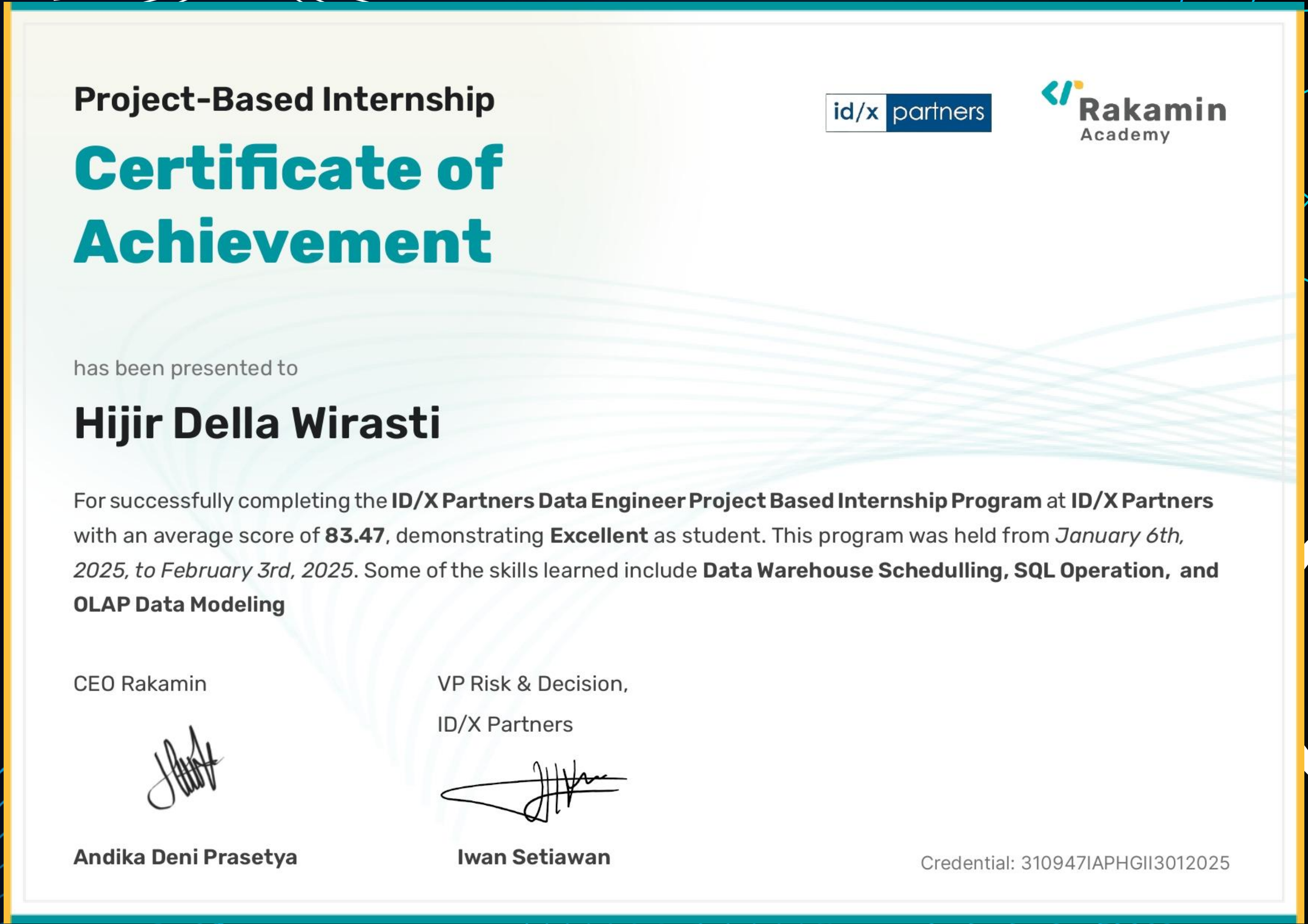
Rakamin Academy X id/x partners

id/x partners

Data Engineer

Project-Based Internship

Presented by:
Hijir Della Wirasti



Project-Based Internship

Certificate of Achievement

id/x partners

Rakamin Academy

has been presented to

Hijir Della Wirasti

For successfully completing the **ID/X Partners Data Engineer Project Based Internship Program** at **ID/X Partners** with an average score of **83.47**, demonstrating **Excellent** as student. This program was held from *January 6th, 2025, to February 3rd, 2025*. Some of the skills learned include **Data Warehouse Schedulling, SQL Operation, and OLAP Data Modeling**

CEO Rakamin

Andika Deni Prasetya

VP Risk & Decision,
ID/X Partners

Iwan Setiawan

Credential: 310947IAPHGII3012025

Hijir Della Wirasti

Data Engineer

I am Hijir Della Wirasti, a Data Engineer with expertise in Python, SQL, Airflow, Kafka, and Spark, specializing in scalable data pipelines and workflow optimization. I am currently pursuing a Master's degree in Information Systems at Telkom University and hold dual Bachelor's degrees in Ocean Engineering from ITB (GPA: 3.21) and Music Education from UPI (GPA: 3.57). I achieved 2nd Runner-Up in Dibimbing's Data Engineering Bootcamp (Score: 92) and Top 2 Student in Rakamin's Data Science Bootcamp. Passionate about data integration and big data analytics, I have delivered impactful projects like real-time streaming pipelines, batch processing, and data warehouse modeling.

Jakarta Selatan

hijirdw@gmail.com

https://www.linkedin.com/in/hijirdella/

Courses and Certification

Course / Certification	Link Certificate	Held On
Bootcamp Data Engineering Batch 6 The Most Outstanding 2nd Runner Up Performer	Link Certificate https://dibimbing.id/certificate-validation?cn=201029DE06102807	9 June 2024 - 26 October 2024 21 Weeks, 5 hours/week
Data Science - Rakamin - Batch 47 Top 2 Student of Bootcamp The Best Student of Final Project The Best Group of Final Project	Certificate of Achievement - Top 2 Student of Bootcamp Certificate of Awardee - Hijir Della Wirasti - The Best Student of Final Project (Byte Me) Certificate of Awardee - Hijir Della Wirasti - The Best Group of Final Project (Byte Me)	06 Juli 2024 - 07 Desember 2024 626 Sessions

Portfolio:
GITHUB | **https://github.com/hijirdella**

Portfolio:
Gdrive | **Data Engineering IDX Partners**

About
ID/X Partners

id/x partners was established in 2002 by ex-bankers and management consultants who have vast experiences in credit cycle and process management, scoring development, and performance management. id/x partners combined experience has served corporations across Asia and Australia regions and in multiple industries, specifically financial services, telecommunications, manufacturing and retail.

id/x partners provides consulting services that specializes in utilizing data analytic and decisioning (DAD) solutions combined with an integrated risk management and marketing discipline to help clients optimize the portfolio profitability and business process.

Comprehensive consulting service and technology solutions offered by id/x partners makes it as a one-stop service provider.

Project Portfolio

Background

A client in the banking sector from ID/X Partners faces challenges in managing data extraction from multiple sources, including Excel, CSV, and SQL Server databases. These data sources include information about transactions, accounts, customers, branches, cities, and states. Due to the complexity of these disparate sources, the client experiences delays in reporting and data analysis, impacting decision-making and operational efficiency.

Available Data

1. **Excel File:**
   - transaction_excel.xlsx contains transaction details.
2. **CSV File:**
   - transaction_csv.csv contains transaction details.
3. **SQL Server Databases:**
   - transaction_db: Stores transaction data.
   - account: Contains account details (e.g., type, balance, status).
   - customer: Includes customer details (e.g., name, age, email, and location).
   - branch: Information about bank branches.
   - city: Details of cities, including the relationship with states.
   - state: Contains state-level information.

Project Portfolio

Problem Statement

The primary issue lies in the inability to extract and combine data from multiple sources effectively. This has resulted in:

- Redundant and inconsistent data across reports.
- Delayed data reporting and analysis.
- Inability to provide stakeholders with actionable insights promptly.

