■ **Course:** CSCI E-103: Reproducible Machine Learning

■ **Lecture:** Lecture 14 – Databricks Roadmap & Emerging Features

■ **Instructors:** Anindita Mahapatra & Eric Gieseke

■ **Objective:** Explore Databricks' latest platform features including Lakebase, AI/BI, Agent Bricks, MCP integration, and review key concepts from Quiz 2 and Assignment 3

Contents

1 Introduction: The Databricks Data Intelligence Platform

Lecture Overview

This lecture explores the cutting-edge features in the Databricks roadmap—technologies that are either in public preview or will be released soon. These innovations represent the platform's evolution toward a comprehensive **Data Intelligence Platform** that handles everything from data storage to AI agent deployment.

Key Learning Objectives:

- Understand the new Databricks One unified experience
- Learn about Lakebase and OLTP capabilities within the Lakehouse
- Explore Agent Bricks and AI/BI features (Genie, Research Mode)
- Understand Lakehouse Apps for building secure data applications
- Review Quiz 2 concepts (Spark optimization, Delta Lake internals)
- Review Assignment 3 concepts (Streaming, SCD, Kafka/Kinesis)

1.1 The Data Intelligence Platform Architecture

The modern Databricks platform is built on a layered architecture that serves diverse workloads:

Layer	Components	Purpose
Foundation	Cloud Storage	Object storage (S3, ADLS, GCS) for all data
Format Layer	Delta, Iceberg	Open table formats ensuring data reliability and ACID compliance
Governance	Unity Catalog	Centralized governance, security, and lineage tracking
Compute	Serverless SQL, Clusters	Scalable compute for all workload types
Applications	Agent Bricks, DBSQL, Apps	User-facing applications and AI capabilities

Table 1: *Databricks Data Intelligence Platform Layers*

Key Information

Key Investment Areas in Databricks:

1. **Data Intelligence Capabilities**: Making the platform smarter so data professionals do less manual work
2. **Built-in Governance & Security**: Data sharing tools with ABAC and fine-grained access control
3. **Tools to Build AI Agents**: Agent Bricks, MCP servers, model serving
4. **Data Engineering & Migration**: LakeFlow, connectors, zero-copy ETL

2 Databricks One: The Unified Experience

Definition:

Databricks One **Databricks One** is a new unified portal experience designed for business users. It provides account-level access to dashboards, Genie spaces, and apps across all workspaces in a single, intuitive interface.

2.1 Portal Switcher: Three Experiences

When you log into Databricks, you'll see a portal switcher with three options:

Portal	Target Users	Purpose
Lakehouse	Data Engineers, Data Scientists	Full workspace with SQL, ML, Data Engineering features. This is the traditional Databricks experience.
Databricks One	Business Users, Analysts	Simplified view showing dashboards, Genie, and apps across all workspaces. No technical complexity.
Lakebase Postgres	Application Developers	OLTP database interface for transactional workloads. PostgreSQL-compatible.

Table 2: *Databricks Portal Experiences*

2.2 Databricks One Features

- **Account-Level Access**: View assets from all workspaces in one place
- **Vanity URL**: Companies can use their own domain (e.g., `analytics.yourcompany.com`)
- **Domain Integration**: Organize content by business domain (Marketing, Finance, Tech)
- **Intuitive Navigation**: Search and discover dashboards, Genie spaces, and apps easily

Example:

Why Databricks One Matters Imagine a large enterprise with 50+ workspaces. Without Databricks One, a marketing analyst would need to know which specific workspace contains their dashboard. With Databricks One, they simply log in, see all marketing-related assets, and get to work—regardless of which workspace hosts the asset.

3 Lakebase: Bringing OLTP to the Lakehouse

Definition:

Lakebase **Lakebase** is Databricks' managed PostgreSQL implementation that brings **OLTP (Online Transaction Processing)** capabilities to the Lakehouse platform. It enables transactional workloads alongside analytical workloads, all within the same governance framework.

