

CSCI E-103

Data Engineering for Analytics to Solve Business Challenges

Operationalizing Data Pipelines

Lecture 07

Anindita Mahapatra & Eric Gieseke

Harvard Extension, Fall 2025

Logistics

- Assignment-1: Spark
 - Grades out, let us know if you didn't receive yours
- Assignment-2: ETL and Exploratory Data Analysis (EDA)
 - Grades will be out soon
- Assignment-3: Batch & Streaming
- Quiz-1
- Case Study-1: Data Architectures (will release towards EoW)
 - Please check your groups and company to research
 - A suggested template deck is provided for organizing your material

Agenda

- Operationalizing a Pipeline from PoC to Production
- Requirements - functional & non-functional
- Data Lake Guiding Principles
- Data Pipeline Components
- Data checks
 - Ex. Missing Data, Duplicates, Late data, Upstream error
- Data profiling, data quality, PII data masking
- Data Governance
- Data Scaling
- Pipeline Automation
- Data Security
- Disaster Recovery
- Lab
 - Architecture considerations & Data checks
 - Delta Live Tables (DLT) - Simplifying ETL of data pipelines while hardening for production

Clear Requirements for a high quality data product/service

Functional Requirements

*Specifies **What** The System Should Do*

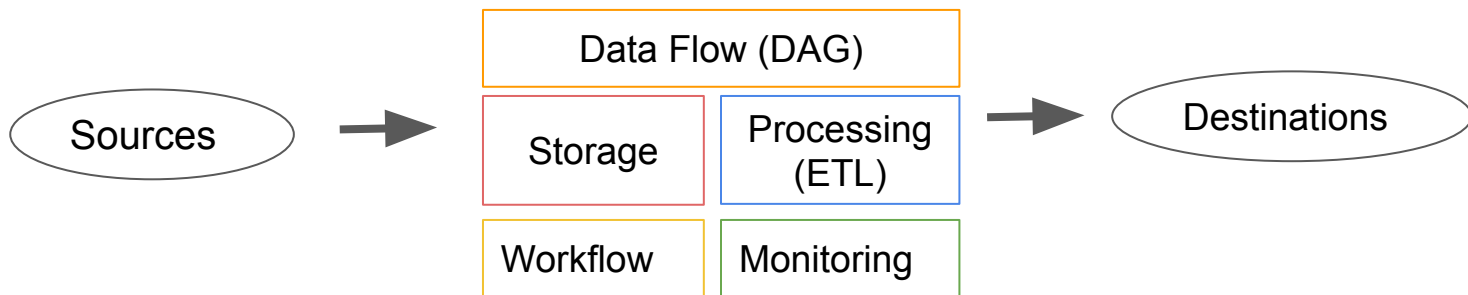
- Business Rules
- Transaction corrections, adjustments and cancellations
- Administrative functions
- Authentication
- Authorization levels
- Audit Tracking
- External Interfaces
- Certification Requirements
- Reporting Requirements
- Historical Data

Non-Functional Requirements

*Specifies **How** The System Performs A Certain Function*

- Performance – for example Response Time, Throughput, Utilization, Static Volumetric
- Scalability
- Capacity
- Availability
- Reliability
- Recoverability
- Maintainability & Serviceability
- Security & Regulatory
- Manageability
- Environmental
- Data Integrity
- Usability & Interoperability

Components of a Data Pipeline



- Sources/Destinations
- Data Flow graph captures data dependencies and movement
- Storage
- ETL or ELT
- Workflow controls Scheduling & Orchestration
- Monitoring for operational continuity and data drift
 - Robustness - notification, retries, auto heal ex. Scale, Failures
 - Metrics - performance, data characteristics, KPIs

Guiding Principles for an effective Data Lakehouse

[Link](#)

1. Curate Data and Offer Trusted Data-as-Products
2. Remove Data Silos and Minimize Data Movement
3. Democratize Value Creation through Self-Service Experience
4. Adopt an Organization-wide Data Governance Strategy
5. Encourage the Use of Open Interfaces and Open Formats
6. Build to Scale and Optimize for Performance & Cost