Goals

- Create a unified Data Warehouse that integrates data from all sources.
- Design a scalable ETL process to ensure timely updates.
- Develop efficient Stored Procedures to support rapid data retrieval and summary generation.

Project explanation video here!
https://drive.google.com/drive/folders/1W0_9NC2ctnLvhff2QNJwRkCZVTMHOo_d?usp=sharing

Challenges

As a Data Engineer, the task involves designing and implementing an optimal ETL (Extract, Transform, Load) process to create a Data Warehouse (DWH) for the client. This includes:

1. **Database Creation:**
   - ○ Design and create a DWH containing three dimension tables (`DimAccount`, `DimCustomer`, `DimBranch`) and one fact table (`FactTransaction`).
   - ○ Ensure all tables have proper primary and foreign key relationships.
2. **ETL Pipeline Development:**
   - ○ Build ETL jobs in Talend to extract, transform, and load data:
     - ■ Transform and load data into dimension tables.
     - ■ Merge and de-duplicate transaction data from `transaction_excel`, `transaction_csv`, and `transaction_db` into `FactTransaction`.
3. **Stored Procedures:**
   - ○ **DailyTransaction:** Summarize daily transaction counts and amounts for a given date range.
   - ○ **BalancePerCustomer:** Calculate the current balance for a specific customer based on their transactions, considering account status and transaction type.

TABLE OF CONTENTS

01 1. Data Warehouse Creation

02 2. Create ETL Job for Dimension Table

03 3. Create ETL Job for Fact Table

04 4. Create Stored Procedure

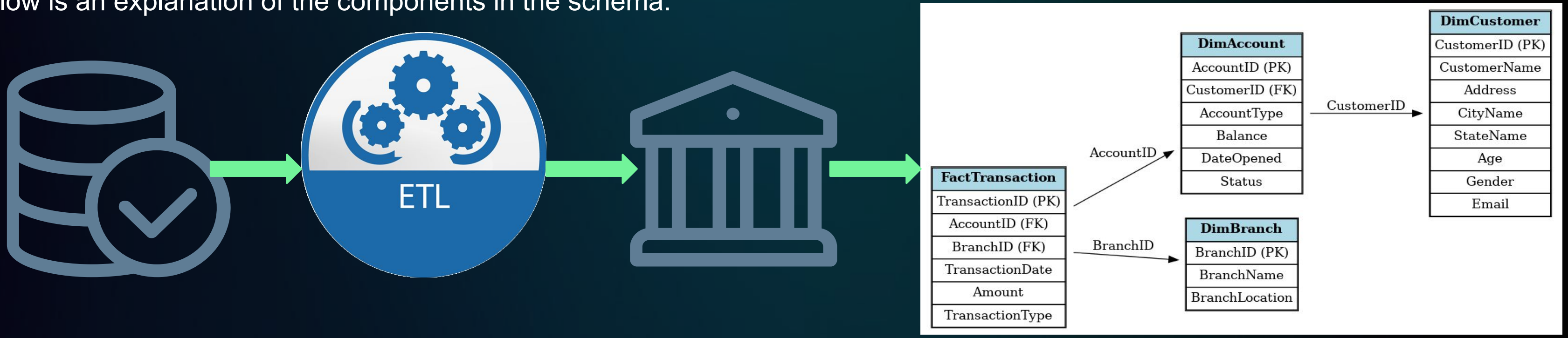
01

Data Warehouse Creation

Data Warehouse Creation

A **Data Warehouse** is a system designed to integrate data from multiple sources into a centralized database. Its primary purpose is to provide an organized database for deep analysis and reporting.

In this example, the steps involve creating **3 dimension tables** and **1 fact table** to form the Data Warehouse framework. Below is an explanation of the components in the schema:



FactTransaction
TransactionID (PK)
AccountID (FK)
BranchID (FK)
TransactionDate
Amount
TransactionType

DimAccount
AccountID (PK)
CustomerID (FK)
AccountType
Balance
DateOpened
Status

DimBranch
BranchID (PK)
BranchName
BranchLocation

DimCustomer
CustomerID (PK)
CustomerName
Address
CityName
StateName
Age
Gender
Email

1. Dimension Tables

a. DimCustomer

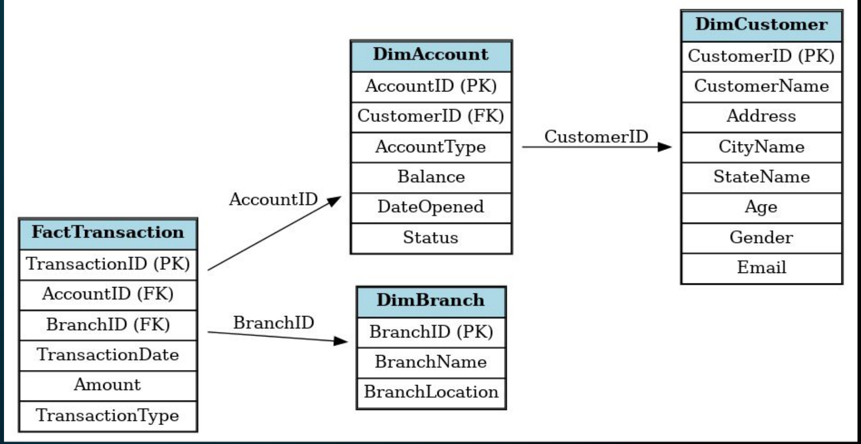
- **Primary Key (PK):** `CustomerID`
- **Descriptive Columns:**
  - `CustomerName`: The name of the customer.
  - `Address`: The address of the customer.
  - `CityName`: The city where the customer resides.
  - `StateName`: The state where the customer resides.
  - `Age`: The age of the customer.
  - `Gender`: The gender of the customer.
  - `Email`: The customer's email address.

b. DimAccount

- **Primary Key (PK):** `AccountID`
- **Foreign Key (FK):** `CustomerID` (links the customer to their account).
- **Descriptive Columns:**
  - `AccountType`: The type of account (e.g., savings, checking).
  - `Balance`: The current balance in the account.
  - `DateOpened`: The date the account was created.
  - `Status`: The account status (e.g., active, inactive).

c. DimBranch

- **Primary Key (PK):** `BranchID`
- **Descriptive Columns:**
  - `BranchName`: The name of the bank branch.
  - `BranchLocation`: The location of the branch.



FactTransaction
TransactionID (PK)
AccountID (FK)
BranchID (FK)
TransactionDate
Amount
TransactionType

DimAccount
AccountID (PK)
CustomerID (FK)
AccountType
Balance
DateOpened
Status

DimBranch
BranchID (PK)
BranchName
BranchLocation

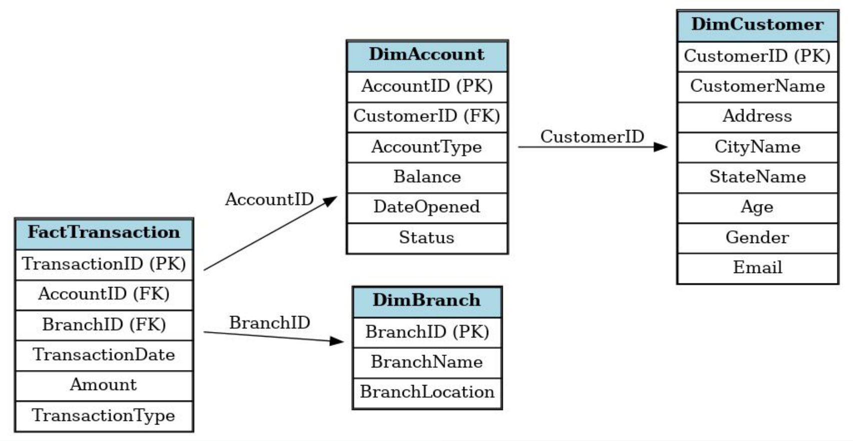
DimCustomer
CustomerID (PK)
CustomerName
Address
CityName
StateName
Age
Gender
Email

2. Fact Table

Fact tables contain transactional data linked to dimension tables.

a. FactTransaction

- **Primary Key (PK):** `TransactionID`
- **Foreign Keys (FK):**
  - `AccountID` (links the transaction to an account in `DimAccount`).
  - `BranchID` (links the transaction to a branch in `DimBranch`).
- **Transaction Data Columns:**
  - `TransactionDate`: The date and time of the transaction.
  - `Amount`: The amount of money involved in the transaction.
  - `TransactionType`: The type of transaction (e.g., deposit, withdrawal).



