Brought to you by:

databricks

Lakebase

for dummies

A Wiley Brand

Build on open, serverless Postgres

Power agents and apps at scale

Branch databases like you branch code



Databricks Special Edition

Stephanie Diamond

About Databricks

Databricks is the data and AI company. More than 20,000 organizations worldwide — including adidas, AT&T, Bayer, Block, Mastercard, Rivian, Unilever, and 70 percent of the Fortune 500 — rely on Databricks to build and scale data and AI apps, analytics, and agents. Headquartered in San Francisco, with 30+ offices around the globe, Databricks offers a unified platform that includes Lakebase, Genie, Agent Bricks, Lakeflow, Lakehouse, and Unity Catalog. To learn more, follow Databricks on LinkedIn, X, YouTube, and Instagram.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.



Lakebase

Databricks Special Edition

by Stephanie Diamond

for dummies®
A Wiley Brand

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Lakebase For Dummies®, Databricks Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

Copyright © 2026 by John Wiley & Sons, Inc., Hoboken, New Jersey. All rights, including for text and data mining, AI training, and similar technologies, are reserved.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permissions.

Trademarks: Wiley, For Dummies, the Dummies Man logo, The Dummies Way, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries, and may not be used without written permission. Databricks and the Databricks logo are registered trademarks of Databricks. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, or how to create a custom *For Dummies* book for your business or organization, please contact our Business Development Department in the U.S. at 877-409-4177, contact info@dummies.biz, or visit www.dummies.com/custom-solutions. For information about licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com.

ISBN 978-1-394-43062-8 (pbk); ISBN 978-1-394-43063-5 (ebk); ISBN 978-1-394-43064-2 (epub)

Publisher's Acknowledgments

Project Manager and Editor: Carrie Burchfield-Leighton

Acquisitions Editor: Traci Martin

Senior Managing Editor: Rev Mengle

Sales Manager: Molly Daugherty

Content Refinement Specialist: Athiyappan Lalith Kumar

Databricks Special Help: Susan Pierce, Bryan Clark, Jonathan Katz, Pranav Aurora, and Evan Pandya

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Table of Contents

INTRODUCTION 1

CHAPTER 1: **Introducing Lakebase** 3
Tracing the Evolution to Lakebase 4
Seeing how data warehouses created silos 4
Recognizing how the lakehouse unified analytics 4
Examining how Databricks extended the model to operations 5
Defining What a Lakebase Is 5
Seeing What's Broken in Today's Databases 6
Exposing the split-stack problem 6
Recognizing the AI-driven surge in database demand 7
Hitting the ceiling on scalability and cost 9
Understanding Why the Lakebase Architecture Is Emerging Now 9
Exploring Top Use Cases 10
Surveying key application use cases 10
Identifying high-impact AI use cases 10
Counting the True Cost of Staying Put 11
Tallying the engineering time spent on plumbing 11
Losing ground to faster competitors 12
Building Your First Lakebase-Powered App 12

CHAPTER 2: **Exploring the Lakebase Architecture** 13
Separating Compute from Storage 13
Cutting costs by separating compute from storage 14
Scaling only what the workload demands 14
Comparing a Lakebase to Familiar Systems 15
Contrasting Lakebase with traditional OLTP databases 15
Distinguishing Lakebase from the Lakehouse 16
Building on Postgres and open standards 17
Mapping the Full Architecture 18
Identifying the core layers 18
Seeing how each layer scales 19

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

CHAPTER 3: **Examining How Databricks Lakebase Works Under the Hood** 21
Running Serverless Compute for Transactional Workloads 21
Scaling compute elastically, including to zero 22
Protecting data integrity 22
Building on a Unified Data Foundation 23
Sharing one store for apps and analytics 23
Eliminating the data sync problem 24
Governing data with Unity Catalog 25
Delivering Reliability, Consistency, and Performance 26
Handling failure without interruption 26
Branching production data without copying it 26
Setting realistic performance expectations 27

CHAPTER 4: **Building Apps and AI with a Lakebase** 29
Using Lakebase as Your System of Record 29
Establishing a single source of truth 30
Structuring your data model for operational use 30
Supporting AI Agents with Real-Time Data 31
Defining what AI agents need from a database 31
Enabling real-time decisioning without data lag 32
Eliminating Data Movement and Pipeline Complexity 33
Calculating the pipeline tax 33
Simplifying your stack 34
Syncing changes automatically 35
Accelerating Developer Productivity 35
Adopting lakebase without retraining your team 35
Branching databases the way you branch code 36
Getting to production faster 37
Proving the Model with Customer Results 37

CHAPTER 5: **Tapping into the Databricks Platform** 39
Connecting Lakebase to the Databricks Ecosystem 39
Building Apps with Databricks Apps and Developer Tools 40
Connecting with AI Coding Assistants and Popular SDKs 41
Simplifying Your Stack with a Unified Foundation 42

CHAPTER 6: **Ten Reasons Why You Need a Lakebase** 43

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Introduction

Database infrastructure is at an inflection point. For four decades, the core assumptions held: humans in the loop, separate systems for transactions and analytics, and infrastructure that scaled on a human schedule. Those assumptions no longer fit the way modern applications and artificial intelligence (AI) systems work.

AI agents and real-time applications work around the clock. They spin up servers on demand, fire off queries faster than any human, and expect transactional data and analytical context in one place. The old split-stack setup, with one database for transactions and another for analytics, can't keep up.

That is where the lakebase comes in. A lakebase is a new category of serverless Postgres database built alongside the lakehouse with separate compute and storage. This architecture enables you to unify apps, analytics, and AI on one foundation, spin up databases in seconds, branch them like code, and scale to zero when you're done.

About This Book

Welcome to *Lakebase For Dummies*, Databricks Special Edition. This book covers the essential concepts and architecture behind the lakebase architecture and shows you how to put it to work in your organization.

This book is for data and AI leaders, product teams, and senior engineers who need to quickly understand what a lakebase is and how it fits into their architecture. It examines how Databricks Lakebase addresses that architectural gap by combining a fully managed Postgres database with the openness, scalability, and AI capabilities of the lakehouse. It gives you the foundation to understand the lakebase and learn how it will benefit your organization.

Here's what you find inside:

- Understand why traditional databases break under AI workloads and real-time application demands.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

- See what the lakebase category is and how it differs from a data warehouse, data lake, and lakehouse.
- Understand how the lakebase architecture works and why each decision matters.
- Build modern apps and AI workflows on Databricks Lakebase.
- Quantify ten reasons why your organization should make the move to a lakebase architecture.

Icons Used in This Book

Throughout this book, various icons highlight important information. Here's what they mean:



The Tip icon highlights information that can make doing things easier or faster.



The Remember icon highlights items you need to recall when searching your memory bank.



The Warning icon alerts you to potential hazards that could harm you or your company.



Sometimes, I provide you with a few additional tidbits of research or facts beyond the basics. If you want to know the technical details, watch out for this icon.

Beyond the Book

This book focuses on what a lakebase is and how it works. If you want resources beyond what this short book offers, visit `www.databricks.com/product/lakebase`.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

- Tracing the evolution from data warehouse to lakebase
- Defining what a lakebase is
- Seeing what's broken in today's database architectures
- Counting the true cost of staying with the status quo

Chapter 1
Introducing Lakebase

Every traditional database was built around the core assumption that a human was in the loop. Someone logged in, asked a question, made a change, and moved on. That model has held up for four decades. But it doesn't hold up today.

The reason is artificial intelligence (AI) agents. They don't wait for someone to set up a database or schedule a query. They create databases, run experiments, spin up parallel environments, and discard everything when they're done. They do it in seconds. No human schedules it, no administrator approves it, and no traditional database handles this kind of demand.

That's what a lakebase is for. This chapter introduces the lakebase architecture and explains why it exists and what it replaces. You see how the data landscape got to this point, where today's architectures fall short, and what it can cost your organization to stay where you are.

