta e reter zero horas pode quebrar leitores e a time travel; compactar é OPTIMIZE.

⚠ **Alerta de prova:** OPTIMIZE ZORDER BY em tabela com liquid clustering.
Como não cair: ZORDER não é compatível; rode OPTIMIZE simples, que é incremental; OPTIMIZE FULL só após trocar chaves.

⚠ **Alerta de prova:** Passar --var só no bundle run para mudar o catálogo.
Como não cair: A variável é resolvida no deploy; passe --var no deploy (ou override no target).

⚠ **Alerta de prova:** mode: production troca o catálogo de destino automaticamente.
Como não cair: mode só aplica comportamentos de implantação; valores por ambiente vêm de variables no target.

⚠ **Alerta de prova:** Filtrar categorias indesejadas na consulta KQL para reduzir o custo do Log Analytics.
Como não cair: A cobrança é na ingestão; selecione só as categorias necessárias na configuração de diagnóstico.

Como cai na prova

📝 **Como cai:** Job com várias tarefas e uma condição de execução ou ramificação.
Como resolver: Pergunte se a decisão é por valor (If/else) ou por estado (Run if); se for estado, escolha entre All succeeded, None failed, All done e At least one failed pela frase do requisito (pulada pode? falha pode?).

📝 **Como cai:** Job falhou ou precisa se recuperar sozinho.
Como resolver: Parâmetro errado: Repair run com parâmetro novo. Falha transitória: retries com intervalo. Trava: timeout com retry on timeout. Exceção engolida: relançar. Disparo pulado: fila.

📝 **Como cai:** Requisitos de implantação com bundles em dev, hml e prod.
Como resolver: Diferença por ambiente vai em targets (variables e resources); identidade em run_as; acesso em permissions; comando pela ação: validate, deploy, run, summary ou destroy; CI com entidade de serviço e OIDC.

📝 **Como cai:** Lentidão com sintomas do Spark UI ou da computação.
Como resolver: Poucas tarefas lentas = skew; spill em todas = partições ou memória; tarefas minúsculas = AQE; join com dimensão pequena = broadcast; intermitente com spot = log de eventos da computação; CPU baixa e folga = reduzir o cluster.

📝 **Como cai:** Monitorar falhas e custo fora do Databricks.
Como resolver: Plano Premium + configuração de diagnóstico com só as categorias necessárias; consulta KQL em DatabricksJobs (runFailed); regra de alerta de logs com frequência, janela e limite; grupo de ações para e-mail, SMS e webhook.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
ramificar pelo valor de um task value	**If/else condition**
rodar mesmo com tarefa pulada, mas não com falha	**Run if None failed**
compensação só em erro	**Run if At least one failed**
mesma lógica para cada item de uma lista que muda	**For each com concorrência limitada**
falhou no meio de um job longo	**Repair run**
sem perder disparos e sem paralelismo	**Concorrência 1 + fila**
arquivos em horário imprevisível	**Gatilho file arrival**
esperar duas tabelas e o fim da rajada	**Table update com All tables updated + Wait after last change**
dependências, incremental e qualidade gerenciados	**Lakeflow Spark Declarative Pipelines**
CI/CD sem segredo de longa duração	**Entidade de serviço + workload identity federation (OIDC)**
mesmo job, valores diferentes por ambiente	**variables com override em targets**
poucas tarefas muito mais lentas que a mediana	**Skew: AQE skew join ou salting**
custo por projeto ou área	**custom_tags + system.billing.usage + list_prices**
SMS e webhook quando alerta do Azure Monitor dispara	**Grupo de ações**

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

- Run if = estado das tarefas; If/else condition = valor de um dado.
- Repair run refaz só o que falhou e aceita parâmetros novos.
- Retry só é seguro com tarefa idempotente; relance a exceção.
- Job contínuo não aceita retry policy; para parar, Pause.
- Produção roda código do Git (tag ou commit) ou de bundle, com entidade de serviço.

Essenciais para a prova (checklist)

- ☐ Sei escolher a condição Run if certa e quando usar If/else condition e For each.
- ☐ Sei passar valores com task values, parâmetros de job, referências dinâmicas e widgets.
- ☐ Sei diferenciar dbutils.notebook.run de %run.
- ☐ Sei configurar retries, timeout, fila, concorrência e notificações com mute.

☐ Sei escolher entre cron Quartz, file arrival, table update e contínuo.

☐ Sei usar Repair run, Run now with different parameters e Pause.

☐ Sei quando usar notebook ou pipeline declarativo e operar modo desenvolvimento x produção.

☐ Sei o fluxo de Git folders, resolução de conflitos e pull request no provedor.

☐ Sei configurar targets, variables, run_as, permissions e os comandos validate, deploy, run, summary e destroy.

☐ Sei diagnosticar skew, spill, shuffle e OOM no driver e monitorar custo e falhas com tabelas de sistema e Azure Monitor.

Questões de revisão do capítulo

R2.1 · Implantar e manter pipelines de dados e cargas de trabalho

A equipe da Agência de Viagens Mundo Afora usa GitHub Actions para rodar databricks bundle deploy -t prod a cada merge na main. A segurança não aceita segredos de longa duração (PAT ou client secret) guardados no GitHub e exige que a implantação seja feita por uma identidade de automação, não por um usuário. Como você deve configurar a autenticação?

A. Client secret OAuth da entidade de serviço guardado como secret do GitHub
B. PAT do engenheiro líder guardado como secret do repositório no GitHub
C. Login interativo com databricks auth login executado no runner do pipeline a cada deploy
D. Entidade de serviço com federação de identidade de carga de trabalho (OIDC)

R2.2 · Implantar e manter pipelines de dados e cargas de trabalho

Em um cluster all-purpose da Agência de Marketing Ideia Viva, alguém instalou como biblioteca de cluster a versão 2.0 de um pacote interno que conflita com a versão 1.8 usada pelos notebooks da equipe, e os imports começaram a falhar. O administrador removeu a biblioteca 2.0 da lista de bibliotecas do cluster, mas os erros continuam. O que falta fazer?

A. Reinstalar a versão 1.8 com %pip em uma célula nova
B. Limpar o estado de cada notebook com Clear state
C. Desanexar e reanexar os notebooks da equipe ao cluster
D. Reiniciar o cluster para recarregar o ambiente

R2.3 · Implantar e manter pipelines de dados e cargas de trabalho

Uma tarefa de job de uma empresa de logística chama uma API de rastreio que às vezes fica indisponível por alguns minutos, e a chamada às vezes trava sem responder. A tarefa é idempotente. O que você deve configurar para que ela se recupere sozinha?

A. Novas tentativas na tarefa com intervalo mínimo entre elas, um timeout e a opção de tentar de novo no timeout
B. Execução contínua do job, que reinicia a tarefa em segundos depois de cada falha da API
C. Notificação de falha por e-mail para que o plantão reexecute a tarefa quando a API voltar
D. Run if All done nas tarefas seguintes para que rodem mesmo depois da falha da chamada à API

R2.4 · Implantar e manter pipelines de dados e cargas de trabalho

Um atacadista de Goiânia carrega a fato_vendas com chaves substitutas (surrogate keys) obtidas por lookup nas dimensões de cliente e produto. Em algumas noites, vendas de clientes recém-cadastrados chegam ao gold com chave de cliente nula. As cargas das dimensões e da fato rodam como tarefas paralelas do mesmo job. O que você deve fazer?

A. Fazer a tarefa da fato depender das tarefas que carregam as dimensões de cliente e de produto
B. Aumentar o número de workers do cluster para as tarefas paralelas terminarem antes
C. Executar a carga da fato primeiro e atualizar as dimensões em seguida na mesma execução do job
D. Trocar a junção da fato com as dimensões por um full outer join para preservar as chaves nulas

Respostas comentadas

R2.1. D) Entidade de serviço com federação de identidade de carga de trabalho (OIDC)

Por quê: Em CI/CD, a implantação de bundles deve usar uma entidade de serviço. A federação de tokens (workload identity federation) com o provedor de CI elimina segredos armazenados, trocando o token do provedor por um token OAuth de curta duração.

✗ A) Incorreta. Usar a entidade de serviço é o caminho, mas guardar o client secret no GitHub mantém um segredo de longa duração.

✗ B) Incorreta. Um PAT de usuário é um segredo de longa duração ligado a uma pessoa, justamente o que foi proibido.

✗ C) Incorreta. Login interativo exige uma pessoa e não funciona em um runner automatizado.

> **Dica de prova:** CI/CD sem segredo guardado: entidade de serviço + workload identity federation (OIDC).

R2.2. D) Reiniciar o cluster para recarregar o ambiente

Por quê: Mudanças em bibliotecas de cluster, configurações Spark e variáveis de ambiente exigem reinício do cluster para valer. Reiniciar também resolve estados corrompidos ou conflitos de pacotes carregados em memória.

✗ A) Incorreta. Instalar a 1.8 com escopo de notebook pode até funcionar na sessão, mas a remoção definitiva da 2.0 depende do reinício.

✗ B) Incorreta. Clear state limpa as variáveis da sessão do notebook, mas não desinstala bibliotecas do cluster.

✗ C) Incorreta. Reanexar cria uma sessão nova, mas a biblioteca 2.0 continua instalada no ambiente do cluster até o reinício.

> **Dica de prova:** Removeu a biblioteca do cluster e o erro continua: reinicie o cluster.

R2.3. A) Novas tentativas na tarefa com intervalo mínimo entre elas, um timeout e a opção de tentar de novo no timeout

Por quê: Configure por tarefa: número de novas tentativas, intervalo mínimo entre elas, tempo limite e repetição em timeout. Isso só é seguro quando a tarefa é idempotente.