1. Curate Data and Offer Trusted Data-as-Products

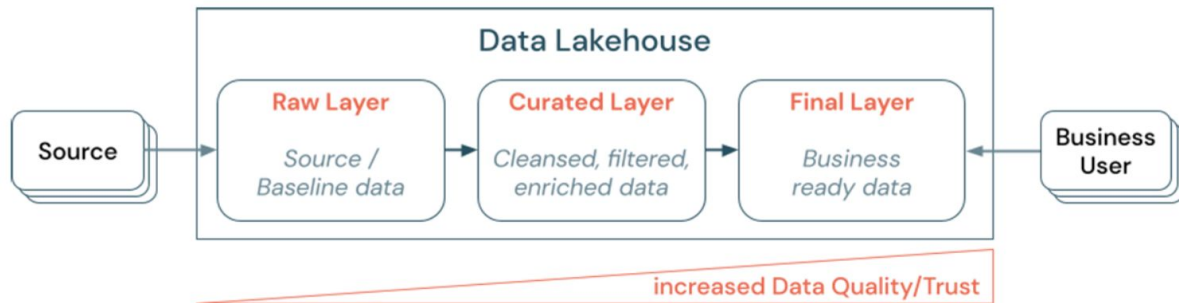


Figure 1: Data quality, as well as trust in data, increase as data progresses through the layers.

● Multi-Hop Architecture

- Bronze - Raw
 - Usually necessary for rebuilding/re-processing
- Silver - Harmonized
 - Provides a sound, reliable foundation for analyses and reports across all roles and functions
- Gold - Curated, Prepares data around security needs (Eg. anonymization, pre-aggregated)
 - Aggregated
 - **Business Ready** - Semantic, Marts/Cubes

2. Remove Data Silos and Minimize Data Movement

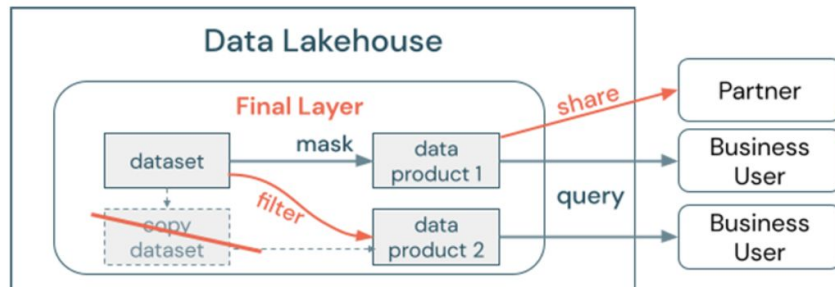


Figure 2: Lakehouses have capabilities that allows business user to query data, as well as share it with partners.

- Data movement introduces fragility, quality concerns, latency, cost
- Avoid data duplication that introduces operational dependencies
- Data sync
 - Clone (shallow, deep)
 - Sharing
 - Federation
 - Views/materialized views
 - Delta time travel

3. Democratize Value Creation through Self-Service Experience

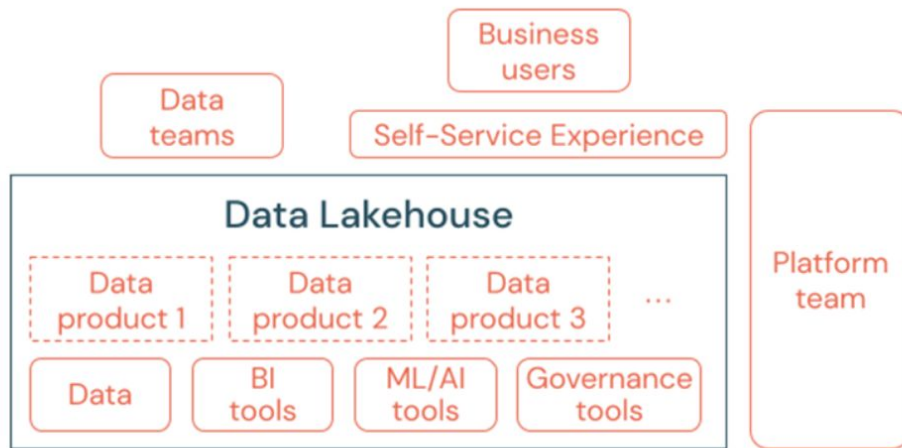


Figure 3: Lakehouse allows data teams to build data products that can be used through a self-service experience.