Tracing the Evolution to Lakebase

Every technology category solves the problems left behind by the previous one. This section shows you how the path from the data warehouse to a lakebase evolved and why each step mattered.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Seeing how data warehouses created silos

In the 1980s and 1990s, the data warehouse was a big step forward for organizations needing a way to run reports and analyses without slowing down their core business systems. The solution was to copy the operational data into a separate analytics system.

For years, it worked well. But it locked organizations into a pattern that still causes problems today. You have two copies of your data and two systems to manage, and you need constant effort to keep them in sync.

The warehouse handled reporting. The operational database handled the day-to-day transactions. Each did its job. Together, they became the split stack (the name for keeping your transactional and analytical systems separate), and it's the root of most modern data complexity.



REMEMBER

The data warehouse solved a real problem. Its core design assumption was that operational and analytical data should be stored in separate systems. That assumption is exactly what a lakebase is built to replace.

Recognizing how the lakehouse unified analytics

The lakehouse came next. It brought structured and unstructured data onto a single platform. This eliminated the need to maintain separate data lakes and warehouses. Organizations could store raw data in open formats and run both reporting and machine learning on the same foundation, without moving data between systems.



TIP

Databricks built the lakehouse architecture that shaped this shift. It was a major improvement for analytics and data science teams.

The lakehouse changed how teams work with data. It collapsed the warehouse-and-lake split, opened analytics and AI to the same governed copy of your data, and quickly became the standard for analytical workloads. But a new kind of workload emerged at the same time. AI agents and apps need fast reads and writes at machine speed — the operational side of the house. The lakehouse wasn't built for that, and it didn't house need to be. A new layer was needed to meet it.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Examining how Databricks extended the model to operations

Lakebase closes the gap. It adds full operational database capabilities alongside the lakehouse foundation. Instead of running a separate transactional database alongside your lakehouse, you run both on the same platform.

What that means in practice is that you have one system for your agents, apps, your analytics, and your AI. No extra infrastructure and no pipelines moving data between layers.

Defining What a Lakebase Is

A lakebase is a new database category that runs both transactional and analytical workloads on a single lakehouse foundation. You get the full capabilities of an operational database without maintaining a separate system alongside your data platform.

Here are five principles that define what makes a lakebase different:

- **Openness:** Built on Postgres, the most popular database in the developer community, your existing tools, drivers, and frameworks connect to it from day one.
- **Compute and storage separation:** Data resides in object storage. Compute runs independently and scales on its own, and you pay only for what you use.
- **Serverless design:** Resources scale up when your workloads need them and down to zero when they don't. There's no idle infrastructure running up your bill.
- **Modern development workflows:** Branching a database works like branching code, instantly, with no data copied. Dev, staging, and production environments are straightforward to manage.
- **AI-agent scale:** A lakebase handles thousands of AI agents running at machine speed — each agent in its own isolated environment — without collisions and waiting.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Seeing What's Broken in Today's Databases

The problems with today's database architectures aren't hypothetical. They show up as budget overruns, delayed projects, and engineering teams spending time on maintenance rather than building. This section provides some insight into those issues.

Exposing the split-stack problem

Most organizations run two separate database systems. One handles *transactions*, which are the reads and writes that power your applications. That's the Online Transaction Processing (OLTP) system. The other handles *analytics* — the queries, reports, and dashboards. That's the Online Analytical Processing (OLAP) system. Keeping these two systems separate is what the industry calls *the split stack.*

Examining the OLTP and OLAP divide

The split made sense at the time. Transactional and analytical workloads have very different needs, query patterns, performance requirements, and data-organization approaches. Two specialized systems seemed like good engineering.

But your data resides in both places. Every change to the operational system must be reflected in the analytical system. That takes pipelines, and pipelines need engineers. Those are engineers who aren't building your product.

Stale data is the other cost. By the time a change in your app shows up in your analytics, hours have passed. For AI systems that need current data to make good decisions, that delay is a real problem.



WARNING

The split stack can create friction not just between systems but also between teams. The group managing your operational database and the group running your analytics platform often work on different schedules with different priorities. That friction is organizational overhead, not just technical debt.

Uncovering the hidden cost of data pipelines

One pipeline is fine. The problem is that you don't have one; you have dozens and sometimes hundreds. These are the Extract,

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Transform, Load (ETL) jobs that copy operational data into your analytical layer so reports, dashboards, and AI models can use it. Each one runs on its own schedule. Each one can break. And because they connect systems that were never designed to talk to each other, one engineer can spend most of the week just keeping them alive.

The cost of all this movement doesn't show up in a budget line. It shows up as

- **Engineering time:** Someone builds a pipeline. Someone has to update it when a data source changes or a schema shifts. That time adds up fast.
- **Failure cost:** When the pipeline breaks, your data goes stale, your dashboards break, and your teams make decisions based on outdated information.
- **Storage and compute costs:** You keep the same data in multiple places and run compute in multiple systems. That bill is larger than it looks.
- **Latency:** Even a healthy pipeline introduces a delay. For real-time applications, any delay is a problem.

  As more applications are built by humans and agents, the number of pipelines to manage increases drastically, compounding this friction.

Recognizing the AI-driven surge in database demand

The split-stack problems existed before AI. Organizations learned to manage them, but AI multiplies the pressure beyond what traditional architectures can handle.

Tracking the AI agent explosion

According to the 2026 *State of AI Agents* report, AI agents now create 80 percent of all databases on Neon (the serverless Postgres technology powering Databricks Lakebase). Two years ago, that figure was 27 percent, and in 2023 it was just 0.1 percent. The same data shows that AI agents account for 97 percent of all database testing and development environments. Multi-agent workflows grew 327 percent in under four months. This comes from more than 20,000 Databricks customers, including over 60 percent of the Fortune 500.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Serving real-time apps at scale

Real-time applications have always pushed databases to the limit. A customer-facing app needs a response in milliseconds. A financial platform needs to reflect account changes in real time. An AI system needs current data to make accurate decisions, not data from a pipeline that ran two hours ago. Traditional databases were built for human interaction speeds. But today's applications operate at machine speeds.

Picture what an AI agent actually needs. It requires its own isolated database environment to run tests against. It spins up that environment, runs a batch of experiments, and tears it down. Then it starts over. Now picture thousands of agents doing that in parallel. A traditional database requires manual provisioning for each environment and a full data copy for isolation. At that volume, provisioning times become a bottleneck and storage costs become prohibitive.



REMEMBER

This isn't a future scenario. AI agents are creating and destroying database environments in production right now. If your architecture requires a human to provision each one, it's a bottleneck.

Hitting the ceiling on scalability and cost

When demand grows faster than your architecture can absorb it, you hit a wall. Traditional databases respond to that wall in predictable ways. None of them is free.

Reaching the limits of vertical scaling

The traditional response to a performance problem is to scale up. That means you add more CPU, more memory, and faster storage to the same machine. This works up to a point. But vertical scaling has a ceiling. The largest available server only gets you so far, and the cost per unit of compute rises as you climb. You also hit a harder limit. A single server can only handle so many concurrent connections. For AI workloads that spin up thousands of parallel environments, that ceiling arrives fast.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Understanding the overprovisioning tax

The usual workaround is overprovisioning, buying more capacity than you need to leave headroom for demand spikes. That gets expensive fast. And it still doesn't solve the speed problem.

Understanding Why the Lakebase Architecture Is Emerging Now

Three things are happening at the same time. Together, they explain why lakebase is arriving now and not five years ago:

- **The AI agent explosion:** AI agents now create 80 percent of all databases on Neon. Human-in-the-loop provisioning is no longer viable at the scale AI demands.
- **The compounding cost of the split stack:** More data means more pipelines, more maintenance, and more duplication. That overhead scales with your data estate. For most organizations, it has been growing quietly for years.
- **The expectation of real-time data:** Users expect applications to reflect what's happening right now. AI systems need current data to make accurate decisions. A sync process that runs every few hours isn't acceptable anymore.