✗ B) Incorreta. O modo contínuo é para jobs que devem estar sempre em execução, não para recuperar uma tarefa agendada.

✗ C) Incorreta. Uma notificação depende de ação humana, o oposto de recuperação automática.

✗ D) Incorreta. Run if All done só deixa as tarefas seguintes rodarem; a tarefa com falha não é repetida.

> **Dica de prova:** Falha transitória: novas tentativas com intervalo + timeout com retry on timeout.

R2.4. A) Fazer a tarefa da fato depender das tarefas que carregam as dimensões de cliente e de produto

Por quê: Em um modelo dimensional, as dimensões são carregadas antes das fatos, porque a fato precisa resolver as chaves substitutas por lookup. No Lakeflow Jobs, essa ordem é expressa por dependências entre tarefas.

✗ B) Incorreta. Mais workers não garantem ordem; a condição de corrida entre as tarefas paralelas continuaria.

✗ C) Incorreta. Essa é a ordem invertida: a fato faria o lookup antes de as dimensões receberem os registros novos.

✗ D) Incorreta. O full outer join não resolve a chave ausente; só mantém as linhas com chave nula no gold.

> **Dica de prova:** Dimensões primeiro, fatos depois: modele com dependências entre as tarefas.

03

CAPÍTULO 3 · 30-35% DA PROVA

Preparar e processar dados

Este é um dos domínios de maior peso (30-35%) e mistura modelagem, ingestão e transformação. Preciso escolher formato e tipo de tabela, grão e SCD, particionamento ou liquid clustering, e a ferramenta de ingestão certa (Lakeflow Connect, Auto Loader, COPY INTO, Kafka do Event Hubs, Azure Data Factory, notebook). Também caem limpeza com PySpark e SQL (nulos, duplicados, tipos, joins, PIVOT, janelas), MERGE e evolução de esquema, Structured Streaming com watermark e checkpoint, Change Data Feed, AUTO CDC e qualidade com constraints e expectations. Onde mais erro: o modo de schemaEvolutionMode, expect x expect_or_drop x expect_or_fail, INSERT OVERWRITE x CREATE OR REPLACE e quando particionar.

Neste capítulo

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

Tópicos-chave

Delta e tipo de tabela

Delta Lake é Parquet colunar com log de transações: ACID, time travel e MERGE no mesmo requisito. Sem exigência de caminho externo, use tabela gerenciada do Unity Catalog com predictive optimization; caminho fixo usado por outro sistema ou dados que sobrevivem ao DROP pedem tabela externa sobre external location. Motor externo que fala Iceberg: tabela Iceberg gerenciada (USING iceberg) via Iceberg REST Catalog.

Lakeflow Connect

Origem SaaS com conector gerenciado (ServiceNow, Salesforce) vai para Lakeflow Connect antes de código próprio. SQL Server exige change tracking ou CDC na origem, gateway e pipeline de ingestão. Histórico de mudanças: history tracking com SCD_TYPE_2; uma tabela inconsistente: full refresh só dela, sem recriar gateway.

Outras rotas de ingestão

API REST sem conector gerenciado: notebook Python que pagina e grava em Delta, agendado em job. Origem on-premises com Azure Data Factory: self-hosted integration runtime (serverless não tem rota para a rede local). Carga SQL repetível que pula arquivos já lidos: COPY INTO. Criar e carregar em uma instrução: CTAS com read_files.

Auto Loader

Ingestão incremental de arquivos com checkpoint; com trigger(availableNow=True) roda agendado sem computação ligada. schemaHints corrige o tipo de colunas inferidas; maxFilesPerTrigger ou maxBytesPerTrigger limitam o micro-lote; muitos arquivos e listagem lenta pedem file events na external location com cloudFiles.useManagedFileEvents. Dentro do pipeline declarativo, schemaLocation e checkpoint são automáticos.

schemaEvolutionMode e _rescued_data

addNewColumns (padrão quando não se informa esquema): o stream para, atualiza o esquema e segue após o reinício. rescue: nunca evolui o esquema, não para e guarda o novo em _rescued_data. failOnNewColumns: falha até haver revisão. none (padrão quando o esquema é informado): ignora colunas novas, que só são resgatadas com a opção rescuedDataColumn. Valores fora do esquema vão para _rescued_data; _metadata traz caminho, nome e tamanho do arquivo.

Event Hubs pela API Kafka

Fonte kafka do Structured Streaming no namespace servicebus.windows.net, porta 9093, SASL_SSL, PLAIN e usuário $ConnectionString com a connection string vinda de escopo de segredo. value chega binário: cast('string') + from_json. startingOffsets = 'earliest' lê o que já está retido. Auto Loader não lê Event Hubs.

Structured Streaming

Cada consulta tem checkpointLocation exclusivo para retomar sem perder nem duplicar. Deduplicar sem estado infinito: withWatermark + dropDuplicates; join entre streams: watermark nos dois lados + condição de tempo; enriquecer com dimensão Delta: stream-static join. Upsert em streaming: foreachBatch + MERGE. Fonte Delta com DELETE quebrando o stream: skipChangeCommits.

Change Data Feed

Habilite delta.enableChangeDataFeed = true e leia com table_changes ou readChangeFeed a partir da última versão processada; cada linha traz o tipo de mudança. update_preimage é o valor antigo e update_postimage o novo. Ler os JSON de _delta_log não é interface suportada.

AUTO CDC (antes APPLY CHANGES)

Substitui MERGE manual de CDC em pipeline declarativo com KEYS, SEQUENCE BY e APPLY AS DELETE WHEN. SEQUENCE BY usa o tempo da origem, não o da ingestão. STORED AS SCD TYPE 2 gera __START_AT e __END_AT.

SCD, grão e tabelas de histórico

Só o valor atual importa: SCD tipo 1 (sobrescrever); valor da época do fato: SCD tipo 2 com nova linha e datas de vigência; tipo 0 congela o primeiro cadastro. Estado em uma data: data entre valid_from e valid_to, tratando fim nulo com coalesce. Histórico de anos: tabela de histórico, não time travel. Grão: o mais fino pedido pelo negócio; a gold para painel agrega no menor grão exibido.

Particionamento x liquid clustering

Partição só compensa acima de 1 TB e em coluna de baixa cardinalidade; abaixo disso, liquid clustering nas colunas de filtro (até quatro, sem redundância). Partição diária a partir de timestamp: coluna gerada com CAST AS DATE. Trocar chaves: ALTER TABLE ... CLUSTER BY; filtro imprevisível em tabela gerenciada: CLUSTER BY AUTO. Liquid não combina com partição nem com ZORDER.

Gravação: CTAS, OVERWRITE e append

Recriar tabela sem perder histórico nem GRANT: CREATE OR REPLACE TABLE, não DROP + CREATE. Trocar dados sem deixar o esquema mudar: INSERT OVERWRITE; aceitar novo esquema: CREATE OR REPLACE AS SELECT. Reprocessar só um dia: overwrite com replaceWhere. Coluna nova aceita: append com mergeSchema = true.

MERGE

Várias linhas de origem para a mesma chave dão erro: deduplique a origem antes. Apagar no destino o que sumiu da origem: WHEN NOT MATCHED BY SOURCE THEN DELETE. Coluna nova no MERGE: WITH SCHEMA EVOLUTION; tipo maior: type widening.

Constraints e expectations

No Delta só NOT NULL e CHECK são impostas; PK e FK são informativas (RELY só orienta o otimizador). ADD CONSTRAINT valida as linhas existentes e falha se houver violação: corrija ou remova e repita. Expectations: expect só mede, expect_or_drop descarta e segue, expect_or_fail para; métricas por regra no log de eventos do pipeline; quarentena = expect_or_drop + tabela com a regra invertida.

Tipos de dados

Valor monetário: DECIMAL, nunca DOUBLE. Horário de parede sem fuso: TIMESTAMP_NTZ. JSON com esquema volátil: VARIANT com parse_json. Chaves dinâmicas: MAP; campos fixos: STRUCT; listas: ARRAY. Modo ANSI (padrão no serverless) transforma cast inválido em erro CAST_INVALID_INPUT; try_cast e try_to_number devolvem NULL.

Profiling, nulos e duplicados

summary() traz percentis, describe() só o básico, dbutils.data.summarize faz perfil rápido; em tabela enorme, approx_count_distinct e percentile_approx. Nulos: na.drop(subset), fillna, coalesce entre colunas e nullif(trim(coluna), '') para vazios. Último registro por chave: row_number com ORDER BY data DESC e filtro = 1 (QUALIFY); dropDuplicates guarda uma linha qualquer.

Joins, conjuntos e reformatação

Quem não aparece no outro conjunto: left_anti; conciliação nos dois sentidos: FULL OUTER JOIN; comuns: INTERSECT; só na primeira: EXCEPT; colunas em outra ordem ou faltando: unionByName(allowMissingColumns=True). Linhas viram colunas com PIVOT e colunas viram linhas com UNPIVOT; explode gera uma linha por elemento de array.

Agregações e janelas

Filtro sobre linha vai no WHERE e sobre agregado no HAVING. Subtotais hierárquicos e total geral: ROLLUP; todas as combinações: CUBE. Acumulado: sum() OVER (PARTITION BY grupo ORDER BY data); sem ORDER BY vira total do grupo. Contagem condicional: count_if ou count(*) FILTER (WHERE ...).

Macetes para memorizar

schemaEvolutionMode

add = cai e volta; rescue = guarda no bolso; failOn = para tudo; none = finge que não viu.

Expectations

expect avisa, drop joga fora, fail para a fábrica.