FactTransaction
TransactionID (PK)
AccountID (FK)
BranchID (FK)
TransactionDate
Amount
TransactionType

DimAccount
AccountID (PK)
CustomerID (FK)
AccountType
Balance
DateOpened
Status

DimCustomer
CustomerID (PK)
CustomerName
Address
CityName
StateName
Age
Gender
Email

DimBranch
BranchID (PK)
BranchName
BranchLocation

3. Relationships Between Tables

- **DimAccount** → **DimCustomer** via `CustomerID` as a Foreign Key.
- **FactTransaction** → **DimAccount** via `AccountID` as a Foreign Key.
- **FactTransaction** → **DimBranch** via `BranchID` as a Foreign Key.

STEPS

Create Dimension Tables:

- Define tables like `DimAccount`, `DimCustomer`, `DimBranch`, etc.
- Ensure each table has a `Primary Key`.

Create Fact Table:

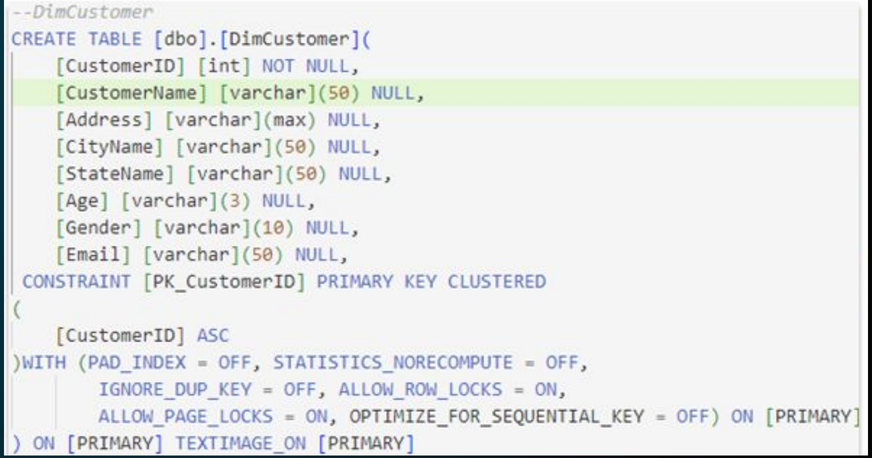
- Define the fact table (e.g., `FactTransaction`) after the dimension tables.
- Add foreign key relationships to the primary keys in the dimension tables.

Populate Dimension Tables:

- Load data into the dimension tables first (e.g., customer, account, branch details).

Populate Fact Table:

- Load transactional data into the fact table, ensuring foreign keys reference existing rows in the dimension tables.



```sql
--DimCustomer
CREATE TABLE [dbo].[DimCustomer](
    [CustomerID] [int] NOT NULL,
    [CustomerName] [varchar](50) NULL,
    [Address] [varchar](max) NULL,
    [CityName] [varchar](50) NULL,
    [StateName] [varchar](50) NULL,
    [Age] [varchar](3) NULL,
    [Gender] [varchar](10) NULL,
    [Email] [varchar](50) NULL,
 CONSTRAINT [PK_CustomerID] PRIMARY KEY CLUSTERED
(
    [CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

DimCustomer

STEPS

Create Dimension Tables:

- Define tables like `DimAccount`, `DimCustomer`, `DimBranch`, etc.
- Ensure each table has a `Primary Key`.

Create Fact Table:

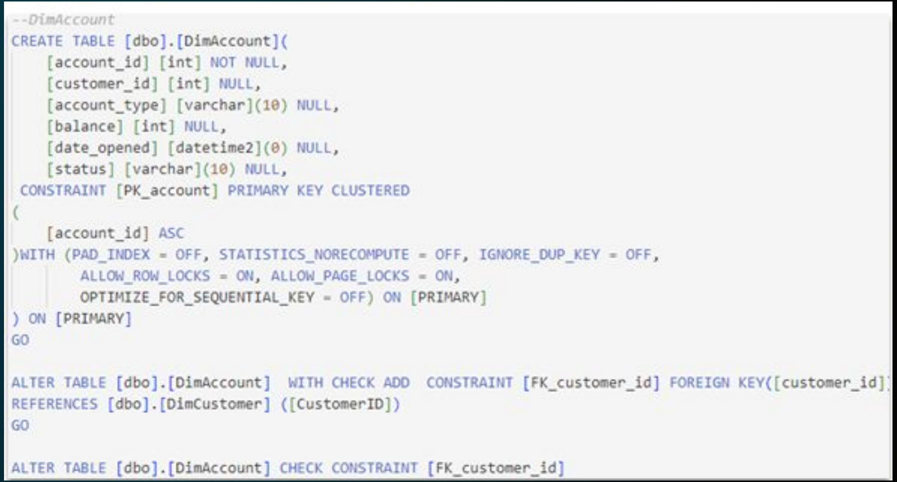
- Define the fact table (e.g., `FactTransaction`) after the dimension tables.
- Add foreign key relationships to the primary keys in the dimension tables.

Populate Dimension Tables:

- Load data into the dimension tables first (e.g., customer, account, branch details).

Populate Fact Table:

- Load transactional data into the fact table, ensuring foreign keys reference existing rows in the dimension tables.



```sql
--DimAccount
CREATE TABLE [dbo].[DimAccount](
    [account_id] [int] NOT NULL,
    [customer_id] [int] NULL,
    [account_type] [varchar](10) NULL,
    [balance] [int] NULL,
    [date_opened] [datetime2](0) NULL,
    [status] [varchar](10) NULL,
 CONSTRAINT [PK_account] PRIMARY KEY CLUSTERED
(
    [account_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
        ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[DimAccount]  WITH CHECK ADD  CONSTRAINT [FK_customer_id] FOREIGN KEY([customer_id]
REFERENCES [dbo].[DimCustomer] ([CustomerID])
GO

ALTER TABLE [dbo].[DimAccount] CHECK CONSTRAINT [FK_customer_id]
```

DimAccount

STEPS

Create Dimension Tables:

- Define tables like `DimAccount`, `DimCustomer`, `DimBranch`, etc.
- Ensure each table has a `Primary Key`.

Create Fact Table:

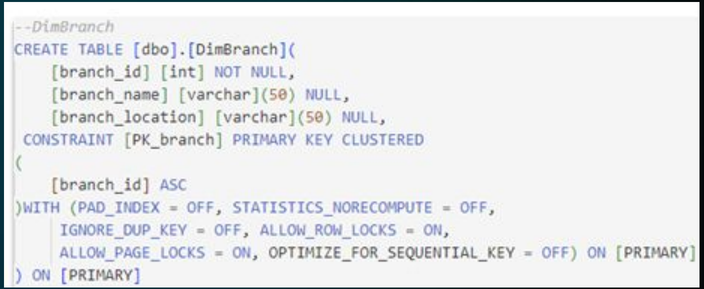
- Define the fact table (e.g., `FactTransaction`) after the dimension tables.
- Add foreign key relationships to the primary keys in the dimension tables.

Populate Dimension Tables:

- Load data into the dimension tables first (e.g., customer, account, branch details).

Populate Fact Table:

- Load transactional data into the fact table, ensuring foreign keys reference existing rows in the dimension tables.