A lakebase addresses all three. It handles the elastic, machine-speed scaling AI agents require, eliminates the pipeline layer that makes split stacks expensive, and keeps data consistent across applications and analytics without custom integration work.

Exploring Top Use Cases

Lakebase handles two broad categories of workload: application use cases and AI use cases. Most organizations that adopt it find they're running both within the first few months.

Surveying key application use cases

Any application that talks to users has the same basic requirements: fast reads and writes, the ability to handle many users at

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

once, and the need to show users' information drawn from their history and context, not just their current session.

Lakebase fits these application patterns. Here are the most common use cases:

- **Customer-facing apps:** E-commerce platforms, financial dashboards, patient portals, any app where users expect an immediate response to their action
- **Internal tools:** Operations dashboards, team portals, and workflow systems that employees use every day and that need to stay current with what's happening
- **Software-as-a-Service (SaaS) platforms:** Multi-tenant applications where each customer's data stays isolated, and new environments spin up instantly

Identifying high-impact AI use cases

AI workloads make specific demands on a database: current data, isolated environments on demand, and no human in the loop for provisioning. High-impact use cases include

- **AI agents for customer support:** Agents that pull account history, update records, and log interactions in real time.
- **Supply chain optimization:** AI systems that update recommendations continuously as inventory, pricing, and logistics data change throughout the day.
- **Real-time feature serving:** Machine learning models that need fresh operational data as input, not data from yesterday's pipeline run.
- **Persistent agent memory:** AI agents that carry context from one session to the next, storing and retrieving state without a separate memory system bolted on.



Lakebase supports pgvector, the Postgres extension for vector similarity search. That means you can run AI-powered semantic search directly against your operational data, with no separate vector database to manage and keep in sync.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Counting the True Cost of Staying Put

The case for a lakebase isn't only about what you gain. It's also about putting a number on the cost of your current architecture.

Tallying the engineering time spent on plumbing

Every hour your engineers spend on pipeline maintenance, sync failures, and cross-system debugging is an hour they're not building product. That's not a technology problem. It's a business problem.

Here's a quick way to size it for your organization. Ask these questions:

- How many engineers spend time on data pipeline maintenance each week?
- How many hours go to debugging failures when a sync job breaks or a schema changes?
- How long does it take to onboard a new data source end-to-end?

Multiply your answers by the engineering cost. The result is usually larger than anyone expects. And the number grows as your data estate grows because every new source means new pipelines and new failure modes.

Losing ground to faster competitors

Shipping speed is a competitive factor. Organizations that ship faster, adapt faster, and serve users with more current data have an edge over those who don't.

The pattern has repeated before. Early cloud adopters outpaced competitors still running on-premises infrastructure. Early lakehouse adopters moved faster than those maintaining separate data lakes and warehouses. Lakebase represents the same kind of shift: an architectural change that compounds over time into a real competitive gap.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.



WARNING

Staying on a split-stack architecture doesn't mean holding steady. It means absorbing overhead that faster-moving competitors may not have to carry. That gap grows quietly, quarter by quarter, until it shows up in shipping speed and engineering cost.

Building Your First Lakebase-Powered App

Getting started on your first lakebase-powered app takes less setup than you may expect. Lakebase Postgres is fully compatible with the Postgres standard, so your existing tools, connection libraries, and frameworks connect to it without changes.

The basic path is straightforward. Create a Lakebase project in your Databricks workspace, point your application at it using a standard Postgres connection string, and deploy using Databricks Apps. (See Chapter 4 for more information.)

The case for lakebase isn't built on a single feature. It's built on the cost of the alternatives, split systems, pipeline maintenance, and an architecture designed before AI agents existed. After you add up that cost, the question shifts from whether to change to how.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

- Separating compute from storage
- Comparing lakebase to other databases and platforms
- Identifying the core layers of a lakebase

Chapter 2
Exploring the Lakebase Architecture

Lakebase architecture addresses three specific problems: the cost of bundling compute and storage, the gap between operational and analytical data, and the assumption that a human is always in the loop. Traditional databases leave all three unresolved. A lakebase builds a structural answer to each one into the architecture itself.

This chapter shows you how those choices translate into a working architecture. You learn about the layers, how they fit together, and how they scale.

Separating Compute from Storage

The single most important design decision in the lakebase architecture is that compute and storage are separate systems that scale independently. Everything else in the architecture follows from that choice.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Cutting costs by separating compute from storage

In a traditional database, you set up one thing: a server. That server bundles CPU and storage together in fixed amounts. You pay for both whether or not you use them. When your storage needs grow, you scale the whole server. When your compute demand spikes, you do the same. The two can't move independently because they're built into the same machine.

Cloud infrastructure changed the constraint that made this necessary. Networks are fast and reliable. Object storage is cheap. The performance penalty for separating compute from storage doesn't outweigh the benefit. A lakebase takes advantage of that shift.

With a lakebase, your data resides in cloud object storage — the same low-cost storage the lakehouse already uses. Compute runs as a separate, independent layer on top. You scale compute up when your application is busy; you scale it down when it's not. Storage grows as your data grows and only as fast as your data grows. The two costs are separate because the two systems are separate.

For most organizations, this changes the math on database spend. Traditional databases charge you for the peak capacity you may need, not the average capacity you actually use. A workload that spikes once a day at ten times normal traffic forces you to provision for that spike around the clock. With a lakebase, you provision for the average and scale up only when the traffic demands it. The difference compounds quickly across dozens of databases.


TIP

Separating compute from storage is the decision that makes everything else possible: branching, scaling to zero, instant provisioning, and read scaling without data duplication. Every capability that sets a lakebase apart from a traditional database can be traced back to this single architectural choice.

Scaling only what the workload demands

Separate compute and storage, and each layer scales independently. When demand is high, a lakebase can add compute. When demand drops, compute scales back to zero if nothing is running.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

A traditional database runs continuously even during idle periods because stopping and restarting it is slow and risks data loss. With a lakebase, stopping compute is safe because the data isn't tied to the compute layer. When demand returns, compute restarts instantly and attaches directly to the existing data in storage. No recovery process. No data risk.

The same logic applies to read replicas. A standard Postgres setup duplicates data across replicas, multiplying storage costs. With a lakebase, multiple compute instances read from the same storage layer simultaneously. You get the read performance of replicas without the storage overhead of copying the data.


REMEMBER

Scaling to zero doesn't mean your data disappears. The data stays in object storage. Only the compute layer stops. It restarts in hundreds of milliseconds when your application needs it again. For development environments and low-traffic workloads, this eliminates a significant portion of your database bill.


WARNING

Traditional database cost calculators don't apply to a lakebase. If you're estimating lakebase costs by looking at what you currently pay per instance, you're overestimating.

Comparing a Lakebase to Familiar Systems

Most organizations already run a database infrastructure. Understanding where a lakebase fits in relation to what you have is the fastest way to make adoption decisions. Here's how it compares to the two systems most teams already operate.

Contrasting Lakebase with traditional OLTP databases

Traditional online transaction processing (OLTP) databases are monolithic systems — compute and storage live together. Scaling means provisioning a larger instance. Cloning a database for testing means copying the data, which takes time and costs money. The more data you have, the higher the cost.

These systems assume persistent, human-managed infrastructure. Someone provisions the server. Someone monitors it.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Someone handles backups, updates, and capacity planning. That process works when you have a small number of databases managed by a dedicated team. It stops working when

- Artificial intelligence (AI) agents start creating thousands of database environments a day.
- You need to launch a new customer environment in seconds.
- Your engineering team spends more time on database maintenance than on building a product.

The infrastructure model is what breaks down, not the SQL interface.

A lakebase keeps full Postgres compatibility. Your SQL, drivers, connection libraries, and existing tools all work without changes. What changes is the infrastructure underneath: a cloud-native architecture that handles machine-speed provisioning, elastic scaling, and instant branching by design.