Event Hubs via Kafka

9093, SASL_SSL, PLAIN e $ConnectionString: porta, protocolo, mecanismo, usuário.

PIVOT e UNPIVOT

PIVOT deita (linhas viram colunas); UNPIVOT levanta (colunas viram linhas).

Particionar

Partição só para gigante (acima de 1 TB) e de poucos valores; o resto é liquid.

ROLLUP x CUBE

ROLLUP desce a escada (hierarquia); CUBE abre todas as portas (todas as combinações).

Overwrite

INSERT OVERWRITE protege o esquema; CREATE OR REPLACE aceita o esquema novo.

Tabela de memorização: não confunda

Quando é...	...e quando é
Streaming table Origem só de acréscimo processada incrementalmente (eventos limpos).	**Materialized view** Agregado que precisa refletir correções e mudanças da origem.
INSERT OVERWRITE Trocar os dados e falhar se o esquema da consulta mudar.	**CREATE OR REPLACE TABLE AS SELECT** Aceitar o esquema novo da consulta, mantendo histórico e GRANTs.
SCD tipo 1 Só o valor atual interessa; sobrescreve.	**SCD tipo 2** Precisa do valor vigente na época do fato; nova linha com vigência.
Particionamento Tabela acima de 1 TB com filtro em coluna de baixa cardinalidade (ou tabela particionada que não migra, com ZORDER fora da coluna de partição).	**Liquid clustering** Tabelas novas, abaixo de 1 TB, alta cardinalidade ou padrão de filtro que muda.
Marca d'água por updated_at Tabela grande com coluna de atualização confiável e sem exclusões físicas a capturar.	**CDC (change tracking)** Precisa de inclusões, alterações e exclusões com baixa latência sem varrer tudo.
expect_or_drop Descartar o inválido e seguir (com quarentena para não perder o registro).	**expect_or_fail** Uma única violação já é motivo para parar a atualização.
trigger(availableNow=True) em job agendado Incremental agendado (de hora em hora, a cada 6 horas): processa o acumulado e encerra, sem computação ligada entre as execuções.	**Job com gatilho contínuo** Stream com latência de segundos, 24 horas por dia, reiniciado automaticamente. Não confunda com trigger(continuous=...), que é para latência de milissegundos.

Números que caem

Item	Valor
Tamanho mínimo para o particionamento compensar	**acima de 1 TB**
Chaves de liquid clustering recomendadas	**até quatro**
Porta Kafka do Event Hubs	**9093 (5672 é AMQP)**

Alertas de prova (pegadinhas)

> ⚠ **Alerta de prova:** Auto Loader (cloudFiles) apontando para o namespace do Event Hubs.
> **Como não cair:** Auto Loader lê arquivos; Event Hubs se lê com readStream format('kafka') na porta 9093.

⚠ **Alerta de prova:** Usar a porta 5672 ou option('valueDeserializer', 'json') na leitura do Event Hubs.
Como não cair: 5672 é AMQP; Kafka usa 9093. A fonte Kafka não tem desserializador: cast('string') + from_json.

⚠ **Alerta de prova:** Marca d'água por data de atualização também captura exclusões físicas.
Como não cair: Linha apagada some da origem e não aparece na consulta incremental; exclusões pedem CDC.

⚠ **Alerta de prova:** SEQUENCE BY ts_ingestao no AUTO CDC.
Como não cair: A ordem de chegada repete a desordem; sequencie pelo tempo da origem.

⚠ **Alerta de prova:** Particionar por id_cliente ou id_paciente para cada um ter sua pasta.
Como não cair: Alta cardinalidade gera milhões de arquivos pequenos; use liquid clustering.

⚠ **Alerta de prova:** ALTER TABLE ... CLUSTER BY mantendo o particionamento atual.
Como não cair: Liquid clustering não pode ser combinado com partição; recrie sem partição ou use ZORDER na tabela particionada.

⚠ **Alerta de prova:** CHECK constraint só vale para linhas novas.
Como não cair: ADD CONSTRAINT valida as linhas existentes; corrija ou remova as violadoras antes.

⚠ **Alerta de prova:** RELY na PRIMARY KEY passa a validar duplicados na escrita.
Como não cair: RELY só informa o otimizador; deduplique antes e confira com GROUP BY ... HAVING count(*) > 1.

⚠ **Alerta de prova:** @dp.expect para tirar os registros inválidos da silver.
Como não cair: expect só registra métrica e grava o inválido; descartar é expect_or_drop.

⚠ **Alerta de prova:** mergeSchema resolve erro de várias linhas de origem no MERGE.
Como não cair: mergeSchema trata esquema; o erro é origem duplicada, que se deduplica antes.

⚠ **Alerta de prova:** mode('append') com o dia corrigido para reprocessar a data.
Como não cair: Append duplica as vendas do dia; use overwrite com replaceWhere.

⚠ **Alerta de prova:** Modo complete com trigger de 10 minutos grava só janelas fechadas.
Como não cair: Complete regrava o resultado inteiro; emitir cada janela uma vez é append com watermark.

⚠ **Alerta de prova:** count(status = 'CANCELADO') conta só os cancelados.
Como não cair: count conta não nulos, e a comparação dá true ou false; use count_if ou FILTER (WHERE ...).

⚠ **Alerta de prova:** dropDuplicates(['id_cliente']) mantém o registro mais recente.
Como não cair: Guarda uma linha qualquer; use row_number com ORDER BY data DESC e filtro = 1.

Como cai na prova

📝 **Como cai:** Fonte X com requisito de latência, exclusões e esforço mínimo; alternativas com quatro ferramentas de ingestão.
Como resolver: Conector gerenciado existe? Lakeflow Connect. Arquivos? Auto Loader ou COPY INTO. Stream de eventos? Kafka do Event Hubs. On-premises com ADF? Self-hosted IR. API sem conector? Notebook em job. Exclusões? CDC.

📝 **Como cai:** Coluna nova chegando nos arquivos ou no MERGE.
Como resolver: Auto Loader: escolha o schemaEvolutionMode pelo verbo do requisito (parar, guardar, seguir). Batch append: mergeSchema. MERGE: WITH SCHEMA EVOLUTION. Tipo maior: type widening.

📝 **Como cai:** Stream com estado crescendo, duplicados ou join entre streams.
Como resolver: Aplique withWatermark na coluna de tempo do evento; dropDuplicates com a chave e o tempo; em join entre streams, watermark nos dois lados e condição de tempo.

📝 **Como cai:** Tabela lenta por layout: partições demais ou arquivos minúsculos.
Como resolver: Confira tamanho (acima de 1 TB?) e cardinalidade; alta cardinalidade ou tabela menor: liquid clustering pelas colunas de filtro; trocar chaves com ALTER TABLE ... CLUSTER BY.

📝 **Como cai:** Escolher a função SQL ou PySpark que entrega exatamente o resultado.
Como resolver: Leia o resultado esperado: acumulado precisa de ORDER BY; contar condição precisa de count_if; não aparece no outro = left_anti; ambos os lados = FULL OUTER; vazio vira NULL com nullif(trim()).

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
ACID, versão anterior e upsert	**Delta Lake**
origem SaaS com conector gerenciado	**Lakeflow Connect**

servidor on-premises com Azure Data Factory	**Self-hosted integration runtime**
arquivos chegando, incremental, sem cluster ligado	**Auto Loader + availableNow + checkpoint**
carga SQL repetível que não recarrega arquivos	**COPY INTO**
coluna nova deve parar a carga para revisão	**schemaEvolutionMode failOnNewColumns**
valor fora do esquema sem perder a linha	**_rescued_data**
CDC fora de ordem em pipeline declarativo	**AUTO CDC com KEYS e SEQUENCE BY (tempo da origem)**
linhas alteradas com antes e depois	**Change Data Feed**
estado crescendo e dados atrasados	**withWatermark na coluna de tempo do evento**
upsert em streaming	**foreachBatch + MERGE**
reprocessar só um dia	**overwrite com replaceWhere**
valor monetário	**DECIMAL**
CAST_INVALID_INPUT onde antes vinha NULL	**Modo ANSI; usar try_cast**

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

- Só NOT NULL e CHECK são impostas; PK e FK são informativas.
- Liquid clustering não combina com partição nem com ZORDER.
- Auto Loader lê arquivos; Event Hubs se lê pela fonte Kafka.
- SEQUENCE BY usa o tempo da origem.
- Cada consulta de streaming tem seu próprio checkpoint.

Essenciais para a prova (checklist)

- ☐ Sei escolher entre Lakeflow Connect, Auto Loader, COPY INTO, Kafka do Event Hubs, ADF e notebook.
- ☐ Sei os quatro modos de schemaEvolutionMode e para que serve _rescued_data.
- ☐ Sei configurar a leitura do Event Hubs pela fonte Kafka com segredo.
- ☐ Sei usar watermark, checkpoint, foreachBatch e stream-static join.
- ☐ Sei habilitar e ler o Change Data Feed e diferenciar preimage e postimage.
- ☐ Sei escrever AUTO CDC com KEYS, SEQUENCE BY e SCD TYPE 2.
- ☐ Sei decidir entre SCD 1 e 2, o grão da tabela e quando usar tabela de histórico.
- ☐ Sei decidir entre partição e liquid clustering e escolher as chaves.
- ☐ Sei usar MERGE com NOT MATCHED BY SOURCE, WITH SCHEMA EVOLUTION e origem deduplicada.
- ☐ Sei tratar nulos, vazios, duplicados, tipos inválidos, PIVOT, ROLLUP e janelas acumuladas.