3.1 Understanding OLTP vs OLAP

Aspect	OLTP (Transactional)	OLAP (Analytical)
Purpose	Handle real-time transactions	Analyze historical data
Operations	INSERT, UPDATE, DELETE (many small)	SELECT (few large queries)
Data Volume	Current operational data	Historical aggregated data
Users	Applications, end users	Analysts, data scientists
Examples	Banking transactions, order processing	Sales reports, trend analysis
Traditional Tools	Oracle, SQL Server, PostgreSQL	Data warehouses, Databricks

Table 3: *OLTP vs OLAP Comparison*

3.2 Why Lakebase is Revolutionary

Important:

The Storage-Compute Separation Paradigm Shift Traditional databases (Oracle, SQL Server) have compute and storage tightly coupled. If you buy a database server, it runs 24/7 whether you have one row or one billion rows.

Lakebase changes this fundamentally:

- Storage and compute are completely separated
- Compute can scale up as load increases
- Compute can scale **down to zero** when idle—you pay nothing
- Sub-second startup time means instant readiness when needed

This was possible for data lakes and warehouses, but **never before for transactional databases**. Lakebase is the first to achieve this for OLTP workloads.

3.3 Lakebase Architecture

```
-- Connection string format
psql "postgres://your-lakebase-instance.databricks.com:5432/database_name"

-- Use OAuth token for authentication
```

```
-- Default role is your Databricks username
```

Listing 1: Lakebase PostgreSQL Connection

3.3.1 Key Features

1. **PostgreSQL Compatibility**: Standard PostgreSQL syntax and tools work seamlessly
2. **Version Selection**: Choose your PostgreSQL version (e.g., PostgreSQL 17)
3. **Branch-Based Development**:
   - Production branch (for live data)
   - Development branch (for testing)
   - Git-style workflow: Test safely without touching production
4. **Backup and Recovery**:
   - Scheduled snapshots
   - Point-in-time recovery
   - Instant recovery capabilities
5. **SQL Editor**: Built-in query interface, monitoring, and table management

3.4 Catalog Types in Unity Catalog

When creating a catalog in Unity Catalog, you now have four options:

Catalog Type	Use Case	Description
Standard	Default projects	Typical catalog for tables, views, models
Foreign	Data federation	Connect to external data stores (Snowflake, Redshift, Teradata) without moving data
Shared	Delta Sharing	Zero-copy data sharing with external parties
Lakebase PostgreSQL	OLTP workloads	Connect to your Lakebase databases for transactional access

Table 4: *Unity Catalog Types*

3.5 Bidirectional Sync: Lakehouse ↔ Lakebase

Example:

Data Flow Between Lakehouse and Lakebase **Forward Sync (Lakehouse → Lakebase):**

- Sync a subset of your analytical data to Lakebase
- Enables low-latency reads for applications

- Useful for ML inference, real-time dashboards

Reverse Sync (Lakebase → Lakehouse):

- Business users make small corrections in Lakebase
- Changes sync back to the Lakehouse
- Maintains a single source of truth

```
-- Create table in Unity Catalog
CREATE TABLE catalog.schema.user_segments (
    user_id INT,
    segment STRING,
    engagement STRING
);

-- Sync to Lakebase PostgreSQL
-- (Done through UI or sync configuration)

-- Query in Lakebase
SELECT * FROM default.user_segment_synced
WHERE engagement = 'high';
```

4 AI/BI: Genie and Research Mode

Definition:

AI/BI **AI/BI** is Databricks' AI-powered business intelligence solution consisting of two main components:

1. **Dashboards**: Interactive visualizations with drill-down capabilities
2. **Genie**: Natural language interface for querying data

4.1 Genie: Answering "What" Questions

Genie transforms natural language questions into SQL queries against your structured data:

- **Factual Questions**: "What were total sales last quarter?"
- **SQL Generation**: Automatically generates optimized SQL
- **Custom Instructions**: Add context, join logic, and example queries
- **Consistent Responses**: Trained on your data semantics

4.2 Research Mode: Answering "Why" Questions

Important:

Research Mode vs Regular Genie **Regular Genie**: Answers *what* questions with direct SQL queries.