REMEMBER

A lakebase doesn't replace the Postgres interface; it replaces the infrastructure model. Your team keeps working the way it works today. The shift is in what happens underneath, not what it sees at the query level.

Distinguishing Lakebase from the Lakehouse

Databricks designed the lakehouse and lakebase to work together, not to compete. The distinction between them is about workload type.

The lakehouse handles analytical workloads: batch processing, machine learning (ML), business intelligence (BI), and historical analysis. The lakehouse optimizes for large reads across large datasets. It handles terabytes of data efficiently. It doesn't optimize for the fast individual reads and writes that applications and AI agents need.

The lakebase handles operational workloads: the reads and writes that power live applications, real-time decisioning, and AI agents working with current data. The lakebase optimizes for low-latency access to individual records, not for scanning billions of rows.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Here's what makes a lakebase different from simply running a Postgres database alongside your lakehouse. The lakebase sits on the same storage layer as the lakehouse. Data written to a lakebase is visible to your analytics workloads in near real-time. Data from the lakehouse is immediately available to your applications through the lakebase — no Extract, Transform, Load (ETL) pipeline, no sync job, no lag between what your app writes and what your analysts see.


TIP

If your team has asked how to get fresher data into analytics or how to push analytical insights back into applications, a lakebase is the architectural answer. It removes the boundary between operational and analytical data without requiring either team to change how they work.

Building on Postgres and open standards

The lakebase builds on Postgres for two distinct reasons: ubiquity and openness. Postgres leads the Stack Overflow developer survey as the most widely used database in the developer community. It has a deep ecosystem of libraries, frameworks, object-relational mappers (ORMs), extensions, and tools that developers already know.


TIP

Choosing Postgres as the foundation means your team doesn't need to learn a new query language or replace its existing tooling. They connect to the lakebase the same way they'd connect to any Postgres instance because from the application's perspective, it is one.

This matters especially for AI development. Every major AI framework, including LangChain, LlamaIndex, and the major cloud AI software development kits (SDKs), has built-in Postgres support. Your AI engineers don't need a new database integration to work with the lakebase. They use the same connection pattern they already use for any Postgres database. The cloud-native architecture runs in the background without any additional configuration.

Open standards matter for a separate reason: Your data stays yours. In a traditional managed database, migrating away from the vendor means a complex extraction process, often involving proprietary formats and significant engineering time. With a lakebase, your data lives in open formats in object storage. Another system can read it. Another tool can access it. You're not locked into a format controlled by a single vendor.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.



The lakebase supports the pgvector extension for vector similarity search. That means you can run AI-powered semantic search directly against your operational data without a separate vector database. It also supports PostGIS for geospatial queries. Because a lakebase is Postgres, any standard Postgres extension that your team relies on works the same way it does today.

Mapping the Full Architecture

With the design principles in place, this section explains how the full lakebase architecture fits together: two layers, each with a distinct responsibility, each scaling independently.

Identifying the core layers

Lakebase architecture has two primary layers: compute and storage. A governance and integration layer also connects everything to the rest of the Databricks Platform. The following explains each layer:

- **Object storage is the foundation:** All data lives here, in open formats, in the same cloud object storage your organization already uses. It persists when compute stops, scales as your data grows and is readable by other systems on the platform.
- **Serverless compute is the database engine:** This is the Postgres layer that executes queries and handles reads and writes. It scales up and down independently of storage. Multiple compute instances can run against the same storage layer at the same time. That is how the lakebase supports parallel workloads without duplicating data.
- **Governance and integration connect the lakebase to the lakehouse.** A unified governance layer manages access control, auditing, and data lineage across both sides. The same rules apply whether the data is serving an application, feeding a dashboard, or training an AI model. You have one set of permissions, one audit trail, one source of truth for who can see what.

Each layer handles one job. Storage holds data. Compute processes it. Governance secures and connects it. Because the layers

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

are separate, each one scales on its own terms, and a change in one doesn't force a change in the others.

Seeing how each layer scales

The layered design means you pay for what each layer actually uses, independent of the others. Here's what that looks like in practice.

Storage scales with your data. The storage bill reflects how much data you have, nothing more. A traffic spike doesn't push up storage costs. An analytical query that scans terabytes doesn't change what you pay to store your operational data. The two are on separate meters.

Compute scales with demand. A surge in application traffic adds compute automatically and releases it when the surge passes. A batch of AI agents running parallel experiments each gets its own isolated compute and its own isolated view of the data. Nothing is copied. Each environment is created instantly. When the agents finish, their environments disappear. The underlying data in storage is unchanged.

The governance layer remains consistent regardless of how the other layers are configured. Access controls apply whether one compute instance is running or a hundred. Audit logs capture activity across all of them. Data lineage tracks how data moves through the system at any scale.

For teams with regulatory requirements around data access and retention, this reduces audit complexity. Governance in traditional split-stack architectures covers each system separately: one set of controls for the operational database, another for the analytics platform. With lakebase, the same policies apply across both, so a permission you set for analytical data carries through automatically to operational data. An audit request that used to require pulling logs from two systems now comes from a single location.


TIP

With a lakebase, the database no longer forces you to overprovision a single resource because another resource needs more capacity. You pay for storage based on data volume. You pay for compute based on actual usage. The two bills reflect two different things.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

- Running serverless compute
- Building from a unified data foundation
- Delivering reliability, consistency, and performance

Chapter 3
Examining How Databricks Lakebase Works Under the Hood

Running a separate database for applications and another system for analytics costs you every day, in engineering time, stale data, and infrastructure you didn't need to build. The lakebase architecture eliminates that split. This chapter shows how Databricks Lakebase scales compute automatically, keeps data current across your platform without custom pipelines, and delivers production-grade reliability.

Running Serverless Compute for Transactional Workloads

The compute layer in Databricks Lakebase is responsible for executing queries and handling transactions. It runs Postgres and scales automatically. The key idea is simple: Compute and data are no longer tied together. Because your data lives in the storage layer rather than inside the compute node itself, the compute layer can start, stop, scale up, and scale down to zero without putting your data at risk.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Scaling compute elastically, including to zero

With traditional databases, compute is locked in. *Scaling up* means moving to a larger database and migrating your data, which takes planning, downtime windows, and careful coordination. Scaling down carries the same cost and risk. Most teams avoid scaling down at all, which means they pay for capacity they don't need.

Databricks Lakebase compute works differently because the compute node is stateless. It reads from and writes to storage over the network, and it can be replaced or resized at any time without touching the data. Resizing doesn't move data; it swaps the compute engine.

When no queries are running and no connections are active, the compute layer shuts down entirely, and you pay nothing to run it. The data stays in the storage layer, intact and immediately accessible. When the next query arrives, compute starts up in hundreds of milliseconds and attaches to the existing storage.

Here's what that means in practice. Your development environment runs a small compute server during business hours. At night, it scales to zero. When your first developer connects in the morning, it restarts in seconds. Your production workload scales up automatically during a traffic spike and releases that capacity when the spike passes. The result is automatic scaling with no idle capacity running up your bill.

This pattern also fits how artificial intelligence (AI) agents work. An agent may spin up a database, run a few queries, and never need it again, paying nothing in between.


TIP

On Lakebase, scale-to-zero is disabled by default on production branches, so compute stays continuously available. Enable it on development, staging, and testing branches to drive idle costs to zero. For agentic workloads where each agent spins up its own database, scale-to-zero turns idle environments into zero-cost environments, you pay only for the storage they use.

Protecting data integrity

If the compute layer holds no data, how does the system guarantee that writes are durable? A network failure or a compute crash mid-write results in the transaction being lost entirely.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Databricks Lakebase handles this through a *write-ahead log* (WAL). The WAL is a standard Postgres durability mechanism. Before a transaction completes, a record of the change goes to a set of dedicated storage nodes called *Safekeepers* (part of the Neon architecture underlying Lakebase). A transaction is confirmed only after a majority of Safekeepers have successfully stored it. The compute node never reports success until durability is guaranteed in storage.