Questões de revisão do capítulo

R3.1 · Preparar e processar dados

A Escola de Idiomas Fala Mais quer garantir que a coluna email da tabela Delta alunos não receba nulos. Ao executar ALTER TABLE alunos ALTER COLUMN email SET NOT NULL, o comando falhou porque 312 linhas antigas já têm email nulo. O que você deve fazer para aplicar a restrição?

A. Corrigir ou remover as 312 linhas e repetir o ALTER TABLE
B. Repetir o comando com NOT ENFORCED para aceitar as linhas antigas
C. Criar uma CHECK constraint (email IS NOT NULL) que vale para novas linhas
D. Recriar a tabela com mergeSchema para aplicar a restrição sem validar

R3.2 · Preparar e processar dados

Antes de modelar a tabela de fretes, a engenheira de dados da Transportadora Rota Certa quer, com PySpark, ver para as colunas peso_kg e valor_frete a contagem, a média, o desvio padrão, o mínimo, o máximo e também os percentis 25%, 50% e 75%, para avaliar a distribuição e possíveis outliers. Qual método você deve usar no DataFrame?

A. df.summary()
B. df.describe()
C. df.printSchema()
D. df.count()

R3.3 · Preparar e processar dados

Toda noite, a Rede de Academias Movimento recria a tabela silver.planos_ativos a partir de uma consulta sobre a bronze. Hoje o processo executa DROP TABLE seguido de CREATE TABLE ... AS SELECT, e a governança reclama que, a cada noite, se perdem as permissões concedidas e o histórico de versões, além de consultas concorrentes falharem durante a troca. O que você deve usar no lugar?

A. CREATE OR REPLACE TABLE silver.planos_ativos AS SELECT ... FROM bronze
B. COPY INTO silver.planos_ativos a partir dos arquivos da camada bronze
C. DROP TABLE IF EXISTS e, em seguida, CREATE TABLE ... AS SELECT
D. CREATE TABLE planos_ativos_novo AS SELECT ... e depois renomear a tabela

R3.4 · Preparar e processar dados

A tabela silver pedidos da Loja Virtual Casa e Decoração tem a coluna itens do tipo ARRAY de STRUCT com sku, quantidade e preco. O time de BI quer uma tabela gold larga, com uma linha por item de pedido e já com os atributos do cliente e do produto, para evitar joins nos dashboards. Qual abordagem você deve usar?

A. PIVOT por sku para criar uma coluna por produto em cada linha de pedido
B. explode(itens) para gerar uma linha por item e join com as dimensões de cliente e produto
C. Manter o ARRAY e criar uma view com size(itens) para o dashboard contar itens
D. flatten(itens) para juntar os arrays aninhados e manter uma linha por pedido na tabela gold

Respostas comentadas

R3.1. A) Corrigir ou remover as 312 linhas e repetir o ALTER TABLE

Por quê: No Delta Lake, NOT NULL e CHECK são restrições impostas: ao adicioná-las, as linhas existentes são verificadas, e depois toda escrita que violar a regra falha. Limpe os dados antes de aplicar a restrição.

✗ B) Incorreta. NOT NULL é uma restrição imposta no Delta e não tem variante informativa; restrições não impostas são as de chave primária e estrangeira.
✗ C) Incorreta. Uma CHECK constraint também valida as linhas existentes ao ser adicionada e falharia pelo mesmo motivo.
✗ D) Incorreta. mergeSchema trata a evolução de colunas na escrita; não aplica restrições nem dispensa a validação.

> **Dica de prova:** ALTER para NOT NULL ou CHECK falhou por dado antigo? Corrija ou remova as linhas violadoras e repita.

R3.2. A) df.summary()

Por quê: Para profiling rápido em PySpark, describe() dá estatísticas básicas e summary() acrescenta percentis aproximados. Em notebooks, dbutils.data.summarize e o perfil de dados dos resultados oferecem visões semelhantes, com gráficos.

✗ B) Incorreta. describe() traz count, mean, stddev, min e max, sem os percentis.
✗ C) Incorreta. printSchema() mostra os nomes e tipos das colunas, sem estatísticas.
✗ D) Incorreta. count() devolve o número de linhas, sem estatísticas de distribuição.

> **Dica de prova:** Percentis no profiling do DataFrame: summary(); só estatísticas básicas: describe().

R3.3. A) CREATE OR REPLACE TABLE silver.planos_ativos AS SELECT ... FROM bronze

Por quê: A Databricks recomenda CREATE OR REPLACE TABLE em vez de DROP seguido de CREATE: a substituição é atômica, preserva o histórico de versões e os privilégios concedidos, filtros de linha e máscaras, e não quebra transações concorrentes.

✗ B) Incorreta. COPY INTO acrescenta arquivos novos de forma idempotente; não recria a tabela a partir de uma consulta nem substitui o conteúdo.

✗ C) Incorreta. Continua com DROP: perde histórico e permissões e deixa a tabela inexistente por alguns instantes.

✗ D) Incorreta. Criar outra tabela e renomear perde o histórico da original e exige conceder as permissões de novo; também não é uma troca atômica para os leitores.

> **Dica de prova:** Recriar tabela sem perder histórico nem GRANT: CREATE OR REPLACE TABLE, não DROP + CREATE.

R3.4. B) explode(itens) para gerar uma linha por item e join com as dimensões de cliente e produto

Por quê: Para desnormalizar dados aninhados em uma tabela gold, use explode (ou inline para arrays de struct) para chegar ao grão desejado e joins com as dimensões para trazer os atributos descritivos.

✗ A) Incorreta. Pivotar por sku cria uma coluna por produto, com milhares de colunas esparsas, e não traz os atributos das dimensões.

✗ C) Incorreta. Manter o array obriga o dashboard a tratar o dado aninhado e não entrega os atributos de cliente e produto.

✗ D) Incorreta. flatten achata um array de arrays em um único array; não gera linhas por item.

> **Dica de prova:** Uma linha por elemento de array: explode; tabela larga para BI: join das dimensões na gold.

04

CAPÍTULO 4 · 15-20% DA PROVA

Proteger e governar objetos do Unity Catalog

Aqui eu preciso conceder o menor privilégio que resolve, proteger linhas e colunas (filtros de linha, máscaras, views dinâmicas e ABAC com tags), guardar segredos em escopos apoiados no Azure Key Vault, usar entidades de serviço e identidades gerenciadas com Access Connector, e ainda documentar, reter, auditar, rastrear linhagem e compartilhar com Delta Sharing (OpenSharing). O domínio vale 15-20% da prova. As alternativas erradas costumam dar privilégio demais (ALL PRIVILEGES, MANAGE), propor GRANT por coluna ou com WHERE, ou usar PAT e usuário nominal. Onde mais erro: a tripla USE CATALOG + USE SCHEMA + SELECT, logRetentionDuration x deletedFileRetentionDuration e quem precisa de qual privilégio no ABAC e no Delta Sharing.

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

Tópicos-chave

Privilégios em camadas

Ler tabela exige USE CATALOG no catálogo, USE SCHEMA no esquema e SELECT na tabela; criar tabela exige USE CATALOG, USE SCHEMA e CREATE TABLE no esquema. Concedido no esquema ou no catálogo, o privilégio é herdado pelas tabelas criadas depois. ALL PRIVILEGES e MANAGE quase sempre violam o menor privilégio.

BROWSE e MANAGE

BROWSE no catálogo deixa ver metadados sem ler dados, viabilizando busca, linhagem e pedidos de acesso. MANAGE administra privilégios e objetos, mas não é exceção a filtros e máscaras.

Principais: grupos de conta, owner e entidade de serviço

Conceda a grupos de conta sincronizados do Microsoft Entra ID; grupos locais do workspace são legados. Para não depender de pessoa, transfira a propriedade com ALTER TABLE ... SET OWNER TO um grupo. GRANT para entidade de serviço usa o application ID entre crases, depois de adicioná-la à conta e ao workspace.

Entidades de serviço e autenticação

Automação chamando APIs usa entidade de serviço com OAuth M2M, não PAT nem conta compartilhada. A entidade gerenciada pelo Microsoft Entra ID autentica no Databricks e no Azure; a gerenciada pelo Databricks só no Databricks. Job de produção deve ter Run as = entidade de serviço.

Filtros de linha

Crie uma função SQL com a regra e aplique com ALTER TABLE ... SET ROW FILTER funcao ON (coluna). Regras que mudam sempre usam tabela de mapeamento consultada com EXISTS e current_user(). Para remover: primeiro DROP ROW FILTER, depois DROP FUNCTION; revogar EXECUTE não tira o filtro.

Máscaras de coluna e views dinâmicas

Não existe GRANT por coluna no Unity Catalog: esconda colunas com máscara (SET MASK) ou view dinâmica com CASE WHEN is_account_group_member(...). Máscara que depende de outra coluna da linha usa SET MASK ... USING COLUMNS. Expor só o resultado de uma view: GRANT SELECT na view, nada na tabela base.

Computação para controle refinado

Tabelas com filtro ou máscara exigem computação standard, SQL warehouse ou serverless. Em dedicated, só a partir do Databricks Runtime 15.4 LTS, com serverless habilitado, porque a filtragem é delegada ao serverless. Ler os arquivos Delta pelo caminho direto não é suportado.

ABAC: governed tags e políticas

Mesma regra para muitas tabelas, inclusive futuras: política ABAC de ROW FILTER ou de máscara no catálogo ou no esquema, com MATCH COLUMNS has_tag_value(...) e EXCEPT para grupos isentos. Quem cria a política precisa de MANAGE ou propriedade e EXECUTE na UDF; o steward precisa de ASSIGN na governed tag e APPLY TAG nos objetos. Governed tag com valores permitidos evita tags inconsistentes.