- "What were sales in Q4?" → Returns a number
- Fast response, factual answer

Research Mode: Answers *why* questions with hypothesis-driven analysis.

- "Why did sales drop in Q4?" → Explores multiple factors
- Uses **Chain of Thought** reasoning
- Takes longer (several minutes) but provides deeper insights
- Shows the reasoning paths explored

Example:

Research Mode in Action **Question:** "What would happen to my portfolio if oil prices increase by 20%?"

Research Mode Process:

1. Identifies relevant data (portfolio holdings, sector exposure)
2. Generates hypotheses about impact paths
3. Explores correlations with oil-sensitive sectors
4. Analyzes historical patterns during price spikes
5. Synthesizes findings into actionable insights

Output: A detailed analysis showing multiple scenarios, not just a simple number.

4.3 Enhanced AI/BI Features

- **Document Upload**: Attach CSVs, Excel files, or PDFs to combine with structured data
- **Drill-Down for Dashboards**: Click on visualizations to explore hierarchies (Region → Country → State → City)
- **Slack/Teams Integration**: Subscribe to dashboard alerts and notifications
- **Multi-Page Subscriptions**: Schedule reports across multiple dashboard pages

5 Agent Bricks and AI Agents

Definition:

Agent Bricks **Agent Bricks** is Databricks' framework for creating AI agents with minimal code. Similar to AutoML, you point it to data and specify what you want—the framework handles the scaffolding, evaluation, and deployment.

5.1 Types of Agents Available

Agent Type	Purpose
Information Extraction	Extract structured data from unstructured documents
Knowledge Assistant	Answer questions using your organization's knowledge base
AI/BI Genie	Natural language querying of structured data
Multi-Agent Supervisor	Coordinate multiple specialized agents
Custom LLM	Build agents using your own fine-tuned models
Custom Agent	Full control over agent logic and behavior

Table 5: *Agent Bricks Agent Types*

5.2 Creating an Agent

Example:

Information Extraction Agent Creating an information extraction agent:

1. **Name**: Give your agent a descriptive name
2. **Data Location**: Point to your documents (PDFs, emails, forms)
3. **Fields to Extract**: Specify what information to pull (company name, address, dates)
4. **Deploy**: Agent is ready in minutes

No coding required—similar to the AutoML experience for machine learning.

5.3 MCP (Model Context Protocol)

Definition:

MCP **Model Context Protocol (MCP)** is an open standard that enables AI agents to connect to external tools, data sources, and services. MCP servers act as bridges between agents and the resources they need.

- **MCP in Marketplace**: Discover and use pre-built MCP servers

- **Governance**: MCP servers are governed by Unity Catalog
- **Custom MCP**: Build your own MCP servers for proprietary tools
- **Serverless GPU**: MCP servers can leverage GPU compute for AI tasks

6 Lakehouse Apps

Definition:

Lakehouse Apps **Lakehouse Apps** allow you to build and deploy web applications directly on the Databricks platform, where your data lives. Apps can read and write data securely, interact with models, and provide user-friendly interfaces for non-technical users.

6.1 Why Apps vs Dashboards?

Aspect	Dashboards	Apps
Interaction	Read-only visualization	Bidirectional (read & write)
Data Modification	Not possible	Users can update data
Use Case	Reporting, monitoring	Data correction, input forms, chatbots
User Experience	Pre-defined views	Custom, interactive UI

Table 6: *Dashboards vs Apps Comparison*

6.2 Supported Frameworks

- **Python**: Streamlit, Dash, Flask, Gradio, Shiny
- **JavaScript**: Node.js, React
- **Templates**: Data apps, chatbots, model serving interfaces