The practical result: A compute node can crash at any moment, and no committed transaction is lost. When compute restarts, it reads the WAL from the Safekeepers to reconstruct the current state. The application sees no data loss. The restart takes seconds.



REMEMBER

Data integrity in Lakebase doesn't depend on the compute node staying healthy. It depends on the storage layer, which runs independently of the compute layer. This is why you can resize, restart, or replace a compute instance without performing a backup-and-restore.

Building on a Unified Data Foundation

The storage layer in Lakebase not only holds your operational data, but also it connects directly to the same storage infrastructure used by the rest of the Databricks Platform. That connection eliminates the split between operational data and the lakehouse.

Sharing one store for apps and analytics

In a split-stack architecture, operational data and analytical data live in separate systems. When something changes in your application database, that change has to travel through a pipeline before your analytics platform sees it. Depending on your pipeline setup, that delay may be minutes, hours, or more.

With lakebase, both your application and your analytics platform read from the same storage layer. Your application writes to lakebase. That data sits in the same object storage foundation as your lakehouse tables. Your analytical workloads, machine learning (ML) pipelines, and business intelligence (BI) tools can read from that storage layer directly without waiting for a pipeline to run.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

For example, a customer support application updates a ticket status in lakebase. A data science team running a model to predict churn reads that status as part of their feature set. In a traditional architecture, the data science team waits for the nightly sync. With lakebase, they see the current state because both systems read from the same place.


REMEMBER

The shared storage layer isn't just a performance optimization. It's an architectural guarantee. Operational and analytical data don't diverge because they live in the same place. There's no version of the data in the app and a different version in the warehouse. There's a single version, and every system that needs it reads from it.

Eliminating the data sync problem

Most of the engineering complexity in a split-stack architecture comes from synchronization: keeping the operational database and the lakehouse consistent. That work takes time to build, breaks in ways that are hard to detect, and requires constant monitoring. Databricks Lakebase replaces custom pipelines with two native capabilities, one per direction.

To bring lakehouse data into Lakebase, use synced tables. A *synced table* is a Postgres table in Lakebase that mirrors a Unity Catalog managed table from your lakehouse. You pick the sync mode based on freshness and cost: Snapshot for periodic refreshes, Triggered for scheduled incremental updates, or Continuous for streaming. If your app needs data derived from historical analysis, such as a customer's lifetime spend or a product's average rating, a synced table puts it in the operational layer without a custom integration.

To send Lakebase changes back to the lakehouse, use Lakebase CDF. It captures every insert, update, and delete from your Postgres table and writes them to Unity Catalog managed tables automatically. Analytics and ML workloads read from those tables directly, with no additional pipelines required.


TIP

Synced tables and Lakebase CDF each flow in one direction. Use synced tables to pull lakehouse data into Lakebase for application reads, and Lakebase CDF to stream Lakebase changes out for analytics. For both directions on the same data, set up one of each.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.



TECHNICAL STUFF

Lakebase parallelizes bulk writes from the lakehouse by separating storage and compute, bypassing the standard Postgres bottleneck without affecting operational workloads. In the other direction, Lakebase Change Data Feed (CDF) uses change data captures against the Postgres write-ahead log to stream changes to Delta Lake or Apache Iceberg tables with sub-second latency.

Governing data with Unity Catalog

Unity Catalog is the governance layer for the entire Databricks Platform. It manages access control, auditing, data lineage, and discovery across all data assets, whether they live in the lakehouse, Databricks Lakehouse, or Lakebase.

For Lakebase specifically, Unity Catalog gives you a single place to see and manage permissions across your operational and analytical data. You don't need a separate governance tool for the database. Each system enforces its own permissions, but Unity Catalog provides the unified view, the audit trail, and the lineage that show how those permissions tie together.

This matters for any team that needs enterprise-grade controls: whether you're in a regulated industry with strict compliance requirements, a retailer protecting customer data, or a ticketing platform managing access for thousands of users. The audit trail of who accessed what data, when, and from which system comes from a single place. Data lineage tracking shows how a record moved from the source system into downstream models and reports.



TIP

Unity Catalog reflects the same permissions across access paths: a Databricks Lakehouse query, a Lakebase Postgres connection, and a Databricks App Application Programming Interface (API) call all enforce one consistent rule set.



TIP

If your organization manages data governance in a separate tool, verify compatibility before assuming Unity Catalog will replace it entirely. Unity Catalog covers the Databricks Platform comprehensively, but your broader data estate may include systems beyond it. Unity Catalog can connect to external catalogs through Lakehouse Federation. (See Chapter 5 for more details.)

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Delivering Reliability, Consistency, and Performance

Three questions come up consistently when technical teams evaluate a new database: What happens when something fails, what does the system guarantee about consistency, and how does performance compare to what you already run? This section gives you the direct answers to all three.

Handling failure without interruption

The most common failure mode is a compute process crashing. It has no impact on data durability and minimal impact on availability. When a compute node fails, Lakebase starts a replacement and attaches it to the existing storage. From the application's perspective, this looks like a brief connection interruption followed by an automatic reconnect. No data is lost because committed transactions are already durable in the storage layer. Recovery is automatic.

Point-in-time recovery adds a critical second layer of protection for your data. Lakebase automatically maintains a configurable point-in-time restore window. If a bad deployment or a data error affects your production database, you can restore to any moment within that window, not just to the nearest hourly backup. Restoration creates a new branch from that past state, so your original database stays intact while you investigate or cut over.

Branching production data without copying it

Branching gives you a full, isolated copy of your database in seconds, without duplicating the underlying data. Lakebase uses copy-on-write at the storage layer, so a branch starts as a pointer to the parent's data and only stores the differences as you make changes. Creating a branch is near-instant, and the cost stays low because you pay only for the changes, not the full dataset.

This capability changes how teams work with production data. Engineers can run a schema migration against a branch, verify the result, and discard the branch if something goes wrong. Production is untouched. Continuous integration and continuous

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

delivery/deployment (CI/CD) pipelines can spin up a fresh branch for every pull request, run the test suite against real data, and tear it down when the build completes. Performance testing runs against actual production data and query patterns instead of a synthetic staging environment that drifts from reality.

Branching also gives AI agents a safe place to work. An agent can spin up a branch, run experiments against real production data, and explore changes without any risk to the live database. If the agent's work pans out, you promote the branch. If not, you discard it. Production never sees the test runs.


TIP

Build the branch-before-you-deploy pattern into your team's workflow. Before any schema migration or significant data change, branch the database, apply the change to the branch, and verify. If something goes wrong, you discard the branch at no cost. Your production database is untouched.

Setting realistic performance expectations

Lakebase targets sub-10-millisecond latency for standard transactional operations and supports more than 10,000 concurrent transactions. Those numbers reflect production workloads, not benchmarks run under ideal conditions.

Lakebase adds a Local File Cache (LFC) that sits between the compute and storage layers. Frequently accessed data stays in the LFC at low latency. Data outside the cache is pulled from object storage, which adds a small amount of latency.


TECHNICAL STUFF

LFC stores Postgres page files on the compute node's local SSD. It operates between the standard Postgres shared buffer cache and the object storage layer. The LFC size is configurable per compute instance. Larger instances have greater LFC capacity, which is why lakebase recommends sizing compute based on your working set instead of just your peak query load.

For most production workloads, the working set fits comfortably in the cache, and performance is comparable to a traditional Postgres deployment on fast local storage. Workloads that require full sequential scans or access highly random data with no locality see more object storage fetches and benefit from sizing the LFC accordingly.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Lakebase doesn't ask you to trade performance for the architectural benefits. The cache layer is designed to deliver Postgres-level performance for the transactional access patterns that operational applications actually use.