Escopos de segredo com Azure Key Vault

Segredo que já vive no Key Vault: escopo apoiado no Key Vault e dbutils.secrets.get(scope, key). O escopo é só leitura no Databricks (grave no cofre), exige o modelo Vault access policy (não Azure RBAC) e a ACL é por escopo inteiro, não por segredo. Em Spark config ou variável de ambiente use {{secrets/escopo/chave}}; [REDACTED] na saída indica redação funcionando.

Access Connector e storage credential

Identidade gerenciada no Unity Catalog: Access Connector for Azure Databricks com identidade gerenciada, função Storage Blob Data Contributor na conta de armazenamento (Reader é plano de controle) e storage credential com o ID do access connector. Depois, external location sobre a credencial (CREATE EXTERNAL LOCATION no metastore e na storage credential), sem chave de conta nem SAS.

External location e acesso a arquivos

Ler e gravar arquivos por caminho na external location: READ FILES e WRITE FILES concedidos nela. Storage credential não recebe SELECT. Com firewall na conta de armazenamento, adicione regra de instância de recurso para Microsoft.Databricks/accessConnectors em vez de liberar todos os serviços confiáveis.

Comentários e tags

Documente com COMMENT ON TABLE e ALTER TABLE ... ALTER COLUMN ... COMMENT; colunas sem descrição aparecem em information_schema.columns WHERE comment IS NULL. Comentário gerado por IA deve ser revisado e editado antes de salvar e não serve para classificar dados pessoais. Classifique colunas com SET TAGS e encontre com information_schema.column_tags.

Retenção, histórico e exclusão definitiva

delta.logRetentionDuration controla o histórico de operações; delta.deletedFileRetentionDuration controla quanto tempo o VACUUM preserva arquivos antigos. A retenção padrão do VACUUM (7 dias) é o limite prático da time travel. Apagar de verdade: DELETE, REORG TABLE ... APPLY (PURGE) e VACUUM após a retenção. Quem mudou e como voltar: History (DESCRIBE HISTORY) e RESTORE TABLE.

Linhagem

A aba Lineage do Catalog Explorer mostra tabelas, colunas, jobs, notebooks e dashboards a jusante, para análise de impacto. Por SQL, system.access.table_lineage. Linhagem exige tabelas registradas no Unity Catalog (não no Hive metastore), é capturada em standard, dedicated e serverless e não é preservada após renomear objeto.

Auditoria e tabelas de sistema

system.access.audit responde quem concedeu acesso (action_name 'updatePermissions') e consultas de destinatários (deltaSharingQueriedTable); para guardar por mais tempo, copie de forma incremental para tabela própria, pois tabelas de sistema são somente leitura. Não administradores precisam de USE CATALOG system, USE SCHEMA e SELECT. No Log Analytics: configurações de diagnóstico; texto de comandos exige verbose audit logs.

Delta Sharing (OpenSharing)

Provedor: CREATE SHARE e CREATE RECIPIENT no metastore; destinatário Databricks-to-Databricks: CREATE CATALOG e USE PROVIDER. Destinatário com Unity Catalog usa Databricks-to-Databricks com sharing identifier; sem Databricks usa compartilhamento aberto com token ou federação OIDC e lista de acesso IP. Cortar acesso: REVOKE SELECT ON SHARE ... FROM RECIPIENT.

Conteúdo do share

ALTER SHARE ... ADD TABLE ... PARTITION (...) compartilha só parte; WITH HISTORY permite time travel e streaming no destinatário. Filtrar por parceiro: view dinâmica com CURRENT_RECIPIENT('propriedade') e propriedades no destinatário. Compartilhe a partir de catálogo dedicado com objetos preparados, nunca das tabelas internas cruas.

Macetes para memorizar

Leitura de tabela

Catálogo, esquema, tabela: USE, USE, SELECT. Criar: USE, USE, CREATE TABLE.

Volume x tabela x external location

Tabela = SELECT e MODIFY; volume = READ e WRITE VOLUME; caminho = READ e WRITE FILES.

Retenção

Log lembra o que aconteceu; deleted file guarda os dados. Time travel precisa dos dois.

Key Vault

Key Vault no Databricks: lê aqui, escreve lá, política de acesso e nunca RBAC.

ABAC

Política: MANAGE + EXECUTE. Tag: ASSIGN + APPLY TAG.

Delta Sharing

Provedor cria SHARE e RECIPIENT; destinatário cria CATALOG com USE PROVIDER.

Remover filtro

Desliga da tabela, depois apaga a função.

Tabela de memorização: não confunda

Quando é...	...e quando é
Filtro de linha Esconder linhas inteiras de quem não deve vê-las (por região, regional).	**Máscara de coluna** Manter a linha e ocultar o valor de uma coluna (CPF, telefone), inclusive dependendo de outra coluna com USING COLUMNS.
Política ABAC no catálogo ou esquema Centenas de tabelas com a mesma tag sensível, inclusive as futuras.	**Máscara ou filtro tabela a tabela** Regra pontual em uma tabela específica.
delta.logRetentionDuration Quanto tempo o histórico de operações do log fica disponível.	**delta.deletedFileRetentionDuration** Quanto tempo o VACUUM preserva arquivos de dados antigos; é o que limita a time travel (mantenha o log maior ou igual).

Entidade de serviço do Microsoft Entra ID Precisa autenticar no Azure Databricks e também em recursos do Azure.	**Entidade de serviço gerenciada pelo Databricks** Só acessa o Azure Databricks.
Databricks-to-Databricks Destinatário tem workspace com Unity Catalog; usa sharing identifier.	**Compartilhamento aberto** Destinatário sem Databricks (pandas, Power BI); token ou OIDC com lista de IPs.
system.access.audit Quem concedeu privilégio, quando, e quem consultou um share.	**system.access.table_lineage** Quem lê ou grava a partir de uma tabela (dependências a jusante).
SHOW GRANTS Ver os privilégios atuais de um objeto.	**Auditoria (updatePermissions)** Saber quem concedeu e quando.

Números que caem

Item	Valor
Retenção padrão do VACUUM (delta.deletedFileRetentionDuration), limite prático da time travel	**7 dias**
Retenção padrão do histórico do log (delta.logRetentionDuration)	**30 dias**
Runtime mínimo para dedicated ler tabela com filtro ou máscara	**Databricks Runtime 15.4 LTS, com serverless habilitado**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** GRANT SELECT ... WHERE regiao = 'SUL' para cada grupo.
Como não cair: GRANT não aceita WHERE; linhas se restringem com filtro de linha ou view dinâmica.

⚠ **Alerta de prova:** GRANT por colunas para esconder salário e CPF.
Como não cair: O Unity Catalog não tem GRANT por coluna; use máscara de coluna ou view dinâmica.

⚠ **Alerta de prova:** Criar a view filtrada e conceder SELECT também na tabela base.
Como não cair: SELECT na tabela base dá acesso a tudo; conceda só na view.

⚠ **Alerta de prova:** View dinâmica com is_account_group_member no WHERE para ocultar uma coluna.
Como não cair: No WHERE some a linha inteira; para ocultar só a coluna use CASE WHEN no SELECT.

⚠ **Alerta de prova:** Revogar EXECUTE na função para desligar o filtro de linha.
Como não cair: O filtro continua aplicado; rode DROP ROW FILTER e só depois DROP FUNCTION.

⚠ **Alerta de prova:** Conceder MANAGE para o engenheiro escapar do filtro e da máscara.
Como não cair: MANAGE administra o objeto mas não é exceção à regra da função; exceções vão na lógica (grupo) ou no EXCEPT da política.

⚠ **Alerta de prova:** Mudar o Key Vault para Azure RBAC ou dar put-acl por segredo.
Como não cair: Escopo apoiado no Key Vault exige Vault access policy, e a ACL é por escopo; separe equipes em cofres e escopos diferentes.

⚠ **Alerta de prova:** Conceder WRITE no escopo do Key Vault para gravar um segredo pelo Databricks.
Como não cair: Esse escopo é só leitura no Databricks; a gravação é feita no próprio cofre.

⚠ **Alerta de prova:** [REDACTED] na saída significa que o segredo expirou ou não foi lido.
Como não cair: É a redação funcionando; segredo inexistente gera erro na chamada.

⚠ **Alerta de prova:** Função Reader na conta de armazenamento para a identidade ler e gravar.
Como não cair: Reader é plano de controle; dados exigem Storage Blob Data Contributor.

⚠ **Alerta de prova:** Habilitar a exceção de serviços confiáveis do Azure no firewall do storage.
Como não cair: Libera qualquer access connector do tenant; use regra de instância de recurso para Microsoft.Databricks/accessConnectors.

⚠ **Alerta de prova:** Aumentar só logRetentionDuration para manter time travel longa ou auditoria de anos.
Como não cair: Time travel também depende dos arquivos que o VACUUM remove: ajuste deletedFileRetentionDuration e mantenha o log maior ou igual. Auditoria longa se faz copiando system.access.audit, e tabelas de sistema não aceitam ALTER.

⚠ **Alerta de prova:** Linhagem não aparece, então trocar o job para dedicated ou ativar Change Data Feed.
Como não cair: A origem provavelmente está no Hive metastore; linhagem exige tabela no Unity Catalog.

⚠ **Alerta de prova:** Conceder SELECT nas tabelas diretamente ao destinatário do share.
Como não cair: Destinatário recebe SELECT no share e cria catálogo a partir dele; filtro por parceiro é view dinâmica com CURRENT_RECIPIENT.

Como cai na prova

📝 **Como cai:** Grupo precisa de acesso X; alternativas com ALL PRIVILEGES, MANAGE, BROWSE e a tripla de USE.
Como resolver: Elimine o que dá mais que o pedido e o que dá menos (BROWSE não lê, READ VOLUME é de volume); fique com USE CATALOG + USE SCHEMA + privilégio do objeto, no nível que cobre o escopo pedido.