```sql
--DimBranch
CREATE TABLE [dbo].[DimBranch](
    [branch_id] [int] NOT NULL,
    [branch_name] [varchar](50) NULL,
    [branch_location] [varchar](50) NULL,
 CONSTRAINT [PK_branch] PRIMARY KEY CLUSTERED
(
    [branch_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
     IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
     ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

DimBranch

STEPS

Create Dimension Tables:

- Define tables like DimAccount, DimCustomer, DimBranch, etc.
- Ensure each table has a Primary Key.

Create Fact Table:

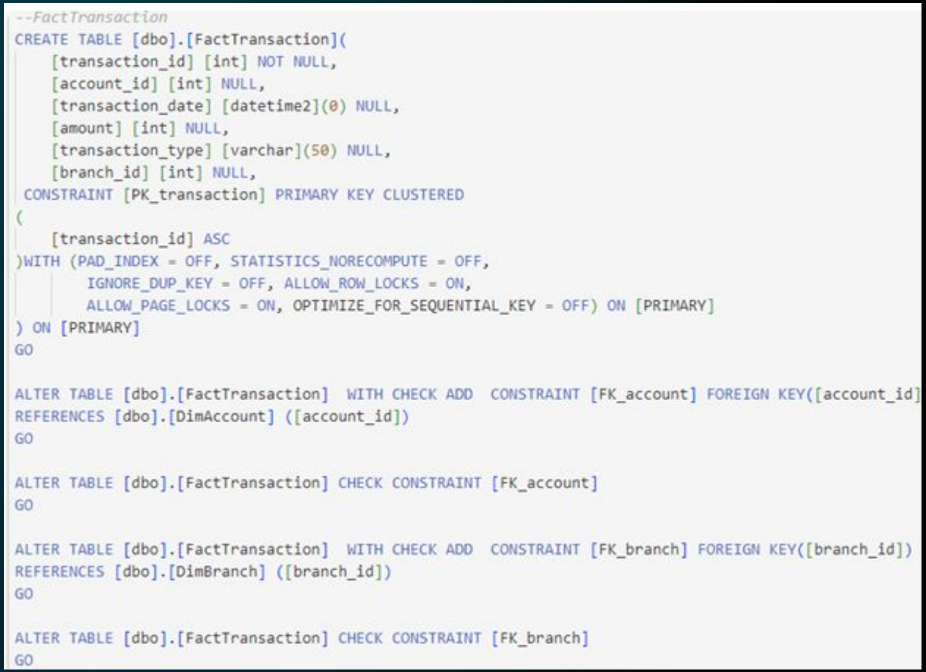
- Define the fact table (e.g., FactTransaction) after the dimension tables.
- Add foreign key relationships to the primary keys in the dimension tables.

Populate Dimension Tables:

- Load data into the dimension tables first (e.g., customer, account, branch details).

Populate Fact Table:

- Load transactional data into the fact table, ensuring foreign keys reference existing rows in the dimension tables.



```sql
--FactTransaction
CREATE TABLE [dbo].[FactTransaction](
    [transaction_id] [int] NOT NULL,
    [account_id] [int] NULL,
    [transaction_date] [datetime2](0) NULL,
    [amount] [int] NULL,
    [transaction_type] [varchar](50) NULL,
    [branch_id] [int] NULL,
 CONSTRAINT [PK_transaction] PRIMARY KEY CLUSTERED
(
    [transaction_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[FactTransaction]  WITH CHECK ADD  CONSTRAINT [FK_account] FOREIGN KEY([account_id]
REFERENCES [dbo].[DimAccount] ([account_id])
GO

ALTER TABLE [dbo].[FactTransaction] CHECK CONSTRAINT [FK_account]
GO

ALTER TABLE [dbo].[FactTransaction]  WITH CHECK ADD  CONSTRAINT [FK_branch] FOREIGN KEY([branch_id])
REFERENCES [dbo].[DimBranch] ([branch_id])
GO

ALTER TABLE [dbo].[FactTransaction] CHECK CONSTRAINT [FK_branch]
GO
```

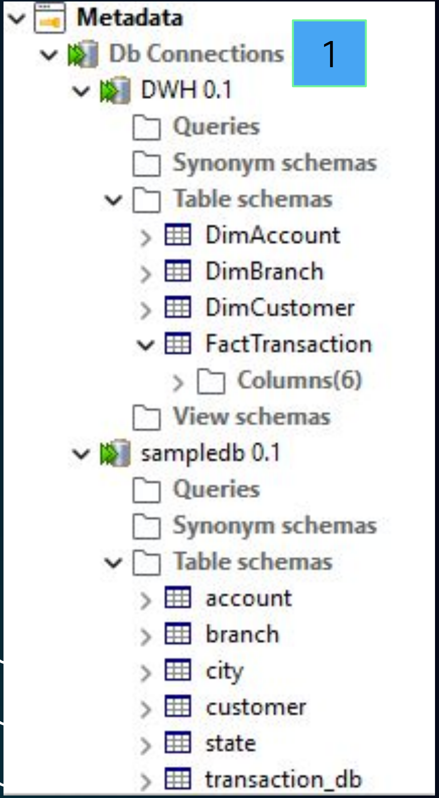
FactTransaction

02

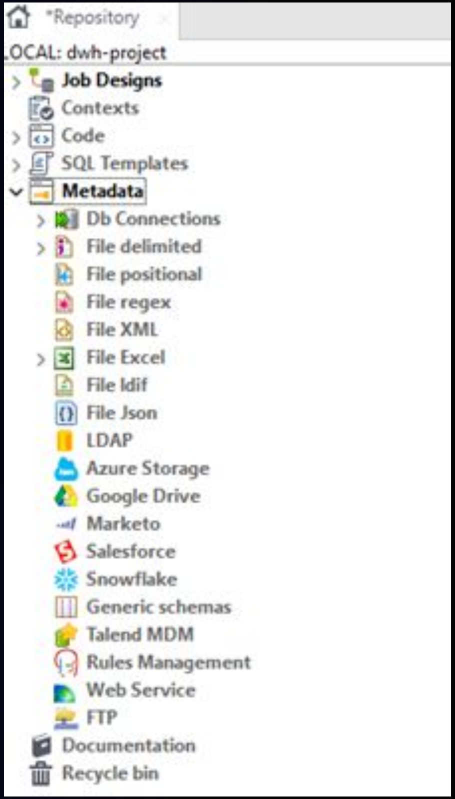
Create ETL Job for Dimension Table

talend

Create ETL Job for Dimension Table



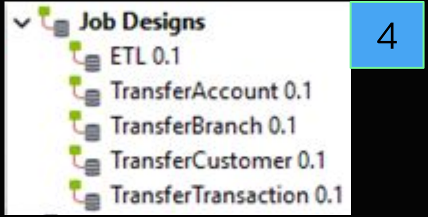
- Metadata
  - Db Connections
    - DWH 0.1
      - Queries
      - Synonym schemas
      - Table schemas
        - DimAccount
        - DimBranch
        - DimCustomer
        - FactTransaction
          - Columns(6)
      - View schemas
    - sampledb 0.1
      - Queries
      - Synonym schemas
      - Table schemas
        - account
        - branch
        - city
        - customer
        - state
        - transaction_db



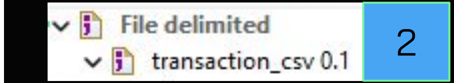
*Repository

OCAL: dwh-project

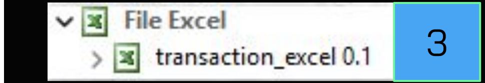
- Job Designs
- Contexts
- Code
- SQL Templates
- Metadata
  - Db Connections
  - File delimited
  - File positional
  - File regex
  - File XML
  - File Excel
  - File ldif
  - File Json
  - LDAP
  - Azure Storage
  - Google Drive
  - Marketo
  - Salesforce
  - Snowflake
  - Generic schemas
  - Talend MDM
  - Rules Management
  - Web Service
  - FTP
- Documentation
- Recycle bin