- Data driven decision making
- Centralized -> Decentralized decision making
- Teams need access to the right data along with the right tooling
- Self Service requires effective Guardrails

4. Adopt an Organization-wide Data Governance Strategy

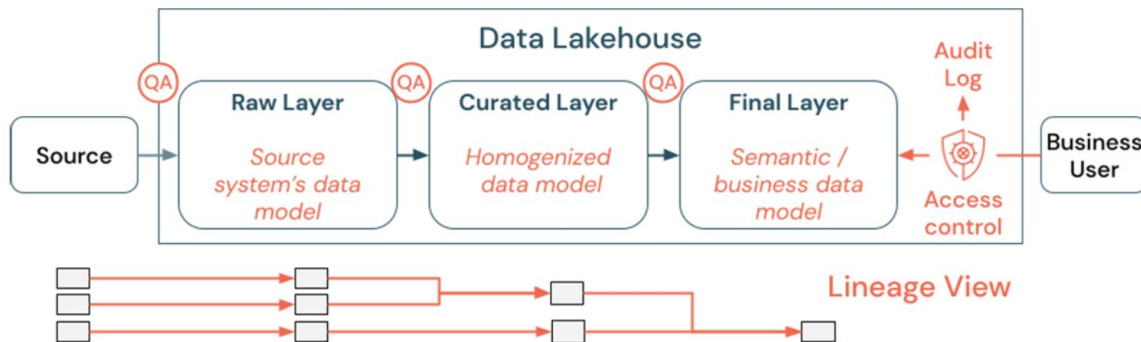


Figure 4: Lakehouse governance not only has strong access controls in place, but also can track data lineage.

- Data Quality
 - Constraints to ensure quality
 - Data timeliness via SLAs
- Data Catalog
 - Data Discovery, Metadata, Lineage
- Access Control
 - Row/column level access
 - Masking/anonymization
 - Audit Logs

Governance				
Users	Data		Compute	
Who has access	Who has access to What data		Cloud Infra Team sets up Privileges (Eg. IAM roles, KMS keys)	
Identity Providers	Within Databricks	Outside	Create	Use Limit
Eg. AD, Okta	Unity Catalog	Enterprise Catalog	Workspace admin creates cluster policies & grants entitlements to groups	

Figure 1: Governance of Data Platforms

5. Encourage the Use of Open Interfaces and Open Formats

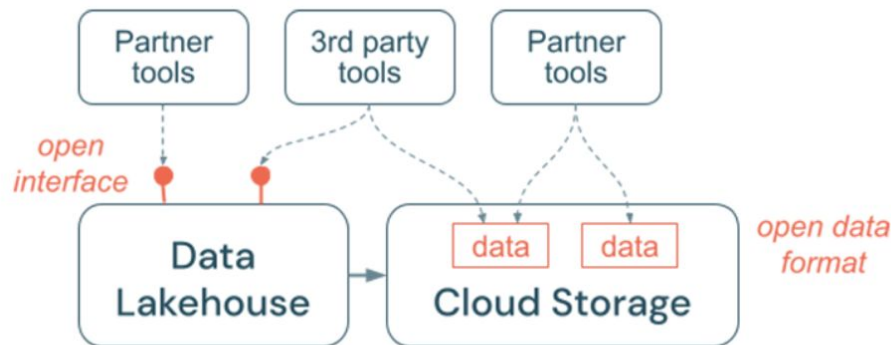


Figure 5: Lakehouse is built on open sources and open interfaces for easier integration with third party tools.

- **Interoperability of systems**
 - opens an ecosystem of partners who can quickly leverage the open interfaces to integrate their tools
- **No vendor lock-in**
 - increases the longevity and portability of the data
- **Lower Cost**
 - access the data directly on the cloud storage without the need to pipe it through a proprietary platform that can incur high egress and computation costs.