📝 **Como cai:** Ocultar dado sensível por grupo ou por região.
Como resolver: Pergunte: some a linha ou o valor? Linha = filtro de linha; valor = máscara; muitas tabelas com tag = ABAC no catálogo ou esquema; regra que muda = tabela de mapeamento.

📝 **Como cai:** Acesso a storage do Azure com segurança e sem rotação de segredo.
Como resolver: Access Connector com identidade gerenciada, Storage Blob Data Contributor, storage credential, external location, READ FILES ou WRITE FILES; com firewall, regra de instância de recurso.

📝 **Como cai:** Retenção e time travel de N dias.
Como resolver: Ajuste deletedFileRetentionDuration para N e mantenha logRetentionDuration maior ou igual; lembre que o VACUUM padrão retém 7 dias.

📝 **Como cai:** Compartilhar com parceiro externo.
Como resolver: Veja se ele tem Unity Catalog (D2D) ou não (aberto); recorte com PARTITION ou view com CURRENT_RECIPIENT; revogue com REVOKE SELECT ON SHARE; comprove com system.access.audit.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
ler uma tabela com o menor privilégio	**USE CATALOG + USE SCHEMA + SELECT**
tabelas futuras do esquema também	**Conceder no esquema ou catálogo (herança)**
ver metadados sem ler dados	**BROWSE no catálogo**
job quebra quando o funcionário sai	**Run as entidade de serviço e owner = grupo**
automação chamando APIs sem PAT	**Entidade de serviço com OAuth M2M**
segredo já está no Key Vault	**Escopo apoiado no Key Vault + dbutils.secrets.get**
segredo em Spark config ou variável de ambiente	**{{secrets/escopo/chave}}**
acesso ao storage sem chave nem SAS	**Access Connector + storage credential + external location**
mesma regra sensível para centenas de tabelas	**Política ABAC com governed tags**

regras de acesso que mudam toda semana	**Filtro de linha com tabela de mapeamento**
análise de impacto antes de mudar coluna	**Aba Lineage do Catalog Explorer**
quem concedeu acesso e quando	**system.access.audit com updatePermissions**
parceiro sem Databricks, sem token longo	**Compartilhamento aberto com OIDC e lista de IPs**
apagar dados pessoais de verdade	**DELETE + REORG TABLE APPLY (PURGE) + VACUUM após a retenção**

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

- Não existe GRANT por coluna nem GRANT com WHERE.
- Ler tabela = USE CATALOG + USE SCHEMA + SELECT.
- Escopo apoiado no Key Vault: só leitura no Databricks, ACL por escopo, Vault access policy.
- Time travel depende de deletedFileRetentionDuration e do VACUUM, não só do log.
- Linhagem exige tabela no Unity Catalog.

Essenciais para a prova (checklist)

- ☐ Sei montar a combinação mínima de privilégios para ler, criar e descobrir objetos.
- ☐ Sei criar e remover filtros de linha e máscaras de coluna, inclusive com USING COLUMNS.
- ☐ Sei montar uma política ABAC e dizer quem precisa de MANAGE, EXECUTE, ASSIGN e APPLY TAG.
- ☐ Sei usar escopos de segredo com Key Vault, dbutils.secrets.get e {{secrets/escopo/chave}}.
- ☐ Sei configurar Access Connector, storage credential e external location.
- ☐ Sei escolher entre entidade de serviço do Entra ID e gerenciada pelo Databricks.
- ☐ Sei ajustar logRetentionDuration e deletedFileRetentionDuration para um requisito de retenção.
- ☐ Sei achar linhagem no Catalog Explorer e em system.access.table_lineage.
- ☐ Sei consultar system.access.audit e configurar diagnóstico com verbose audit logs.
- ☐ Sei configurar Delta Sharing D2D e aberto, com PARTITION, WITH HISTORY e CURRENT_RECIPIENT.

Questões de revisão do capítulo

R4.1 · Proteger e governar objetos do Unity Catalog

O engenheiro João precisa criar no Unity Catalog uma external location para o contêiner landing da conta stparceiros, usando a storage credential cred_parceiros, que já existe. A criação falha por falta de permissão. Quais privilégios ele precisa e como confirmar que o acesso ao armazenamento funciona?

A. READ FILES e WRITE FILES na storage credential; depois, executar dbutils.fs.mount no contêiner
B. Storage Blob Data Reader para o usuário João no Azure; depois, criar um volume gerenciado de teste
C. CREATE CATALOG no metastore e USE SCHEMA no esquema padrão; depois, listar o contêiner com ls
D. CREATE EXTERNAL LOCATION no metastore e na storage credential; depois, testar a conexão no Catalog Explorer

R4.2 · Proteger e governar objetos do Unity Catalog

Uma operadora de saúde quer que os analistas consultem a tabela pacientes, mas só o grupo de conta auditoria_medica deve ver o CPF; os demais devem receber o texto OCULTO. A equipe prefere resolver com uma view, sem alterar a tabela. Qual definição atende?

A. CREATE VIEW v AS SELECT id, CASE WHEN current_user() = 'auditoria_medica@grupo.com' THEN cpf ELSE 'OCULTO' END AS cpf FROM pacientes
B. CREATE VIEW v AS SELECT id, CASE WHEN is_account_group_member('auditoria_medica') THEN cpf ELSE 'OCULTO' END AS cpf FROM pacientes
C. CREATE VIEW v AS SELECT id, cpf FROM pacientes WHERE is_account_group_member('auditoria_medica')
D. CREATE VIEW v AS SELECT id, CASE WHEN is_member('auditoria_medica') THEN 'OCULTO' ELSE cpf END AS cpf FROM pacientes

R4.3 · Proteger e governar objetos do Unity Catalog

A área de compliance de um banco precisa guardar por cinco anos os eventos de auditoria do Azure Databricks (quem acessou dados, concedeu privilégios ou alterou objetos do Unity Catalog) e consultá-los por SQL. A equipe já consulta system.access.audit, mas descobriu que a tabela de sistema não atende ao prazo. O que você deve fazer?

A. Consultar a aba History de cada tabela do catálogo, que registra os acessos de leitura por cinco anos
B. Agendar um job que copie de forma incremental os eventos novos de system.access.audit para uma tabela Delta própria
C. Executar VACUUM com RETAIN de cinco anos na tabela system.access.audit a cada mês
D. Alterar delta.logRetentionDuration da tabela system.access.audit para cinco anos com ALTER TABLE

R4.4 · Proteger e governar objetos do Unity Catalog

O job de produção de uma corretora roda com a identidade do engenheiro que o criou. Quando ele saiu da empresa e a conta dele foi desativada no Microsoft Entra ID, o job passou a falhar. Como você deve evitar que isso se repita?

A. Reativar a conta do ex-funcionário no Microsoft Entra ID para uso restrito do job de produção
B. Trocar o Run as para outro engenheiro sênior da equipe que tenha acesso às mesmas tabelas do job
C. Configurar o Run as do job com uma entidade de serviço que tenha os privilégios necessários
D. Gerar um token pessoal (PAT) de outro engenheiro e gravá-lo nos parâmetros de cada tarefa do job

Respostas comentadas

R4.1. D) CREATE EXTERNAL LOCATION no metastore e na storage credential; depois, testar a conexão no Catalog Explorer

Por quê: External locations combinam um caminho de armazenamento com uma storage credential. Quem cria precisa de CREATE EXTERNAL LOCATION no metastore e na credencial; depois, o acesso dos usuários é governado com READ FILES, WRITE FILES, CREATE EXTERNAL TABLE e CREATE MANAGED STORAGE.

✗ A) Incorreta. READ FILES e WRITE FILES se concedem na external location depois de criada, e montagens do DBFS são legado.
✗ B) Incorreta. O acesso ao armazenamento é feito pela identidade da storage credential, não pela conta do usuário no Azure.
✗ C) Incorreta. CREATE CATALOG e USE SCHEMA não autorizam criar external locations.

> **Dica de prova:** Criar external location: CREATE EXTERNAL LOCATION no metastore e na storage credential.

R4.2. B) CREATE VIEW v AS SELECT id, CASE WHEN is_account_group_member('auditoria_medica') THEN cpf ELSE 'OCULTO' END AS cpf FROM pacientes

Por quê: Views dinâmicas aplicam controle por coluna e por linha com funções como is_account_group_member e current_user. Os usuários recebem SELECT na view, e a tabela base fica protegida.

✗ A) Incorreta. current_user() devolve o e-mail do usuário, não o nome de um grupo; a comparação nunca identifica membros do grupo.
✗ C) Incorreta. Um filtro no WHERE esconde linhas inteiras de quem não é do grupo, em vez de ocultar só a coluna.
✗ D) Incorreta. is_member verifica grupos locais do workspace e a lógica está invertida: membros veriam OCULTO.

> **Dica de prova:** Ocultar coluna por grupo em uma view: CASE WHEN is_account_group_member(...).

R4.3. B) Agendar um job que copie de forma incremental os eventos novos de system.access.audit para uma tabela Delta própria

Por quê: A tabela system.access.audit guarda os eventos de auditoria com retenção gratuita de 365 dias (eventos de workspace são regionais). Para exigências regulatórias maiores, mantenha uma cópia incremental em uma tabela própria.

✗ A) Incorreta. A aba History mostra operações de escrita e manutenção da tabela, não acessos de leitura nem concessões de privilégios.

✗ C) Incorreta. VACUUM remove arquivos antigos de tabelas Delta; não estende a retenção de uma tabela de sistema, que é somente leitura.