- Job Designs
  - ETL 0.1
  - TransferAccount 0.1
  - TransferBranch 0.1
  - TransferCustomer 0.1
  - TransferTransaction 0.1



- File delimited
  - transaction_csv 0.1



- File Excel
  - transaction_excel 0.1

1. DB Connections

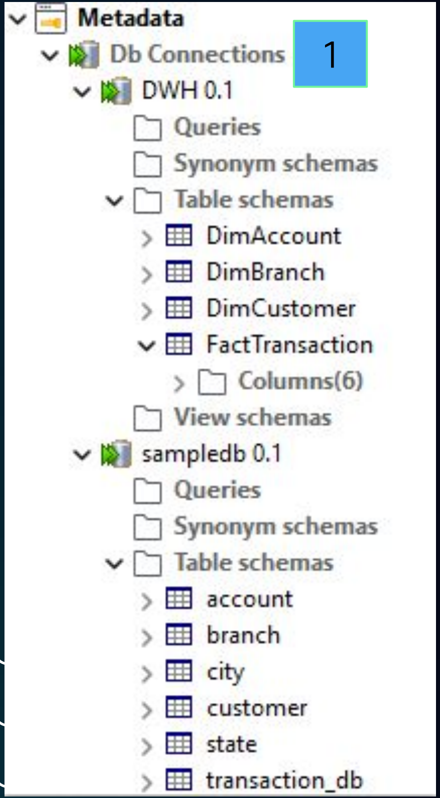
Purpose:

- Centralizes and manages database connections for relational databases such as SQL Server, MySQL, or Oracle.
- Ensures consistency across multiple ETL jobs by reusing connection metadata.

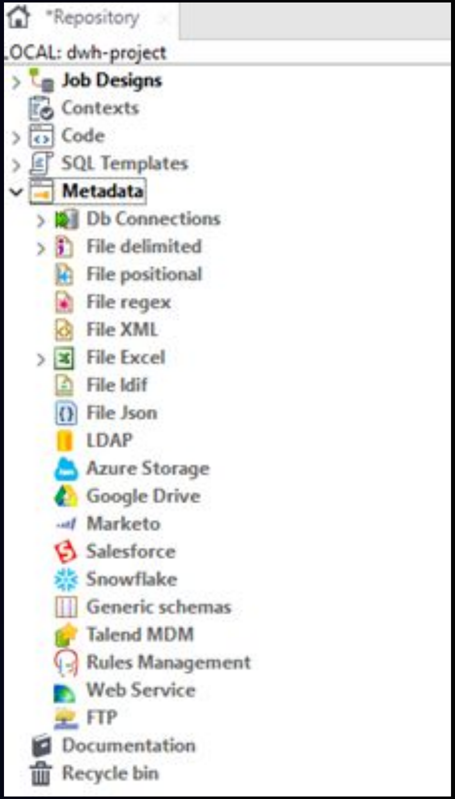
How It's Used:

- The **DWH** database is connected here to enable data extraction or loading for tables like `DimAccount`, `DimBranch`, `DimCustomer`, and `FactTransaction`.
- The **sampledb** connection is used for reading source tables like `account`, `branch`, `customer`, and `transaction_db`.

Create ETL Job for Dimension Table



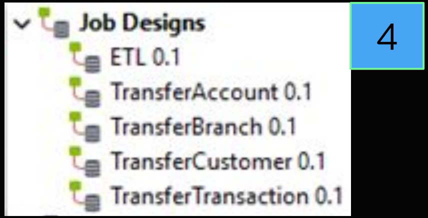
- Metadata
  - Db Connections
    - DWH 0.1
      - Queries
      - Synonym schemas
      - Table schemas
        - DimAccount
        - DimBranch
        - DimCustomer
        - FactTransaction
          - Columns(6)
      - View schemas
    - sampledb 0.1
      - Queries
      - Synonym schemas
      - Table schemas
        - account
        - branch
        - city
        - customer
        - state
        - transaction_db



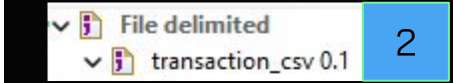
*Repository

.OCAL: dwh-project

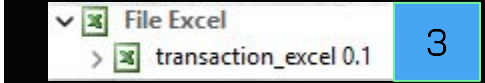
- Job Designs
- Contexts
- Code
- SQL Templates
- Metadata
  - Db Connections
  - File delimited
  - File positional
  - File regex
  - File XML
  - File Excel
  - File ldif
  - File Json
  - LDAP
  - Azure Storage
  - Google Drive
  - Marketo
  - Salesforce
  - Snowflake
  - Generic schemas
  - Talend MDM
  - Rules Management
  - Web Service
  - FTP
- Documentation
- Recycle bin



- Job Designs
  - ETL 0.1
  - TransferAccount 0.1
  - TransferBranch 0.1
  - TransferCustomer 0.1
  - TransferTransaction 0.1



- File delimited
  - transaction_csv 0.1



- File Excel
  - transaction_excel 0.1

2. File Delimited

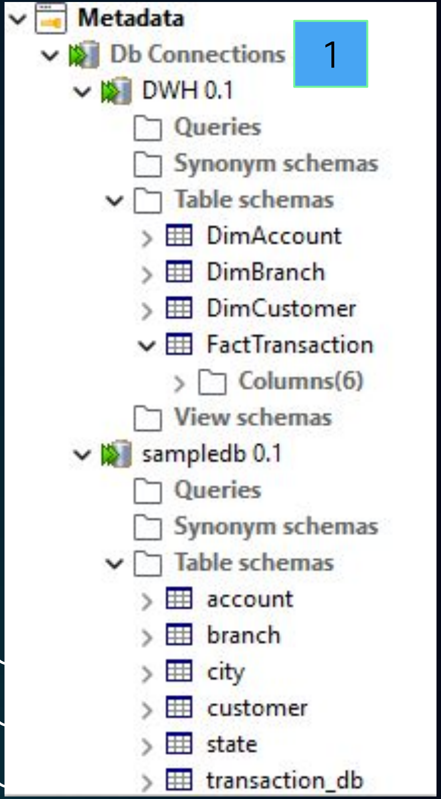
Purpose:

- **Manages delimited file formats (e.g., CSV) and stores metadata such as column names, types, and delimiters.**
- **Standardizes CSV handling to reduce errors and increase efficiency.**

How It's Used:

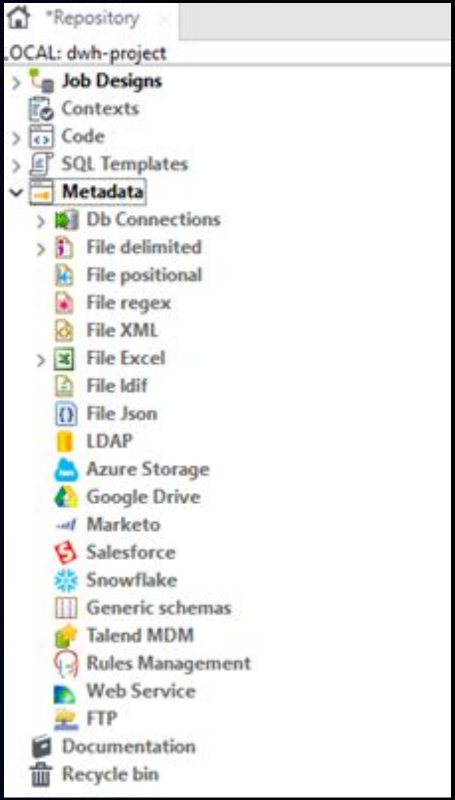
- **The file transaction_csv is defined here for integration into the ETL jobs. It contains metadata needed to process the CSV file.**