6. Build to Scale and Optimize for Performance & Cost

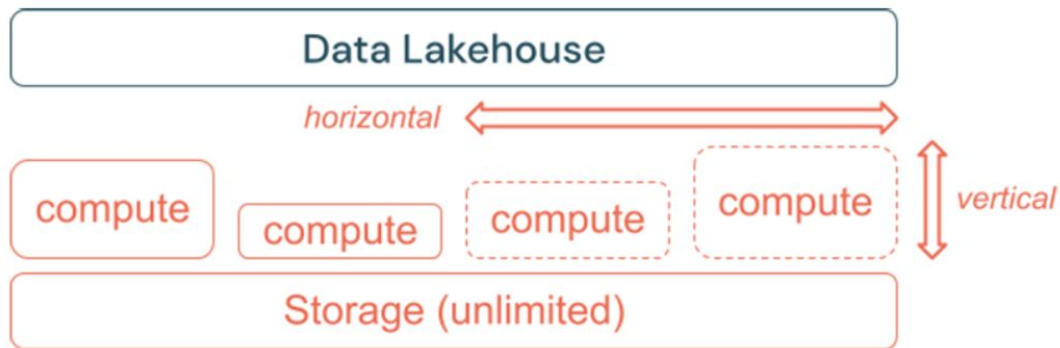


Figure 6: Lakehouse detaching storage from compute as well as leverage elastic compute for better scalability.

- **Scaling**
 - Vertical Scaling
 - Horizontal Scaling
- **Price-Performance**
 - Cost Vs Performance
- **Storage Vs Compute**
 - Decoupled architectures

Data Checks

- Data Checks

- Missing information
- Incomplete information
- Schema mismatch
- Differing formats or data types
- User errors when writing data producers



ParseMode	Behavior
PERMISSIVE	Includes corrupt records in a "_corrupt_record" column (by default)
DROPMALFORMED	Ignores all corrupted records
FAILFAST	Throws an exception when it meets corrupted records

- Bad Records or Files

- files may not be readable (for instance, they could be missing, inaccessible or corrupted)
- even if the files are processable, some records are bad (for example, due to syntax errors and schema mismatch)
 - Use the **badRecordsPath** option in a file-based data source

```
data = """"{"a": 1, "b":2, "c":3}|{"a": 1, "b":2, "c":3}|{"a": 1, "b, "c":10}""".split('|')
```

```
corruptDF = (spark.read.option("mode", "DROPMALFORMED").json(sc.parallelize(data)))
```

```
corruptDF = (spark.read.option("mode", "PERMISSIVE") .option("Corrupt", "_corrupt_record").json(sc.parallelize(data)))
```

```
corruptDF = (spark.read.option("badRecordsPath", myBadRecords).json(sc.parallelize(data)))
```

Data manipulation functions

- Duplicate Data

- user 1 performs a write operation on Delta table A.
- At the same time, user 2 performs an append operation on Delta table A.
- `dedupedDF = duplicateDF.dropDuplicates(["id", "favorite_color"])`

- Strategies for Imputing missing values

- **Dropping these records:** Works when you do not need to use the information for downstream workloads
 - `corruptDroppedDF = corruptDF.dropna("any")`
- **Adding a placeholder (e.g. -1):** Allows you to see missing data later on without violating a schema
- **Basic imputing:** Allows you to have a "best guess" of what the data could have been, often by using the mean of non-missing data
 - `corruptImputedDF = corruptDF.na.fill({"temperature": 68, "wind": 6})`
- **Advanced imputing:** using more advanced strategies such as clustering machine learning algorithms

Function	Use
<code>explode()</code>	Returns a new row for each element in the given array or map
<code>pivot()</code>	Pivots a column of the current DataFrame and perform the specified aggregation
<code>cube()</code>	Create a multi-dimensional cube for the current DataFrame using the specified columns, so we can run aggregation on them
<code>rollup()</code>	Create a multi-dimensional rollup for the current DataFrame using the specified columns, so we can run aggregation on them

Data Quality

- Constraints

- [Docs](#)
- `ALTER TABLE default.people10m ADD CONSTRAINT dateWithinRange CHECK (birthDate > '1900-01-01');`