6.3 App Architecture

```python
import streamlit as st
from databricks import sql

# Connect to SQL warehouse
connection = sql.connect(
    server_hostname="your-workspace.databricks.com",
    http_path="/sql/1.0/warehouses/abc123"
)

# Query data
cursor = connection.cursor()
cursor.execute("SELECT * FROM gold.customer_metrics LIMIT 100")
data = cursor.fetchall()

# Display in Streamlit
st.title("Customer Metrics Dashboard")
```

```
st.dataframe(data)

# Allow user to update data
if st.button("Mark as Reviewed"):
    cursor.execute("UPDATE gold.customer_metrics SET reviewed = TRUE")
    st.success("Data updated!")
```

Listing 2: Simple Streamlit App Example

Key Information

Key Benefits of Lakehouse Apps:

- Data stays in place—no copying to external systems
- Security inherited from Unity Catalog
- Horizontal scaling for many users
- Easy deployment from Git or SDK
- Support for vibe coding tools (Claude, Cursor)

7 Data Intelligence Features

7.1 AI Parse Document

```sql
-- Extract structured data from PDFs
SELECT AI_PARSE_DOCUMENT(
    file_path,
    'company_name, address, invoice_date, total_amount'
)
FROM volumes.documents.invoices;

-- Result: Structured columns extracted from unstructured PDFs
```

Listing 3: AI Parse Document Example

Use Cases:

- Extract data from scanned forms and invoices
- Parse regulatory documents
- Convert unstructured PDFs to structured tables

7.2 AI Query with Multimodal Support

```sql
-- Filter product catalog for white shoes using AI
SELECT AI_QUERY(
    'Filter our catalog for white shoes',
    image_column
)
FROM product_catalog;

-- AI understands both the query and the images
```

Listing 4: Multimodal AI Query

7.3 Spatial SQL

Databricks now supports spatial SQL for geographic analysis:

- **Geometry/Geography Types**: Native support for spatial data
- **Spatial Functions**: Intersection, distance, containment
- **Use Cases**: Disaster impact analysis, location-based services

8 Data Engineering: LakeFlow and Spark Declarative Pipelines

8.1 LakeFlow Connect

Definition:

LakeFlow Connect **LakeFlow Connect** provides a unified way to ingest data from various sources into the Lakehouse. Previously, Databricks relied on data already being in cloud storage—now it actively pulls data from source systems.

Supported Connectors:

- **File Sources**: Excel, SFTP
- **Databases**: PostgreSQL, MySQL, Oracle
- **Marketing**: Google Ads, Salesforce Marketing Cloud, TikTok Ads
- **Industry**: Customer 360, Financial Services, Insurance

8.2 LakeFlow Designer

A visual interface for building ETL pipelines:

1. Drag and drop data sources from your catalog
2. Add operators (Filter, Join, Aggregate, AI Functions)
3. Visual representation of data flow
4. Behind the scenes: Generates Spark Declarative Pipeline code

8.3 Spark Declarative Pipelines

Key Information

Evolution of Delta Live Tables (DLT):

- **Original Name**: Delta Live Tables (DLT)
- **Renamed To**: Live Tables Pipeline (LTP)
- **Current Name**: Spark Declarative Pipelines
- **Status**: Now fully open-sourced and Generally Available

9 Feature Release Lifecycle

Understanding when features are safe to use in production:

Stage	Access	Production Ready?
Beta	Very early, limited	No—experimental only
Private Preview	Selected customers	No—for evaluation
Public Preview	Available in documentation, anyone can try	Caution—some enterprises wait
Generally Available (GA)	Production-ready	Yes—enterprise use

Table 7: *Feature Release Stages*

10 Quiz 2 Review: Spark and Delta Lake Concepts

Key Summary

This section reviews key concepts from Quiz 2 that students found challenging.

10.1 Spark Optimization: The Four S's

Problem	Description	Solution
Data Skew	Uneven data distribution across partitions	Salting, broadcast joins, AQE
Shuffle	Data movement between nodes	Minimize wide transformations, optimize partitioning
Spill	Memory overflow to disk	Increase executor memory, reduce data size
Small Files	Too many tiny files	Use OPTIMIZE, predictive optimization

Table 8: *Spark Performance Problems and Solutions*

10.2 Broadcast Variables

Warning

Common Misconception: Broadcast variables are *not* shared across the cluster.