Create ETL Job for Dimension Table



1

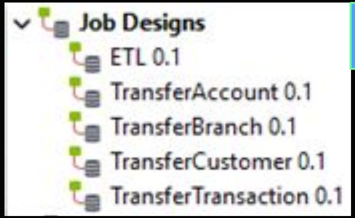
- Metadata
  - Db Connections
    - DWH 0.1
      - Queries
      - Synonym schemas
      - Table schemas
        - DimAccount
        - DimBranch
        - DimCustomer
        - FactTransaction
          - Columns(6)
      - View schemas
    - sampledb 0.1
      - Queries
      - Synonym schemas
      - Table schemas
        - account
        - branch
        - city
        - customer
        - state
        - transaction_db



*Repository

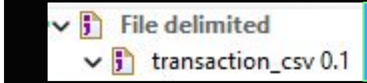
LOCAL: dwh-project

- Job Designs
- Contexts
- Code
- SQL Templates
- Metadata
  - Db Connections
  - File delimited
  - File positional
  - File regex
  - File XML
  - File Excel
  - File ldif
  - File Json
  - LDAP
  - Azure Storage
  - Google Drive
  - Marketo
  - Salesforce
  - Snowflake
  - Generic schemas
  - Talend MDM
  - Rules Management
  - Web Service
  - FTP
- Documentation
- Recycle bin



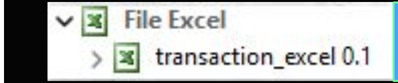
4

- Job Designs
  - ETL 0.1
  - TransferAccount 0.1
  - TransferBranch 0.1
  - TransferCustomer 0.1
  - TransferTransaction 0.1



2

- File delimited
  - transaction_csv 0.1



3

- File Excel
  - transaction_excel 0.1

3. File Excel

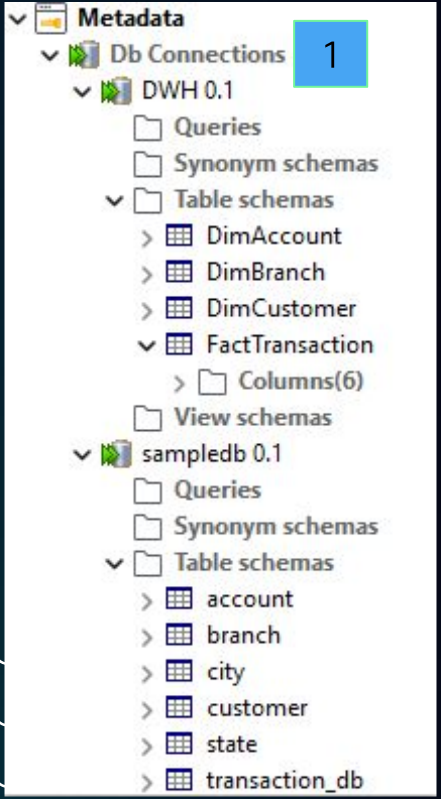
Purpose:

- **Handles metadata configurations for Excel files, making it easy to read or write Excel data in ETL workflows.**

How It's Used:

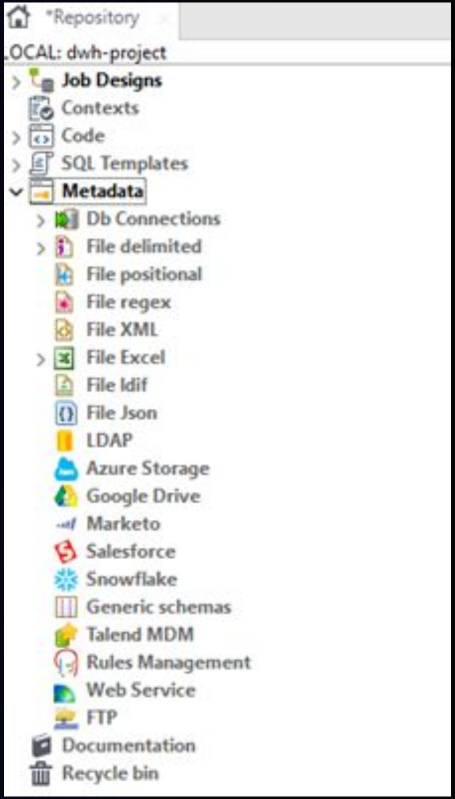
- **The file transaction_excel is registered, allowing it to be used seamlessly in jobs that require Excel input.**

Create ETL Job for Dimension Table



1

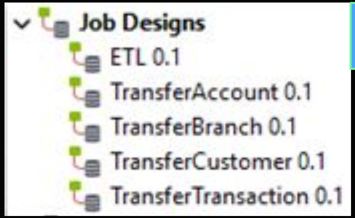
- Metadata
  - Db Connections
    - DWH 0.1
      - Queries
      - Synonym schemas
      - Table schemas
        - DimAccount
        - DimBranch
        - DimCustomer
        - FactTransaction
          - Columns(6)
      - View schemas
    - sampledb 0.1
      - Queries
      - Synonym schemas
      - Table schemas
        - account
        - branch
        - city
        - customer
        - state
        - transaction_db



*Repository

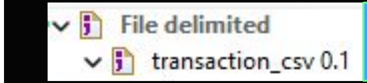
.OCAL: dwh-project

- Job Designs
- Contexts
- Code
- SQL Templates
- Metadata
  - Db Connections
  - File delimited
  - File positional
  - File regex
  - File XML
  - File Excel
  - File ldif
  - File Json
  - LDAP
  - Azure Storage
  - Google Drive
  - Marketo
  - Salesforce
  - Snowflake
  - Generic schemas
  - Talend MDM
  - Rules Management
  - Web Service
  - FTP
- Documentation
- Recycle bin



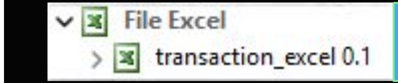
4

- Job Designs
  - ETL 0.1
  - TransferAccount 0.1
  - TransferBranch 0.1
  - TransferCustomer 0.1
  - TransferTransaction 0.1



2

- File delimited
  - transaction_csv 0.1



3

- File Excel
  - transaction_excel 0.1

4. Job Designs

Purpose:

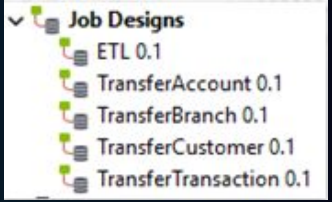
- **Contains the designed ETL workflows for transferring data from sources (CSV, Excel, databases) to the target Data Warehouse (DWH).**
- **Organizes tasks into reusable jobs.**

How It's Used:

- **TransferAccount: Moves data from source tables to the DimAccount table in the Data Warehouse.**
- **TransferBranch: Transfers branch data to the DimBranch table.**
- **TransferCustomer: Loads customer data into DimCustomer.**
- **TransferTransaction: Processes and loads transactions into the FactTransaction table.**

1. ETL Job for DimAccount and DimBranch Table

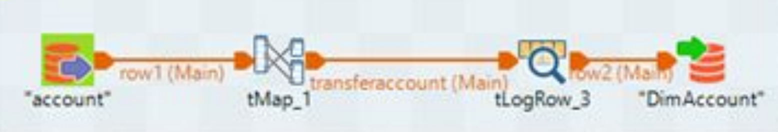
1. Create separate Job Design for each Dimension Table

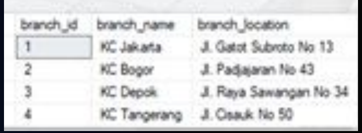


- Job Designs
  - ETL 0.1
  - TransferAccount 0.1
  - TransferBranch 0.1
  - TransferCustomer 0.1
  - TransferTransaction 0.1

2. Create input data and transform data using tMap and output data

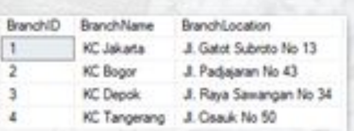
DimAccount





branch_id	branch_name	branch_location
1	KC Jakarta	Jl. Gatot Subroto No 13
2	KC Bogor	Jl. Padjajaran No 43
3	KC Depok	Jl. Raya Sawangan No 34
4	KC Tangerang	Jl. Cisauk No 50