- Schema Enforcement

- Generated Columns

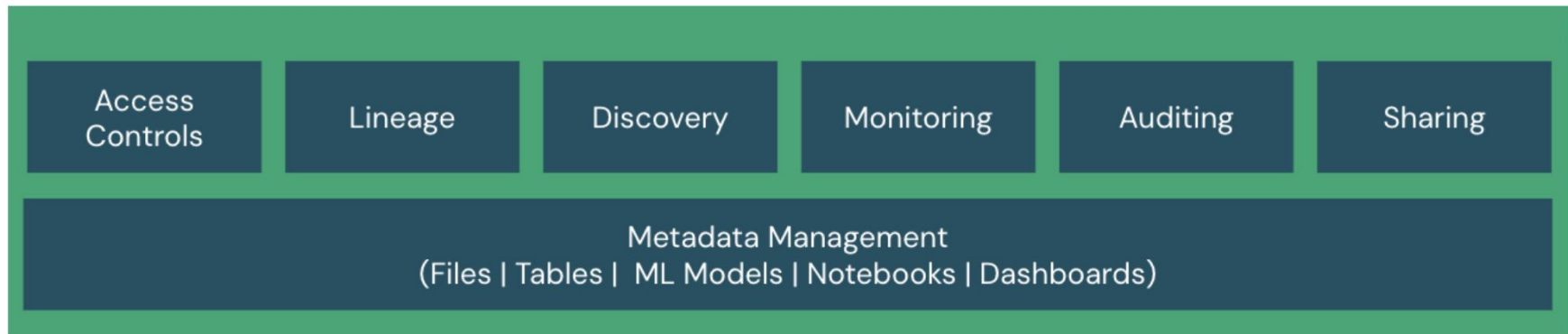
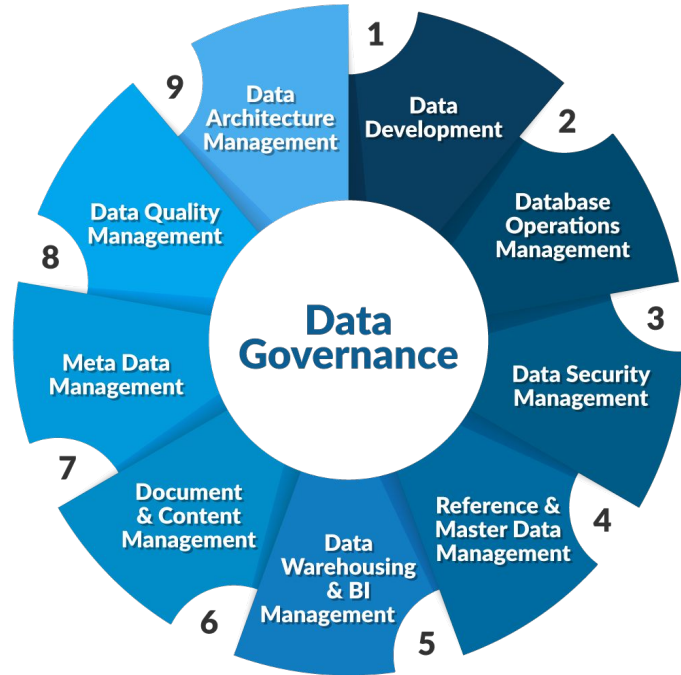
[Docs](#)

```
CREATE TABLE default.people10m (  
  id INT,  
  gender STRING,  
  birthDate TIMESTAMP,  
  dateOfBirth DATE GENERATED ALWAYS AS (CAST(birthDate AS DATE)),  
  salary INT  
)  
USING DELTA  
PARTITIONED BY (gender)
```

Data Governance

Touches every aspect of data not just security

- For All your Data
 - structured/semi/Un-structured
- And All your Data Assets
 - Model
 - Dashboard
 - Services & Products



Scalability Vs Elasticity Vs Availability

Both **scalability** and **elasticity** help to improve availability and performance when demand is changing

A system is said to be **scalable** if it can increase its workload and throughput when additional resources are added.

Elasticity is the degree to which a system can adapt to workload changes by provisioning and deprovisioning resources in an on-demand manner, such that at each point in time the available resources match the current demand as closely as possible. The need is to avoid over/under provisioning.

Reliability - correct behavior

Availability - service is up

Scaling is of 2 types

- **Scale Up** - vertical scaling - bigger nodes
- **Scale Out** - horizontal scaling - more nodes - Autoscaling

Lakeflow helps you get great data

Ingest

All your data in
one place



Connect

Transform

Efficiently clean,
reshape and join data



**Declarative
Pipelines**

Orchestrate

Run workloads in
production



Jobs

Launched this year

Task Types

Power BI (Preview)
dbt Platform (Preview)
dbt Core (GA)
Clean Rooms (GA)
AI/BI Dashboard (GA)
Lakeflow Connect (GA)

Advanced Control Flow

For each Task (GA)
Conditional Task (GA)
Partial runs (GA)
SQL Task Values (GA)
Backfill runs (Preview)

Scalability

Serverless Compute (GA)
Higher task limits (GA)

Triggers

File Trigger (GA)
Table Update Trigger (GA)
Periodic Triggers (GA)