✗ D) Incorreta. Tabelas de sistema são somente leitura e gerenciadas pelo Azure Databricks; as propriedades delas não podem ser alteradas.

> **Dica de prova:** Auditoria por mais de um ano: copie system.access.audit para uma tabela própria.

R4.4. C) Configurar o Run as do job com uma entidade de serviço que tenha os privilégios necessários

Por quê: Jobs e pipelines de produção devem rodar como entidade de serviço (Run as). Quem configura precisa da função Service Principal User na entidade, e os privilégios do Unity Catalog são concedidos à entidade de serviço.

✗ A) Incorreta. Manter ativa a conta de quem saiu é um risco de segurança e viola a governança de identidades.

✗ B) Incorreta. Trocar por outro usuário nominal só adia o problema até a próxima saída ou mudança de função.

✗ D) Incorreta. Um PAT de usuário em parâmetros expõe credencial e continua atrelado a uma pessoa que também pode sair.

> **Dica de prova:** Produção não deve depender de usuário nominal: Run as = entidade de serviço.

DIA DA PROVA

Estratégia de prova

O DP-750 traz cenários de empresas com um sintoma e um requisito explícito (menor privilégio, menor custo, sem refazer, sem segredo guardado, sem perder disparos) e pede a opção que atende exatamente a esse requisito. As alternativas erradas costumam ser recursos reais aplicados ao problema errado: VACUUM no lugar de OPTIMIZE, Run if no lugar de If/else, Run now no lugar de Repair run. Recursos aposentados ou legados (montagens do DBFS, Hive metastore, init scripts no DBFS, No isolation shared) só aparecem como distratores. Os nomes atuais valem: Lakeflow Spark Declarative Pipelines, Declarative Automation Bundles, AUTO CDC, Genie Agent e serverless usage policies. Preparar e processar dados e Implantar e manter pipelines pesam 30-35% cada; os outros dois, 15-20% cada.

Como responder

✓ Administre 120 minutos para a prova; a Microsoft não divulga o número exato de questões, então controle o ritmo pelo relógio, não pela contagem.

✓ Leia o requisito final antes das alternativas e identifique o verbo restritivo: sem copiar, sem refazer, sem segredo, menor privilégio, sem rodar continuamente.

✓ Em múltipla resposta, confira quantas opções o enunciado pede (Escolha DUAS ou TRÊS) e trate cada opção como verdadeiro ou falso isolado.

✓ Elimine primeiro qualquer alternativa com recurso legado (DBFS root, montagem, init script no DBFS, Hive metastore, PAT de usuário, No isolation shared).

✓ Entre duas opções que funcionam, prefira a gerenciada pela plataforma: predictive optimization, AUTO CDC, Lakeflow Connect, política ABAC, serverless.

✓ Em privilégios, descarte ALL PRIVILEGES e MANAGE quando o pedido é só ler ou criar, e confira se a tripla USE CATALOG + USE SCHEMA está presente.

✓ Em código e sintaxe, procure o detalhe que quebra a opção: timeout em segundos, ? no cron Quartz, ORDER BY na janela acumulada, --var no deploy e não no run.

✓ Quando dois serviços parecem servir, volte ao sintoma: Photon não acelera UDF nem RDD, pool não serve a SQL warehouse, Auto Loader não lê Event Hubs, MANAGE não dispensa filtro nem máscara.

Erros que derrubam candidatos

✗ Confundir Run if (estado das tarefas) com If/else condition (valor de um dado).

✗ Usar VACUUM para compactar arquivos ou ZORDER em tabela com liquid clustering.

✗ Dar privilégio demais (ALL PRIVILEGES, MANAGE) ou propor GRANT por coluna ou com WHERE.

✗ Escolher Run now ou aumentar retries quando o caso pede Repair run com parâmetro corrigido.

✗ Ler Event Hubs com Auto Loader ou usar a porta 5672 em vez da fonte Kafka na 9093.

✗ Achar que logRetentionDuration sozinho garante time travel, esquecendo deletedFileRetentionDuration e o VACUUM.

✗ Particionar por coluna de alta cardinalidade ou combinar partição com liquid clustering.

Na semana da prova

☐ Refaça os blocos de Preparar e processar dados e de Implantar e manter pipelines, que somam 60-70% da prova.

☐ Revise as tabelas de comparação: Run if x If/else, Repair run x Run now, streaming table x materialized view, filtro x máscara, D2D x compartilhamento aberto.

☐ Monte uma folha com os comandos de bundle (validate, deploy, run, summary, destroy) e os modos de schemaEvolutionMode e de expectations.

☐ Treine a tripla de privilégios e os privilégios de volume e de external location até sair sem pensar.

☐ Revise os nomes atuais e os antigos (Delta Live Tables, Databricks Asset Bundles, APPLY CHANGES, Genie space, budget policies) para reconhecer os dois.

☐ Na véspera, revise os números que mais caem: VACUUM 7 dias, log 30 dias, UNDROP 7 dias, timeout de dbutils.notebook.run em segundos, porta 9093 e partição só acima de 1 TB; a aprovação exige 700 em uma escala de 1 a 1000.

REVISÃO FINAL

Folha de cola

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

1. Configurar e ajustar um ambiente do Azure Databricks

‣ job agendado de produção, pagar só durante a execução → **Job compute definido no job (ou serverless para jobs)**

‣ sem gerir infraestrutura, partida rápida → **Computação serverless**

‣ RDD, Runtime ML ou GPU com vários usuários → **Modo dedicated atribuído a um grupo**

‣ JAR ou Maven bloqueado em cluster standard → **Allowlist do metastore**

‣ cluster demora a subir → **Pool com mínimo de instâncias ociosas**

⚠ SQL warehouses não usam pools; pools servem a clusters clássicos de notebooks e jobs.

⚠ Não existe configuração que habilite RDD no serverless: reescreva com a API DataFrame e tire o cache().

⚠ DBFS root e init scripts no DBFS são legado; o JAR bloqueado no standard vai para a allowlist do metastore, e o wheel vem de um volume.

2. Implantar e manter pipelines de dados e cargas de trabalho

‣ ramificar pelo valor de um task value → **If/else condition**

‣ rodar mesmo com tarefa pulada, mas não com falha → **Run if None failed**

‣ compensação só em erro → **Run if At least one failed**

‣ mesma lógica para cada item de uma lista que muda → **For each com concorrência limitada**

‣ falhou no meio de um job longo → **Repair run**

⚠ Run if olha estado, não valor; as duas rodariam. Ramificar por valor é If/else condition.

⚠ Pulada não conta como sucesso; tolerar pulada e barrar falha é None failed.

⚠ A tentativa repete o mesmo parâmetro; use Repair run informando o valor correto.

3. Preparar e processar dados

‣ ACID, versão anterior e upsert → **Delta Lake**

‣ origem SaaS com conector gerenciado → **Lakeflow Connect**

‣ servidor on-premises com Azure Data Factory → **Self-hosted integration runtime**

‣ arquivos chegando, incremental, sem cluster ligado → **Auto Loader + availableNow + checkpoint**

‣ carga SQL repetível que não recarrega arquivos → **COPY INTO**

⚠ Auto Loader lê arquivos; Event Hubs se lê com readStream format('kafka') na porta 9093.

⚠ 5672 é AMQP; Kafka usa 9093. A fonte Kafka não tem desserializador: cast('string') + from_json.

⚠ Linha apagada some da origem e não aparece na consulta incremental; exclusões pedem CDC.

4. Proteger e governar objetos do Unity Catalog

‣ ler uma tabela com o menor privilégio → **USE CATALOG + USE SCHEMA + SELECT**

‣ tabelas futuras do esquema também → **Conceder no esquema ou catálogo (herança)**

‣ ver metadados sem ler dados → **BROWSE no catálogo**

‣ job quebra quando o funcionário sai → **Run as entidade de serviço e owner = grupo**

‣ automação chamando APIs sem PAT → **Entidade de serviço com OAuth M2M**

⚠ GRANT não aceita WHERE; linhas se restringem com filtro de linha ou view dinâmica.

⚠ O Unity Catalog não tem GRANT por coluna; use máscara de coluna ou view dinâmica.

⚠ SELECT na tabela base dá acesso a tudo; conceda só na view.

GABARITO

Respostas do diagnóstico

D1. A) Pedir ao administrador do metastore que inclua o JAR e a coordenada Maven na allowlist

→ revise o capítulo 1

Por quê: Na computação com modo de acesso standard, o Unity Catalog exige que JARs, coordenadas Maven e init scripts estejam na allowlist do metastore antes da instalação; quem mantém a allowlist é o administrador do metastore (ou quem tem MANAGE ALLOWLIST).

✗ B) Incorreta. READ VOLUME permite ler o arquivo, mas não libera a instalação de JAR nem de Maven no modo standard, que depende da allowlist.

✗ C) Incorreta. Bibliotecas e init scripts no DBFS root são práticas legadas e obsoletas, e o bloqueio continuaria no modo standard.

✗ D) Incorreta. No isolation shared é um modo legado, sem isolamento entre usuários e sem a governança do Unity Catalog; não deve ser usado.

> **Dica de prova:** JAR, Maven ou init script bloqueado no modo standard: allowlist do metastore.

D2. B) USE CATALOG e CREATE SCHEMA → revise o capítulo 1

Por quê: No Unity Catalog, criar um objeto exige o privilégio CREATE no contêiner pai e USE nos níveis acima: esquema exige USE CATALOG + CREATE SCHEMA; tabela exige USE CATALOG + USE SCHEMA + CREATE TABLE.

✗ A) Incorreta. CREATE CATALOG permite criar catálogos, o que é proibido, e USE SCHEMA não permite criar esquemas.