Input

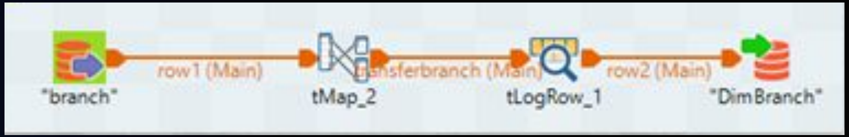


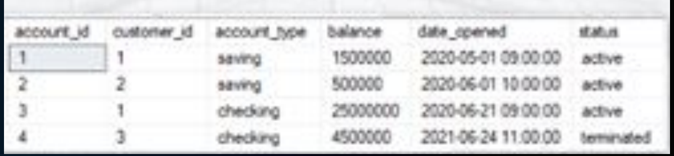
BranchID	BranchName	BranchLocation
1	KC Jakarta	Jl. Gatot Subroto No 13
2	KC Bogor	Jl. Padjajaran No 43
3	KC Depok	Jl. Raya Sawangan No 34
4	KC Tangerang	Jl. Cisauk No 50

Output

2. ETL Job for DimAccount and DimBranch Table

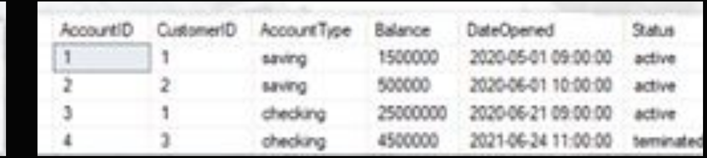
DimBranch





account_id	customer_id	account_type	balance	date_opened	status
1	1	saving	1500000	2020-05-01 09:00:00	active
2	2	saving	500000	2020-06-01 10:00:00	active
3	1	checking	25000000	2020-06-21 09:00:00	active
4	3	checking	4500000	2021-06-24 11:00:00	terminated

Input

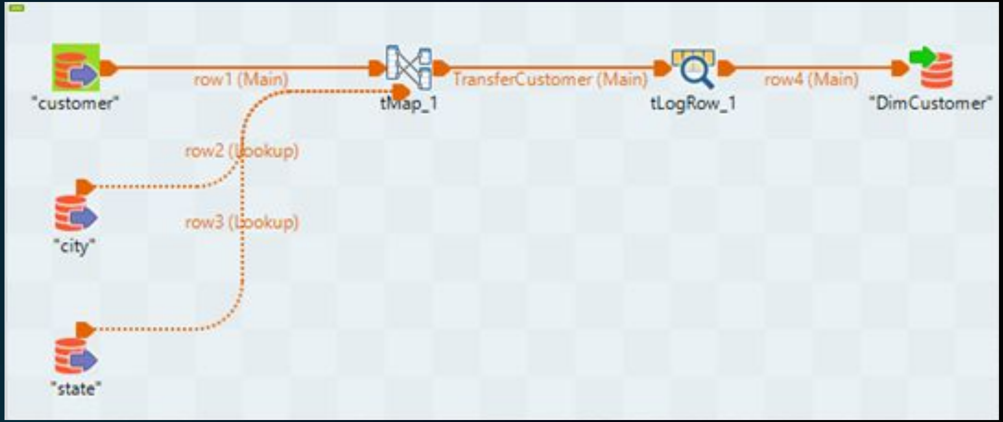


AccountID	CustomerID	AccountType	Balance	DateOpened	Status
1	1	saving	1500000	2020-05-01 09:00:00	active
2	2	saving	500000	2020-06-01 10:00:00	active
3	1	checking	25000000	2020-06-21 09:00:00	active
4	3	checking	4500000	2021-06-24 11:00:00	terminated

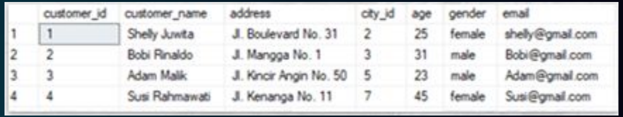
Output

3. ETL Job for DimCustomer Table

DimBranch

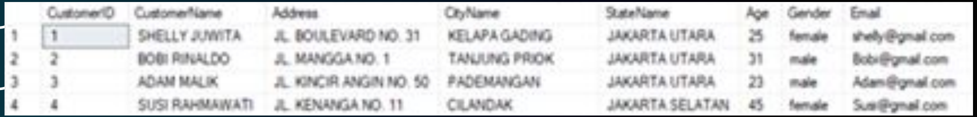


Input



	customer_id	customer_name	address	city_id	age	gender	email
1	1	Shelly Juwita	Jl. Boulevard No. 31	2	25	female	shelly@gmail.com
2	2	Bobi Rinaldo	Jl. Mangga No. 1	3	31	male	Bobi@gmail.com
3	3	Adam Malik	Jl. Kincir Angin No. 50	5	23	male	Adam@gmail.com
4	4	Susi Rahmawati	Jl. Kenanga No. 11	7	45	female	Susi@gmail.com

Output



	CustomerID	CustomerName	Address	CityName	StateName	Age	Gender	Email
1	1	SHELLY JUWITA	JL. BOULEVARD NO. 31	KELAPA GADING	JAKARTA UTARA	25	female	shelly@gmail.com
2	2	BOBI RINALDO	JL. MANGGA NO. 1	TANJUNG PRIOK	JAKARTA UTARA	31	male	Bobi@gmail.com
3	3	ADAM MALIK	JL. KINCIR ANGIN NO. 50	PADEMANGAN	JAKARTA UTARA	23	male	Adam@gmail.com
4	4	SUSI RAHMAWATI	JL. KENANGA NO. 11	CILANDAK	JAKARTA SELATAN	45	female	Susi@gmail.com

ETL Workflow Explanation

1. Input:

The process begins by integrating data from **three source tables**:

- **Customer Table:** Contains details such as `customer_id`, `customer_name`, `address`, `age`, `gender`, and `email`.
- **City Table:** Includes `city_id` and `city_name`.
- **State Table:** Includes `state_id` and `state_name`.

2. Transformation Using tMap:

- **Merging Data:**
  - The `tMap` component is used to join the three source tables (Customer, City, and State) using their respective keys.
  - Data from the `customer` table is enriched with `city_name` and `state_name` based on relationships in the `city` and `state` tables.
- **Data Formatting:**
  - Converts all text columns to **uppercase** (except `CustomerID`, `Age`, and `Email`).

3. Output:

The transformed data is loaded into the **DimCustomer table**, containing:

- `CustomerID`
- `CustomerName`
- `Address`
- `CityName`
- `StateName`
- `Age`
- `Gender`
- `Email`

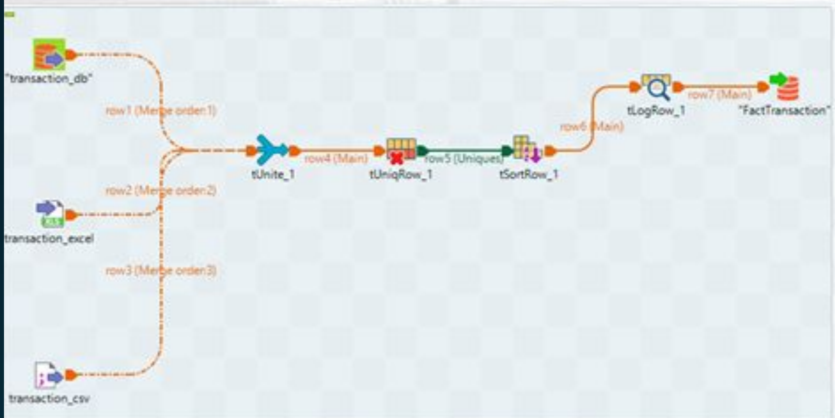
03

Create ETL Job for Fact Table

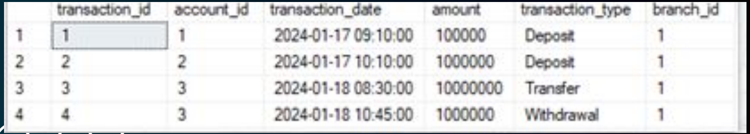
talend

ETL Job for FactTransaction Table

FactTransaction

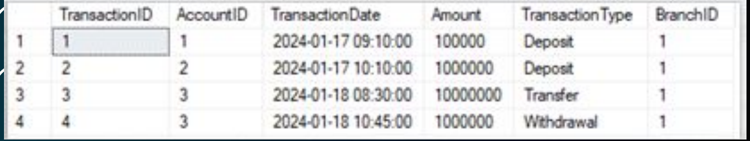


Input



	transaction_id	account_id	transaction_date	amount	transaction_type	branch_id
1	1	1	2024-01-17 09:10:00	100000	Deposit	1
2	2	2	2024-01-17 10:10:00	1000000	Deposit	1
3	3	3	2024-01-18 08:30:00	10000000	Transfer	1
4	4	3	2024-01-18 10:45:00	1000000	Withdrawal	1