REMEMBER

Performance testing against your actual workload matters more than benchmark numbers. Lakebase provides branching specifically to support this: Branch your production database, run your performance tests against the branch, and measure against real data and real query patterns. You don't need a separate staging environment loaded with synthetic data to get a reliable read on production performance.

Every design decision in the lakebase traces back to one principle: Separate the layers so each can do its job without depending on the others. That's what makes elastic compute possible, what keeps data consistent across applications and analytics, and what lets governance apply once instead of everywhere separately.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

- Using a lakebase as your system of record
- Supporting AI agents with real-time data
- Eliminating data movement and pipeline complexity
- Accelerating developer productivity

Chapter 4
Building Apps and AI with a Lakebase

A lakebase changes how teams build operational applications and artificial intelligence (AI) workloads. The data that powers your apps lives in the same architecture as the data your analysts query and your AI models train on, so the system of record stays consistent across the stack. Branching gives developers a Git-like workflow for the database itself. Pipelines that previously moved data between operational and analytical systems become unnecessary for most use cases. This chapter shows you how to apply each of these patterns: how to use a lakebase as your system of record, how to support AI agents that need real-time data, how to eliminate the pipeline tax your team has been paying, and how to accelerate developer productivity through branching and unified deployment. (See Chapter 3 for the Databricks Lakebase implementation.)

Using Lakebase as Your System of Record

A system of record is the authoritative source for a given piece of data. Your application reads from it and writes to it. Every downstream system trusts it. A lakebase is designed to fill this role for

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

operational data with the additional property that your analytical platform shares the same foundation instead of waiting for a sync.

Establishing a single source of truth

In a split-stack architecture, the question of which system holds the authoritative version of your data rarely has a clean answer. Your operational database holds the most current state. Your data warehouse holds a copy that was current as of the last pipeline run. Your analytics platform holds a third version shaped by transformation logic applied during ingestion. All three diverge over time, and reconciling them is a recurring engineering cost.

A lakebase removes that divergence. Because the storage layer is shared with the lakehouse, the data your application writes to the lakebase is the same data your analysts query and your AI models train on. There's no copy in transit, no version waiting to be reconciled. For decision-making, this matters. When a sales team runs a report and a customer success team checks an account record at the same time, they see the same underlying data. When an AI model generates a recommendation, and the application displays it to a user, both draw from the same source. The single source of truth isn't a goal to work toward. It's a property of the architecture.



REMEMBER

Lakebase as a system of record doesn't mean every piece of data in your organization lives in the lakebase. It means the operational data that drives your applications and your AI workloads has one authoritative home, and that home connects directly to the rest of your data platform.

Structuring your data model for operational use

A lakebase runs full Postgres, which means your existing data modeling practices apply directly. Foreign keys, indexes, stored procedures, triggers, schemas: All of it works exactly as it does in standard Postgres. If your team already models relational data, there's no new paradigm to learn.

A few patterns work especially well with the lakebase architecture. Use schemas to isolate different application domains. A customer data schema and an order management schema can carry different access policies through your lakehouse governance layer, with

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

no change to the physical data layout. Using branching to test schema migrations before applying them to production reduces deployment risk because the branch shares the storage layer and reflects real production data.

Infrastructure-as-code tooling lets you manage your lakebase deployment alongside your application code. Database configuration and environment-specific settings live in version-controlled bundles. A single deployment command applies the right configuration for each environment.


TIP

If your team uses GitHub or GitLab for code, use the same repositories for your database infrastructure-as-code configurations. Your database goes through the same review and approval process as your application code. Environment drift becomes visible and reversible. (See Chapter 5 for the specific tooling in Databricks Lakebase.)

Supporting AI Agents with Real-Time Data

AI agents make different demands on a database than human users do. They operate at machine speed. They create and discard isolated environments without human involvement. They need data that reflects the current state of the world, not data from yesterday's pipeline run. Lakebase is designed for exactly this pattern.

Defining what AI agents need from a database

An AI agent working through a task needs three things from its database infrastructure.

- **Fast access to current data:** A customer service agent who retrieves outdated account information gives bad answers. A supply chain optimization agent that reads inventory data from hours ago makes bad recommendations. Freshness isn't a nice-to-have option for AI agents; it determines whether the agent's output is useful.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

- **To be isolated:** Agents often explore multiple hypotheses in parallel, running experiments against the data to determine which path to pursue. Those experiments need their own isolated environment, so they don't interfere with each other or with production data. In a traditional database, creating that isolation requires copying the data, which is slow and expensive at scale.
- **To be provisioned at machine speed:** Agents spin up environments, use them, and discard them, often within a single workflow execution. The database infrastructure has to keep up with that speed. A provisioning process that takes minutes is a bottleneck that limits what agents can do.



REMEMBER

A lakebase addresses all three requirements through a single set of architectural properties: fresh data via the shared storage layer, isolation via instant copy-on-write branching, and machine-speed provisioning via stateless compute. These aren't separate features. They follow from the same underlying design.

Enabling real-time decisioning without data lag

Real-time decisioning means making decisions based on the current state of your data, not a cached or delayed version. For AI agents and for operational applications, data lag introduces errors: The model recommends a product that's out of stock, the application displays a balance that doesn't reflect a recent transaction, and the agent takes an action based on a record that has already changed.

Lakebase eliminates data lag between operational and analytical layers by design. A change written to the lakebase is immediately visible to any analytics or AI workload that reads from the shared storage layer. A feature store built on a lakebase serves machine learning (ML) models the freshest available features without a separate sync step. An AI agent that reads account history and writes interaction logs to the same lakebase database works with a consistent, current view throughout its execution.

Persistent memory for AI agents is a distinct pattern worth noting. Agents that maintain context across sessions — storing what they've learned, what decisions they've made, and what state they're tracking — need a database that's both fast and

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

consistent with the broader data platform. A lakebase serves as that persistent memory layer without requiring a separate infrastructure component.



TIP

If your AI agents currently rely on an in-memory cache or a standalone vector database for session state, consider consolidating that state into the lakebase. You get the same low-latency access, plus the governance, auditability, and lakehouse integration that come with the architecture.

Eliminating Data Movement and Pipeline Complexity

Most of the complexity in a modern data stack comes from moving data between systems. Every pipeline you build is a liability: something to monitor, something to debug when it fails, something to update when schemas change. A lakebase is designed to minimize that movement.

Calculating the pipeline tax

The pipeline tax is the engineering time and infrastructure cost your organization pays to keep data synchronized across separate systems. It includes the time to build each pipeline, the monitoring to detect failures, the debugging when something breaks, and the storage and compute cost of running multiple copies of the same data. The sizing exercise has four components:

- Inventory the pipelines currently running between your operational database and your analytical platform.
- Estimate the engineering hours spent building and maintaining them over the last year.
- Add the cost of the compute running those pipelines continuously.
- Add the storage cost of the duplicated data.

For most organizations with a mature data stack, this number runs into the hundreds of thousands of dollars annually. For larger organizations, it reaches into the millions. But the indirect cost is often larger. Pipeline maintenance time is lost product

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

development time. Every hour a pipeline sits broken, your analysts work with stale data, and your AI systems make decisions on outdated inputs.


REMEMBER

When evaluating the cost of moving to a lakebase, don't just compare database costs. Compare total stack costs: the database, the pipeline infrastructure, the engineering time required to maintain both, and the cost of the data quality issues introduced by the pipelines. The comparison changes significantly when you count the full picture.

Simplifying your stack

With a lakebase, the pipelines that move data between your operational database and your analytical platform become unnecessary for a large portion of your use cases. Operational data written to the lakebase is immediately available to your analytical workloads through the shared storage layer. You don't build a pipeline to move it. You don't maintain a sync process to keep it current. You don't debug failures when the sync breaks. The data is simply there.

The full application stack on a unified platform includes a lakebase for the database, an application runtime with built-in authentication and permissions, and the governance layer that comes with a lakehouse. All of these components live on the same platform, with no cross-environment networking to configure and no separate authentication systems to maintain. (See Chapter 5 for how Databricks Apps and Unity Catalog deliver this stack.)