✗ C) Incorreta. Sem USE CATALOG, o grupo não consegue acessar o catálogo, e CREATE SCHEMA sozinho não basta.

✗ D) Incorreta. MANAGE e ALL PRIVILEGES dão administração e privilégios muito além do necessário.

> **Dica de prova:** Criar esquema: USE CATALOG + CREATE SCHEMA no catálogo.

D3. B) O código falhará; reescreva com a API DataFrame e remova o cache() → revise o capítulo 1

Por quê: A computação serverless para notebooks e jobs elimina a gestão de infraestrutura, mas tem limitações: só APIs do Spark Connect (sem RDD), sem R, sem init scripts e bibliotecas de computação, sem APIs de cache e com poucas configurações Spark suportadas.

✗ A) Incorreta. Init scripts com escopo de computação não são suportados no serverless e não habilitariam a API RDD.

✗ C) Incorreta. Não existe configuração que habilite RDD no serverless, e a maioria das configurações Spark nem pode ser alterada nesse tipo de computação.

✗ D) Incorreta. A API RDD não é suportada na computação serverless.

> **Dica de prova:** RDD, R, init scripts ou cache() no código: revise antes de migrar para serverless.

D4. A) Definir a fonte das tarefas como Git provider, apontando para a tag de release → revise o capítulo 2

Por quê: Jobs de produção devem apontar para uma fonte Git (branch, tag ou commit) ou ser implantados por bundle, em vez de usar cópias editáveis no workspace. Isso garante rastreabilidade e que só código revisado execute.

✗ B) Incorreta. Restringir edição reduz o risco, mas o job continua rodando uma cópia que pode divergir do repositório.

✗ C) Incorreta. Importar arquivos DBC é manual, sujeito a erro e não liga a execução a uma versão do repositório.

✗ D) Incorreta. Uma Git folder pessoal pode estar em qualquer branch e com alterações não enviadas, além de depender de um usuário.

> **Dica de prova:** Produção roda o que está no Git: fonte Git na tarefa com tag ou commit.

D5. D) Editar o arquivo, remover os marcadores, marcar como resolvido e continuar o merge → revise o capítulo 2

Por quê: As Git folders mostram os arquivos em conflito em pull, merge ou rebase e oferecem três caminhos: editar e marcar como resolvido, aceitar uma das versões inteira (Keep all current changes ou Take all incoming changes) ou abortar a operação com Abort.

✗ A) Incorreta. Take all incoming changes aceita integralmente a versão que está chegando da main e descarta as linhas dela, mas o requisito é manter parte das duas.

✗ B) Incorreta. Recriar a Git folder não resolve o conflito, apenas o adia, e arrisca perder alterações; a própria interface oferece a resolução.

✗ C) Incorreta. Reset equivale a git reset --hard com push forçado: descarta as alterações da branch, o oposto de preservar parte delas.

> **Dica de prova:** Conflito em Git folder: edite, Mark As Resolved e Continue Merge; Abort desfaz tudo.

D6. D) Um grupo de ações com ação de SMS e ação de webhook, associado à regra de alerta → revise o capítulo 2

Por quê: No Azure Monitor, a regra de alerta define a condição (por exemplo, uma consulta KQL de log), e o grupo de ações define as respostas: notificações (e-mail, SMS, push, voz) e ações (webhook, Logic App, Azure Function, runbook).

✗ A) Incorreta. Pastas de trabalho servem para visualização; não disparam notificações.

✗ B) Incorreta. Enviar logs a um Event Hub muda o destino dos dados, mas não notifica ninguém.

✗ C) Incorreta. As notificações de job do Azure Databricks usam e-mail e destinos como Teams, Slack e webhooks, mas não enviam SMS, e não aproveitam a regra de alerta já criada.

> **Dica de prova:** Quem avisar e o que acionar quando o alerta dispara: grupo de ações.

D7. A) Criar um pipeline de ingestão com o conector gerenciado do ServiceNow no Lakeflow Connect → revise o capítulo 3

Por quê: Quando existe conector gerenciado do Lakeflow Connect para a origem, ele é a opção de menor manutenção: cria tabelas de streaming no Unity Catalog, controla o incremento e roda como pipeline agendável. Notebook com código próprio fica para origens sem conector.

✗ B) Incorreta. Um notebook com código próprio funciona, mas obriga a equipe a manter paginação, incrementalidade e tratamento de limites de API, o que o cenário quer evitar.

✗ C) Incorreta. Montagens do DBFS são legado; o recomendado é Unity Catalog com volumes e external locations, e isso ainda exigiria um processo de exportação.

✗ D) Incorreta. O Lakehouse Federation consulta dados no lugar, sem copiar; o requisito é ingerir de forma incremental para tabelas no Unity Catalog.

> **Dica de prova:** Origem SaaS com conector gerenciado: Lakeflow Connect antes de código próprio.

D8. D) startingOffsets = 'earliest' e a connection string lida de um escopo de segredo no kafka.sasl.jaas.config → revise o capítulo 3

Por quê: Na fonte Kafka do Spark, startingOffsets (earliest ou latest) vale só quando a consulta começa sem checkpoint; depois, os offsets vêm do checkpoint. Credenciais devem vir de escopos de segredo, como na montagem do kafka.sasl.jaas.config com a connection string do Event Hubs.

✗ A) Incorreta. latest começa pelos eventos que chegarem depois do início da consulta e ignora os já retidos; widget preenchido pelo job expõe o valor na configuração.

✗ B) Incorreta. A posição inicial está certa, mas a connection string escrita no notebook viola a política, mesmo com permissões restritas.

✗ C) Incorreta. Um grupo de consumidores novo não define a posição inicial da leitura no Spark, que guarda os offsets no checkpoint, e o segredo continuaria em texto.

> **Dica de prova:** Ler o que já está retido no Event Hubs: startingOffsets earliest; credencial sempre via segredo.

D9. A) SCD tipo 1, sobrescrevendo os atributos alterados na linha existente → revise o capítulo 3

Por quê: SCD tipo 1 sobrescreve atributos e mantém uma linha por chave; SCD tipo 2 cria nova versão com datas de vigência para preservar histórico. A escolha depende de o negócio precisar ou não analisar valores passados do atributo.

✗ B) Incorreta. A flag is_current é um recurso do SCD tipo 2 e também guarda versões que ninguém vai consultar.

✗ C) Incorreta. O SCD tipo 2 preserva histórico com novas linhas, algo que o negócio declarou desnecessário; aumentaria volume e complexidade.

✗ D) Incorreta. Snapshots diários duplicam a dimensão inteira todos os dias, desperdiçando armazenamento para um requisito de valor atual.

> **Dica de prova:** Só o valor atual importa: SCD tipo 1 (sobrescrever).

D10. C) COMMENT ON TABLE gold.inadimplencia IS '...' e ALTER TABLE gold.inadimplencia ALTER COLUMN dpd COMMENT '...' → revise o capítulo 4

Por quê: Descrições ficam em COMMENT, definido na criação (COMMENT na tabela e nas colunas) ou depois com COMMENT ON e ALTER COLUMN ... COMMENT. Gravar por código no deploy evita perdê-las ao recriar objetos.

✗ A) Incorreta. Tags servem para classificar e filtrar objetos; o texto descritivo pertence ao comentário.

✗ B) Incorreta. Uma view com alias muda o nome para quem a consulta, mas não documenta a tabela original.

✗ D) Incorreta. Propriedades de tabela são pares chave-valor técnicos; não são exibidos como descrição da tabela nem das colunas.

> **Dica de prova:** Descrição para descoberta: COMMENT na tabela e nas colunas.

D11. C) GRANT USE CATALOG no catálogo, mais USE SCHEMA e SELECT no esquema gold, ao grupo → revise o capítulo 4

Por quê: Os privilégios do Unity Catalog são herdados de cima para baixo: SELECT no esquema vale para todas as tabelas e views dele, inclusive as criadas depois. O principal também precisa de USE CATALOG e USE SCHEMA.

✗ A) Incorreta. Funciona, mas exige manutenção contínua e falha na janela entre a criação da tabela e o próximo job, que é o que se quer evitar.

✗ B) Incorreta. ANY FILE é um privilégio de acesso direto a arquivos que contorna a governança; sem USE CATALOG e SELECT no esquema, as tabelas continuam inacessíveis.

✗ D) Incorreta. ALL PRIVILEGES dá muito mais que leitura (inclusive criar e modificar objetos) e viola o menor privilégio.

> **Dica de prova:** Acesso a tabelas futuras: conceda no esquema (ou catálogo); a herança cobre o que for criado depois.

D12. D) Uma entidade de serviço com segredo OAuth, usando o fluxo OAuth M2M → revise o capítulo 4

Por quê: Para automação, a recomendação é usar entidades de serviço com OAuth M2M: o segredo OAuth gera tokens de acesso que expiram em pouco tempo, reduzindo o risco de vazamento. PATs são credenciais legadas de longa duração.

✗ A) Incorreta. Continua dependendo de uma pessoa e de um token de longa duração, o problema original.

✗ B) Incorreta. Contas compartilhadas prejudicam a rastreabilidade, e usuário e senha não são a forma de autenticação indicada para essa automação.

✗ C) Incorreta. A identidade passa a ser não humana, mas um PAT sem expiração é credencial de longa duração, que a recomendação atual substitui por OAuth.

> **Dica de prova:** Automação chamando APIs do Azure Databricks: entidade de serviço + OAuth M2M, não PAT.

Agora, treine no formato real

Faça as 6 provas completas do curso DP-750: 300 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 Microsoft. As questões são inéditas e não reproduzem a prova real.