Correct Understanding:

- Broadcast variables are **immutable**—cannot be changed after creation
- They are **local to each worker node**—each node gets its own copy
- Used for small lookup tables that would otherwise require shuffle

10.3 Repartition vs Coalesce

```
# INCREASING partitions (12 -> 24): Use repartition
df_more = df.repartition(24)  # Full shuffle

# DECREASING partitions (24 -> 12): Use coalesce
df_less = df.coalesce(12)  # No shuffle, more efficient

# ERROR: coalesce(24) when you have 12 partitions
# This won't increase partitions!
```

Listing 5: Partition Operations

10.4 collect() vs take()

Important:

The Danger of collect() **collect()** brings *all* data to the driver. This causes Out of Memory (OOM) errors with large datasets.

Best Practices:

- Use `take(n)` to retrieve only what you need
- Use `collect()` only for debugging small datasets
- Always be suspicious of `collect()` in production code

```
# DANGEROUS for large datasets
all_data = df.collect()  # OOM risk!

# SAFE alternative
sample = df.take(5)  # Returns only 5 rows
```

10.5 Delta Lake Transaction Log

Definition:

Delta Log Structure Every Delta table has an `_delta_log` folder containing:

- **JSON files**: One per transaction (000000000000.json, 000000000001.json, ...)
- **Checkpoint files**: Parquet files created every 10 transactions
- This enables **time travel** without separate copies

Warning

Quiz Correction: The question asked about Delta log structure. The correct answer is that each transaction creates a **separate JSON file** (not a single JSON). If you marked "single JSON" as correct, please reach out to have your grade adjusted.

10.6 DataFrame Operations

```
# EXPLODE is NOT a DataFrame method
# WRONG:
df.explode("array_column")  # This doesn't exist!

# CORRECT - use inside select:
from pyspark.sql.functions import explode
df.select("id", explode("array_column").alias("item"))

# DROP DUPLICATES - use a list for columns
# WRONG:
df.dropDuplicates("col1", "col2")  # Incorrect syntax

# CORRECT:
```

```
df.dropDuplicates(["col1", "col2"])  # Pass columns as a list
```

Listing 6: Common DataFrame Gotchas

10.7 Generated Columns

```
CREATE TABLE transactions (
    transaction_id BIGINT,
    transaction_timestamp TIMESTAMP,
    -- Generated column: automatically computed
    transaction_date DATE GENERATED ALWAYS AS (
        CAST(transaction_timestamp AS DATE)
    )
) PARTITIONED BY (transaction_date);

-- Benefit: Filtering by timestamp automatically
-- resolves the partition pruning on transaction_date
```

Listing 7: Generated Always As

11 Assignment 3 Review: Streaming and Data Architecture

11.1 Kafka vs Kinesis

Aspect	Apache Kafka	AWS Kinesis
Management	Self-managed or Confluent Cloud	Fully managed by AWS
Cloud Agnostic	Yes—runs anywhere	No—AWS only
Scaling Unit	Partitions	Shards
Pricing	Confluent fees or infrastructure	Per-shard pricing
Data Distribution	Round-robin or key-based	Partition key
Ecosystem	KSQL, Kafka Connect, Schema Registry	Kinesis Data Analytics, Firehose

Table 9: *Kafka vs Kinesis Comparison*

Key Information

Other Cloud Equivalents:

- **Azure**: Event Hubs
- **Google Cloud**: Pub/Sub

11.2 Slowly Changing Dimensions (SCD)

Type	Behavior	History	Use Case
Type 1	Overwrite the row	No history kept	Correction of errors
Type 2	Add new row with version	Full history with dates/flags	Customer address changes
Type 3	Add column for previous value	Limited history	Only need previous value

Table 10: *SCD Types Summary*

```sql
MERGE INTO target_table t
USING source_table s
ON t.id = s.id
-- Only update if city actually changed
WHEN MATCHED AND t.city <> s.city THEN
    UPDATE SET t.city = s.city, t.updated_at = CURRENT_TIMESTAMP
WHEN NOT MATCHED THEN
    INSERT (id, name, city, updated_at)
    VALUES (s.id, s.name, s.city, CURRENT_TIMESTAMP);
```