Developers building on this stack go from a prototype running in a development environment to a production deployment without having to move between platforms. The same workspace holds the development branch, the staging branch, and the production database. Promotion between environments is a configuration change, not a migration.


REMEMBER

Simplifying your stack with a unified foundation isn't just about developer experience. Fewer systems mean fewer failure points, fewer security surfaces to manage, and a shorter path from a detected issue to a resolution. The operational benefits compound over time as your stack grows.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Syncing changes automatically

For data that genuinely needs to flow between the lakebase and external systems, lakebase architecture supports Change Data Capture (CDC) natively through the Postgres write-ahead log. CDC tracks changes at the row level and streams them to downstream consumers in near real time, with minimal overhead on the operational workload. The most common use case is keeping a downstream system current, whether that's a reporting tool, an external partner system, or a legacy application still in transition. (See Chapter 3 for how Databricks Lakebase implements synchronizing data with the lakehouse.)

Accelerating Developer Productivity

Developer productivity isn't a soft metric. It determines how quickly your organization ships features, how quickly it responds to production issues, and how much engineering capacity goes toward building products rather than managing infrastructure. This section explores each area.

Adopting lakebase without retraining your team

A lakebase speaks standard Postgres. Every driver, object-relational mapping (ORM), connection library, and query tool that works with Postgres works with a lakebase without modification. A developer connecting to the lakebase for the first time uses the same connection string format, the same authentication patterns, and the same SQL they already know.

Adoption doesn't require a training program or a migration of institutional knowledge. Your team's existing Postgres expertise transfers directly. The productivity gain comes from what the infrastructure does differently, not from learning a new interface. Engineers spend their time building features, not learning a new database paradigm.

The ideal lakebase implementation also provides a native Postgres query editor and schema browser in the same workspace as the rest of your data infrastructure, eliminating the need for a separate database management tool. (See Chapter 5 for what Databricks Lakebase provides.)

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.



TIP

Start your lakebase adoption with a new project rather than migrating an existing production database. Pick a new internal tool, a new microservice, or a new AI agent workflow. Let your team become familiar with the branching workflow, autoscaling behavior, and application runtime integration before they take on a migration of existing operational data.

Branching databases the way you branch code

Database branching in a lakebase works the same way as code branching in Git. You create a branch from the current state of a database, make changes on the branch, and either merge those changes back or discard the branch. The branch is an isolated copy from the moment it's created. Changes on the branch don't affect the source database, and changes to the source don't affect the branch.

Here's what this enables across the development lifecycle. A developer starts work on a feature by creating a branch from the production database. The branch contains real production data because it shares the same storage layer. No synthetic test data, no hope that the test environment resembles production closely enough. Real data, real schema, real behavior.

When the feature is ready for review, a continuous integration and continuous delivery/deployment (CI/CD) pipeline spins up a fresh branch for each pull request, runs the full test suite against real data, and tears the branch down when the build completes. Each pull request gets its own isolated database, so concurrent reviews don't interfere with each other.

Staging branches hold longer-lived integration work where multiple features come together before release. Promoting a change from staging to production happens through the same branch-merge mechanics that developers already use for code, with no separate database migration step. Teams that adopt this pattern report reduced deployment anxiety around database changes because every promotion has been tested against real data, real schema, and real query patterns. The branch-test-promote workflow catches issues that would have reached production in a traditional deployment process, without adding days of setup time to each release cycle.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.



REMEMBER

Branching in lakebase is a metadata operation. You don't copy data when you create a branch. The branch and the source database share the same underlying storage until changes diverge. This means creating a branch of a multi-terabyte production database takes the same amount of time as creating a branch of a small development database: seconds.

Getting to production faster

The traditional path from a working prototype to a production application involves a sequence of steps that each add time: provisioning a production database instance, configuring networking and security, setting up monitoring and backups, writing authentication flows, mapping permissions across the application layers, and coordinating a deployment across multiple environments.

With a lakebase running on a unified data platform, most of those steps are handled by platform defaults. Authentication uses the platform's identity system that your team already operates. You apply permissions through existing governance policies. Backups and point-in-time recovery are enabled by default. Networking runs within the platform without separate configuration. The path from prototype to production narrows to configuring your deployment bundle for the production environment and deploying. (See Chapter 5 for the specific Databricks implementation.)

Proving the Model with Customer Results

The architectural arguments for a lakebase are straightforward. But architecture arguments are easier to make than results are to achieve. Three organizations that adopted the lakebase architecture in production achieved the following:

- A maritime shipping company operating more than 200 vessels built its internal operating portal and DNA Port on a lakebase to serve commercial, technical, and finance teams. Its previous stack centered on SQL Server, SQL Server Integration Services (SSIS), and Power Business Intelligence (BI), with external Postgres providers that didn't integrate natively with its analytics platform. Moving to a lakebase as the transactional engine, a lakehouse as the analytical system of record, cut app delivery from

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

two months to five days, a reduction of more than 96 percent.

- A healthcare revenue cycle company managing operations for more than 30 health systems and processing more than 25 billion transactions across 2 petabytes (PB) of provider, payer, and clinical data ran its legacy environment across 15 individual SQL Server databases. The silos blocked cross-client insights and slowed AI development in an industry where providers operate on 1 percent margins. Consolidating onto a lakebase eliminated brittle pipelines and enabled self-service teams to materialize views directly from harmonized lakehouse tables. The results were a 20 percent efficiency gain, a 3 to 5 percent improvement in net revenue yield for customers, while powering agentic AI workflows for denial prevention, accounts receivable (AR) follow-up, and payer-behavior prediction at scale.
- A major European airline replaced a .NET desktop application running on one of Europe's largest legacy SQL Server environments with a web-based revenue management hub built on a lakebase. The modernization cut its development cycle from nine months to four months and consolidated more than 100 Git repositories into two.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

- Connecting Lakebase to the broader Databricks ecosystem
- Building and deploying apps with Databricks Apps and developer tools
- Using Claude Code, Cursor, and popular SDKs
- Simplifying your stack by running apps and analytics on one platform

Chapter 5
Tapping into the Databricks Platform

Databricks Lakebase runs within the Databricks Platform, meaning every tool, service, and governance layer in that platform connects to it directly. For organizations already using Databricks for analytics, machine learning (ML), or data engineering, Lakebase extends what that investment can do rather than adding a separate system to manage. This chapter covers the specific Databricks Platform capabilities that connect to Lakebase.

Connecting Lakebase to the Databricks Ecosystem

Databricks Lakehouse connects to Lakebase via Unity Catalog, so your SQL analysts can query operational data alongside their lakehouse tables without switching tools or requesting a data export, and without impacting operational database performance. A Databricks Lakehouse warehouse runs the same queries against Lakebase tables as it does against analytical tables,

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

without needing special configurations or a separate connection to manage.

ML workloads benefit from the same connection. A model training pipeline that reads features from the lakehouse and a real-time inference pipeline that reads from Lakebase share the same governance layer and the same lineage tracking. When a feature changes in the operational database, the downstream model quickly reflects that change, closing the freshness gap that creates inconsistency between training and serving data.

Databricks Agent Platform connect to Lakebase for persistent agent state, tool memory, and real-time data retrieval. An agent built on Databricks reads from and writes to Lakebase using standard PostgreSQL connections, with Lakebase enforcing the access policies that Unity Catalog reflects, so every interaction stays governed.



REMEMBER

Unity Catalog is the connective tissue, giving you one place to see and manage permissions across Lakebase, the lakehouse, Databricks Lakehouse, and your ML workloads. Each system enforces those permissions itself, but the rules and visibility live in Unity Catalog.

Building Apps with Databricks Apps and Developer Tools

Databricks Apps is the platform's application runtime layer. It provides a serverless environment for deploying web applications with built-in authentication, fine-grained permissions, and direct access to your Databricks data assets. Combined with Lakebase, it forms a complete full-stack deployment target inside the platform.