Monitoring

Jobs system table (GA)
Pipelines system table (GA)
Unified tagging, listand search (GA)

Analytics projects require collaboration

Data Intelligence
Platform

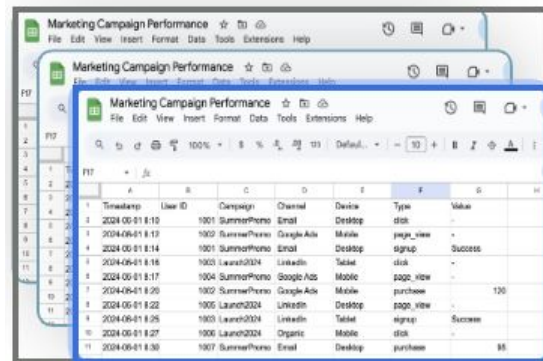


Lakeflow

Data Engineers



Spreadsheets

A screenshot of a spreadsheet application window titled 'Marketing Campaign Performance'. The spreadsheet displays a table with columns for Timestamp, User ID, Campaign, Channel, Device, Type, and Value. The data includes various marketing events like 'signup', 'purchase', and 'click' across different campaigns and devices.

Timestamp	User ID	Campaign	Channel	Device	Type	Value
2024-06-01 11:10	1001	SummerPhone	Email	Desktop	click	-
2024-06-01 11:13	1002	SummerPhone	Google Ads	Mobile	page_view	-
2024-06-01 11:14	1001	SummerPhone	Email	Desktop	signup	Success
2024-06-01 11:16	1003	Launch2024	LinkedIn	Tablet	click	-
2024-06-01 11:17	1004	SummerPhone	Google Ads	Mobile	page_view	-
2024-06-01 11:20	1002	SummerPhone	Google Ads	Mobile	purchase	120
2024-06-01 11:22	1005	Launch2024	LinkedIn	Desktop	page_view	-
2024-06-01 11:25	1003	Launch2024	LinkedIn	Tablet	signup	Success
2024-06-01 11:27	1006	Launch2024	Organic	Mobile	click	-
2024-06-01 11:30	1007	SummerPhone	Email	Desktop	purchase	85

Business
Analysts

Lakeflow Designer

The screenshot displays the Lakeflow Designer interface with a canvas titled "Example canvas". The pipeline consists of the following components:

- Source:** "Franchise details" (Retrieve all data from sales franchises.)
- Source:** "Transaction history" (Retrieve all sales transaction data.)
- Aggregate:** "Franchise revenue" (Calculate total revenue for each franchise.)
- Join:** "franchise revenue d..." (Join franchise details with revenue data.)
- Transform:** "franchise name and..." (Add name and revenue columns.)

A configuration window for the "franchise_revenue_details" join node is open, showing options for "Full join", "Inner join", "Left join", and "Right join". The "Franchise_details" and "Franchise_revenue" sections show a dropdown menu with "Franchise" selected and an "Add join clause" button.

At the bottom, there is a "Preview" section with two tables:

Input Franchise_details

	franchiseID	name	city	district	zipcode
1	3000002	Golden Crumbs	San Francisco	Mission	94110
2	3000000	The Crumbly Nook	Sydney	Bondi Beach	2026
3	3000001	Tokyo Treats	Tokyo	Shibuya	150-0042
4	3000003	Sugar Rush	Melbourne	Fitzroy	3065
5	3000004	Kobe Konfections	Kobe	Kitano	657-0838
6	3000005	Floured Fantasies	Los Angeles	Silver Lake	90026
7	3000006	Crumby Delights	Brisbane	West End	4101
8	3000007	Kyoto Kravings	Kyoto	Gion	605-0811

Output

	franchiseID	name	city	district	zipcode
1	3000016	Nagoya Nibbles	Nagoya	Sakae	460-0008
2	3000004	Kobe Konfections	Kobe	Kitano	657-0838
3	3000013	Hiroshima Delicacies	Hiroshima	Nagarekawa	730-0035
4	3000012	Crust Couture	Adelaide	Norwood	5067
5	3000026	Crumble Delights	Nashville	The Gulch	37203
6	3000000	The Crumbly Nook	Sydney	Bondi Beach	2026
7	3000017	Crumby Creations	Austin	East 6th Street	78702
8	3000035	Kumamoto Crumbles	Kumamoto	Kami-tori	860-0835