Listing 8: MERGE for SCD Type 1 with Condition

11.3 Spark Streaming Key Concepts

11.3.1 Trigger Options

```
# DEPRECATED - reads all available data at once
stream.writeStream \
    .trigger(once=True) \  # Don't use this!
    ...

# RECOMMENDED - micro-batch processing
stream.writeStream \
    .trigger(availableNow=True) \  # Use this instead
    ...

# CONTINUOUS - for low-latency (sub-100ms)
stream.writeStream \
    .trigger(processingTime='10 seconds') \
    ...
```

Listing 9: Streaming Trigger Options

11.3.2 Checkpointing and Exactly-Once Processing

```
stream = (
    spark.readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "server:9092")
    .option("subscribe", "sensor_data")
    .option("startingOffsets", "earliest")  # or "latest"
    .load()
)

# Checkpoint location enables exactly-once processing
stream.writeStream \
    .format("delta") \
    .outputMode("append") \
    .option("checkpointLocation", "/checkpoints/sensor_stream") \
    .table("bronze.sensor_readings")
```

Listing 10: Streaming with Checkpoint

Definition:

Exactly-Once Processing Achieved through the combination of:

1. **Checkpointing**: Tracks progress through the stream
2. **Idempotent Sinks**: Ensure operations can be safely replayed

3. **WAL (Write-Ahead Log)**: Records operations before execution

11.3.3 Reading from Kafka

```python
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StringType, IntegerType

# Define schema
schema = StructType() \
    .add("user_id", StringType()) \
    .add("device_id", StringType()) \
    .add("steps", IntegerType())

# Read and parse Kafka stream
df = (
    spark.readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "server:9092")
    .option("subscribe", "fitness_data")
    .load()
    # Kafka value is binary, cast to string
    .selectExpr("CAST(value AS STRING)")
    # Parse JSON
    .select(from_json(col("value"), schema).alias("data"))
    # Flatten
    .select("data.*")
    # Deduplicate
    .dropDuplicates(["user_id", "device_id"])
)
```

Listing 11: Kafka Stream Processing

11.4 AutoLoader (Cloud Files)

```python
# Read incrementally from cloud storage
df = (
    spark.readStream
    .format("cloudFiles")  # AutoLoader format
    .option("cloudFiles.format", "csv")
    .option("cloudFiles.schemaLocation", "/schema/sales")
    .option("header", "true")
    .load("/data/sales/")
)

# Write to Delta table
df.writeStream \
    .format("delta") \
    .option("checkpointLocation", "/checkpoints/sales") \
    .option("mergeSchema", "true") \
```

```
    .outputMode("append") \
    .table("bronze.sales")
```

Listing 12: AutoLoader Example

Key Information

AutoLoader SQL Equivalent: `COPY INTO`

```
COPY INTO bronze.sales
FROM '/data/sales/'
FILEFORMAT = CSV
FORMAT_OPTIONS ('header' = 'true')
COPY_OPTIONS ('mergeSchema' = 'true');
```