Here's what the development path looks like. A developer builds an application with the tools of their choice, including artificial intelligence (AI) coding assistants, that reads from and writes to a Lakebase database. The application frontend runs as a Databricks App. Authentication flows through the Databricks identity system, which the team already uses. Permissions for the application's database access are managed through Unity Catalog, the same way they apply to every other data access in the platform.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

The result is a prototype-to-production path that stays entirely within the infrastructure perimeter of the platform. A developer working in a Databricks workspace can go from a working local prototype to a deployed, governed, production application without provisioning external infrastructure, configuring separate authentication, or setting up a standalone database.

Declarative Automation Bundles (DABs) manage infrastructure-as-code for your Lakebase deployment. Your database configuration and environment-specific settings live in a version-controlled bundle alongside your application code. A single deployment command applies the right configuration for each environment.


TIP

Databricks Apps supports common Python frameworks like Streamlit, Dash, and FastAPI, and Node.js frameworks like React and Express. If your team already builds internal tools or data applications in any of these, the migration path is simple. The application code stays the same. The deployment and infrastructure model shifts to the platform.

Connecting with AI Coding Assistants and Popular SDKs

Because Lakebase runs Postgres, every tool that connects to Postgres connects to Lakebase. That covers the developer toolchain that AI-assisted development teams use today, including AI coding tools, integrated development environment (IDE) extensions, and language software development kits (SDKs) across the most common development environments.

AI coding assistants can connect to Lakebase using a standard Postgres connection string. A developer using a coding agent to build against a database schema, write migrations, or generate application code works against Lakebase the same way they'd work against any Postgres instance. The tool sees a standard Postgres interface and operates accordingly. Databricks offers AppKit, a JS toolkit that handles the application to the Lakebase connection path to further simplify integration.

Language SDKs follow the same logic. Standard Postgres drivers and object-relational mappers (ORMs) work without modification across Python, Node.js, Java, Go, and other common environments. Because Lakebase is standard Postgres, your team doesn't

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

adopt a new SDK, learn a new connection model, or update existing application code to switch the backend to Lakebase.

Simplifying Your Stack with a Unified Foundation

The compounding benefit of running applications and analytics on a single platform becomes most evident in operational complexity. Every tool you remove from your stack is a tool you no longer need to monitor, license, update, or integrate. Every system boundary you eliminate is a failure point and a security surface that disappears.

A team running Lakebase alongside Databricks Lakehouse, Databricks Apps, and Agent Bricks operates on a single platform with a single governance model, identity system, billing relationship, and support contract. Compare that to a split-stack team managing a separate operational database, a separate analytics warehouse, a separate pipeline infrastructure, and separate application hosting, each with its own monitoring, failure modes, and operational overhead.

The speed benefit compounds just as quickly. When the operational database, the analytics platform, and the application runtime all live in the same place, the coordination work between teams shrinks. A data engineer making a schema change in the lakehouse sees that the change propagates to the Lakebase Change Data Feed (CDF) Table automatically. An application developer building against the updated schema doesn't wait for a data team to publish a new export. An AI agent running in a Databricks workflow reads from the current state of both the operational and analytical layers in the same execution context.


TIP

If your organization uses Databricks for analytics today and runs a separate database for your applications, Lakebase is the path to consolidating those systems into a single platform. You don't need to migrate your analytical workloads. You add Lakebase to the platform you already run and move your application database workloads there over time.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

- Breaking under modern workloads
- Avoiding lock-in
- Counting the cost savings

Chapter 6
Ten Reasons Why You Need a Lakebase

Most organizations are running database infrastructure designed before agents, cloud-native compute, and real-time analytics existed. The following list gives you the ten reasons why a lakebase is needed:

- **Eliminate the split stack and its hidden costs.** Running a separate online transaction processing (OLTP) database for applications and an online analytical processing (OLAP) platform for analytics forces your team to build the pipelines that connect them. Every pipeline is a maintenance liability. A lakebase consolidates both workloads onto a single foundation and eliminates the overhead of keeping two systems in sync. See Chapter 1 for more information.
- **Give your agents a database built for them.** A lakebase handles instant branching and elastic provisioning by design, so artificial intelligence (AI) agents get the isolated environments they need without a human in the loop. Check out Chapter 4 for information about supporting agents.
- **Pay for compute only when you use it.** A lakebase separates the compute and storage. Compute scales to zero when nothing is running and starts up automatically when demand returns.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

- **Ship faster with a leaner developer experience.** Lakebase is Postgres, so your team connects by using tools they already know. Instant database branching lets developers test against production-state data without duplicating storage, which accelerates developer productivity (see Chapter 4 for more information).
- **Reduce your total cost of ownership.** The true cost of a database includes the engineering time to maintain pipelines, the compute running duplicate infrastructure, the storage holding multiple copies of the same data, and the costs of failures introduced by pipelines. Lakebase eliminates most of that overhead. Flip back to Chapter 1 for more information.
- **Keep data fresh without custom pipelines.** Lakebase eliminates the need to build Extract, Transform, Load (ETL) jobs. Sync Tables automatically keep lakehouse and operational data in sync.
- **Build a future-proof data foundation.** Lakebase is built around the architecture that replaces separate systems for operational and analytical work. Chapter 2 covers the full architecture.
- **Govern your entire data estate with Unity Catalog.** Unity Catalog applies the same access controls, audit logging, and data lineage tracking to Databricks Lakebase, managed Delta Lake and Apache Iceberg tables, Databricks Lakehouse, and machine learning (ML) workloads. You define policies once, and they apply everywhere. For regulated industries, this means a single governance model covers both operational and analytical data.
- **Avoid lock-in with Postgres and open standards.** Lakebase runs Postgres and stores data in open formats in cloud object storage. Your existing drivers, object-relational mappers (ORMs), tools, and query libraries connect to it without modification. Your data resides in formats that other systems can read.
- **Simplify your architecture dramatically.** Lakebase, Databricks Apps, and the Databricks Platform together replace the fragmented combination of a standalone operational database, a separate analytics warehouse, custom pipeline infrastructure, and external application hosting. Head to Chapter 5 for more about simplifying your stack.

These materials are © 2026 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

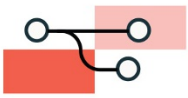
databricks

Scan to learn more

The database for AI agents and apps

Lakebase is a fully managed, serverless Postgres database integrated with the lakehouse. Now teams can do the real work they love, rather than waiting on another ticket or managing database infrastructure. With Lakebase, you can instantly spin up databases for every idea, agent or app, and scale them to zero when they're idle.

Lakebase



Database as code

Branch fast, build even faster



Lower TCO

Scale elastically, and only pay for what you use



Integrated platform

Simplify your stack with centralized governance and no ETL

Learn more at databricks.com/product/lakebase

Build agents and apps on Lakebase

AI agents now create the majority of new databases, and the split stack between operational and analytical systems has become a tax your team pays every day. Provisioning takes too long, testing against production data is risky, and pipelines break. This book introduces Lakebase, a serverless Postgres database built on lakehouse architecture. You discover how to unify apps, analytics, and AI on one foundation, spin up databases almost instantly, branch them like code, and scale to zero when you're done.

Inside...

- Why AI agents break legacy databases
- Uncouple compute from storage
- Scale to zero and branch instantly
- Sync data without custom pipelines
- Build on Postgres and open standards
- Power AI agents with fresh data
- Cut TCO and avoid vendor lock-in

databricks

Stephanie Diamond is a marketing innovator and author with more than 25 years of marketing experience. She writes retail and custom eBooks for Fortune 500 companies and is known for transforming complex ideas into engaging narratives.

Go to Dummies.com™
for videos, step-by-step photos, how-to articles, or to shop!

ISBN: 978-1-394-43062-8
Not For Resale



for dummies®
A Wiley Brand

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.