12 Machine Learning Review (Quiz 2)

12.1 Hyperparameter Optimization

Warning

Hyperopt Deprecation: The `hyperopt` library has stopped development. Use `Optuna` instead for hyperparameter optimization in new projects.

```python
import optuna
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

def objective(trial):
    # Define hyperparameters to tune
    n_estimators = trial.suggest_int('n_estimators', 10, 100)
    max_depth = trial.suggest_int('max_depth', 2, 32)

    model = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth
    )

    score = cross_val_score(model, X, y, cv=5).mean()
    return score  # Optuna maximizes by default

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

Listing 13: Optuna for Hyperparameter Tuning

12.2 MLflow Model Registry

- **Purpose**: Store, version, and govern ML models
- **Aliases**: Champion, Challenger, Production, Staging
- **Integration**: Now part of Unity Catalog
- **Auto-logging**: Use `mlflow.autolog()` for automatic metric capture

12.3 Handling Imbalanced Data

Technique	Description
Oversampling	Duplicate minority class examples
Undersampling	Remove majority class examples
SMOTE	Synthetic Minority Over-sampling Technique—create synthetic minority examples
Class Weights	Adjust loss function to penalize minority misclassification more

Table 11: *Techniques for Imbalanced Data*

13 Advanced Sharing: ABAC with Delta Sharing

Definition:

ABAC in Delta Sharing **Attribute-Based Access Control (ABAC)** policies now carry over during Delta Sharing. When a provider shares a table with ABAC policies (e.g., PII masking), the recipient automatically inherits those policies.

Key Information

Benefits:

- No separate policy configuration needed on recipient side
- Consistent data protection across organizations
- Share tables, views, and materialized views with embedded policies
- Row-level security and column masking preserved

13.1 Sharing to Iceberg Clients

Through **UniForm** (Delta with Iceberg metadata) or **Managed Iceberg**:

- Delta tables generate both Delta and Iceberg metadata
- External tools that only understand Iceberg can read the data
- Delta Sharing protocol works for both formats
- Zero-copy sharing—no data duplication

14 Course Summary and Next Steps

Key Summary

Key Takeaways from Lecture 14:

1. **Databricks One**: Unified experience for business users across workspaces
2. **Lakebase**: Revolutionary OLTP with scale-to-zero compute
3. **AI/BI**: Genie for "what" questions, Research Mode for "why"
4. **Agent Bricks**: Low-code AI agent creation
5. **Lakehouse Apps**: Build secure, interactive applications on data
6. **MCP**: Connect agents to external tools and data
7. **Quiz Review**: Broadcast variables are immutable and local to workers
8. **Streaming Review**: Use `availableNow` instead of deprecated `once`

14.1 Upcoming Events

- **Next Lecture**: Guest speakers from industry—highly recommended attendance
- **Assignment 4**: Review the final assignment requirements
- **Final Project**: Consider building a Lakehouse App for extra credit

Term	Definition
OLTP	Online Transaction Processing—real-time transactional systems
OLAP	Online Analytical Processing—analytical data warehousing
Lakebase	Databricks' managed PostgreSQL for OLTP workloads
Databricks One	Unified business user experience across workspaces
Genie	Natural language interface for data querying
Research Mode	Hypothesis-driven deep analysis mode in Genie
Agent Bricks	Low-code framework for building AI agents
MCP	Model Context Protocol—standard for agent-tool integration
Lakehouse Apps	Web applications built on the Databricks platform
UniForm	Delta tables with Iceberg metadata compatibility
ABAC	Attribute-Based Access Control
SCD	Slowly Changing Dimensions—dimension management pattern
AQE	Adaptive Query Execution—Spark runtime optimization
Checkpoint	Streaming progress tracking for fault tolerance

Glossary

One-Page Summary

CSCI E-103 Lecture 14: Databricks Roadmap & Emerging Features

1. **Databricks One & Portal Experiences**
   - Three portals: Lakehouse (data professionals), Databricks One (business users), Lakebase (OLTP)
   - Account-level access with vanity URLs and domain organization
2. **Lakebase: OLTP Revolution**
   - Managed PostgreSQL with storage-compute separation
   - Scale-to-zero: Pay only when compute is used
   - Production/Development branching like Git
   - Bidirectional sync with Lakehouse
3. **AI/BI Enhancements**
   - Genie: "What" questions → SQL → Answers
   - Research Mode: "Why" questions → Hypothesis exploration
   - Document upload: Combine PDFs/CSVs with structured data
4. **Agent Bricks & MCP**
   - AutoML-like agent creation: Point to data, specify task
   - MCP servers in marketplace for agent integrations
   - Governed by Unity Catalog
5. **Quiz 2 Key Points**
   - Broadcast variables: Immutable + Local to each worker
   - Repartition (increase) vs Coalesce (decrease)