Output



	TransactionID	AccountID	TransactionDate	Amount	TransactionType	BranchID
1	1	1	2024-01-17 09:10:00	100000	Deposit	1
2	2	2	2024-01-17 10:10:00	1000000	Deposit	1
3	3	3	2024-01-18 08:30:00	10000000	Transfer	1
4	4	3	2024-01-18 10:45:00	1000000	Withdrawal	1

ETL Workflow for FactTransaction

1. Input:

Data is sourced from three locations:

- **`transaction_db`**: A table in the database.
- **`transaction_excel`**: An Excel file.
- **`transaction_csv`**: A CSV file.

2. Transformation Using Talend Components:

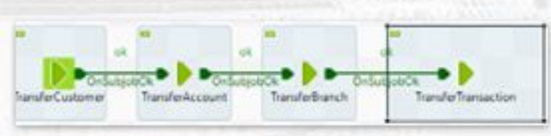
1. **tUnite**:
   - Merges data from the three sources (`transaction_db`, `transaction_excel`, and `transaction_csv`) into a single unified flow.
   - Ensures that data from all sources is handled in one pipeline.
2. **tUniqRow**:
   - Removes duplicate rows by ensuring `transaction_id` is unique.
   - Prevents redundancy in the FactTransaction table.
3. **Text Formatting**:
   - Converts **all text fields** (e.g., `TransactionType`) to **uppercase** to ensure uniformity.
   - Columns such as `TransactionID`, `AccountID`, `TransactionDate`, `BranchID`, and `Amount` are left unchanged.
4. **tSortRow**:
   - Sorts the data by `transaction_id` in ascending order.
   - Ensures the output data is consistent and structured.

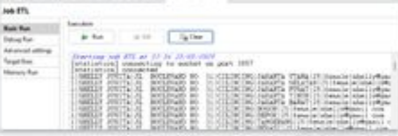
3. Output:

The cleaned, formatted, and transformed data is loaded into the `FactTransaction` table. The table schema includes:

- **`TransactionID`**: Unique identifier for each transaction.
- **`AccountID`**: The account involved in the transaction.
- **`TransactionDate`**: The date and time of the transaction.
- **`Amount`**: The monetary value of the transaction.
- **`TransactionType`**: The type of transaction (formatted in UPPERCASE, e.g., DEPOSIT, WITHDRAWAL).
- **`BranchID`**: The branch where the transaction occurred.

ETL Orchestration





ETL Orchestration Explanation

ETL Orchestration refers to automating and managing the sequence of ETL jobs to ensure a smooth data integration process from various sources into a Data Warehouse. Below are the details based on the provided visual:

1. Function

- Automates the entire ETL process by organizing and executing multiple ETL jobs in a predefined sequence.
- Ensures data dependencies are handled correctly (e.g., dimension tables must be loaded before fact tables).

2. Key Components

1. **tRunJob**:
   - Executes sub-jobs in the main orchestration workflow.
   - Calls individual ETL jobs (e.g., `TransferCustomer`, `TransferAccount`, etc.) in a specified order.
2. **OnSubjobOK**:
   - Acts as an inter-job trigger.
   - Ensures that the next ETL job runs only after the successful completion of the previous job.

ETL Orchestration

ETL Job Details

Each ETL job is responsible for specific transformations and data loading tasks:

1. **ETL Job for DimCustomer**:
   - Combines customer, city, and state data.
   - Converts text columns to uppercase (except IDs, age, and email).
2. **ETL Job for DimAccount**:
   - Maps customers to accounts.
   - Processes account type, balance, and account status.
3. **ETL Job for DimBranch**:
   - Transfers branch data such as branch names and locations.
4. **ETL Job for FactTransaction**:
   - Merges data from multiple sources (CSV, Excel, database).
   - Removes duplicates, sorts data, and ensures proper formatting.

Output

The final outputs include:

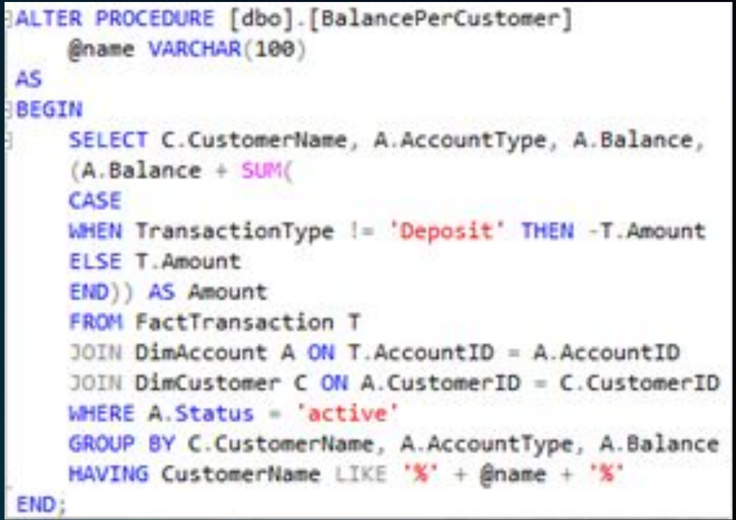
1. **DimCustomer**: Enriched and formatted customer data.
2. **DimAccount**: Cleaned and mapped account data.
3. **DimBranch**: Organized branch details.
4. **FactTransaction**: Consolidated transactional records.

04

Create Stored Procedure

Store Procedure

DailyTransaction Stored Procedure



```
ALTER PROCEDURE [dbo].[BalancePerCustomer]
    @name VARCHAR(100)
AS
BEGIN
    SELECT C.CustomerName, A.AccountType, A.Balance,
    (A.Balance + SUM(
    CASE
    WHEN TransactionType != 'Deposit' THEN -T.Amount
    ELSE T.Amount
    END)) AS Amount
    FROM FactTransaction T
    JOIN DimAccount A ON T.AccountID = A.AccountID
    JOIN DimCustomer C ON A.CustomerID = C.CustomerID
    WHERE A.Status = 'active'
    GROUP BY C.CustomerName, A.AccountType, A.Balance
    HAVING CustomerName LIKE '%' + @name + '%'
END;
```

1. DailyTransaction

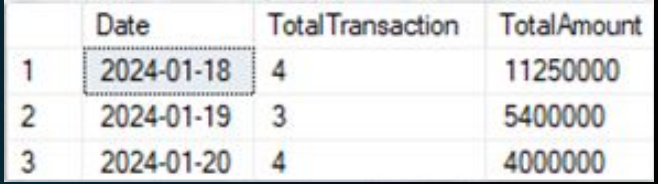
Purpose:

Calculates the total number of transactions (`TotalTransaction`) and their total amounts (`TotalAmount`) for each day within a specified date range.

Call Store Procedure:

```
EXEC DailyTransaction @start_date = '2024-01-18', @end_date = '2024-01-21';
```

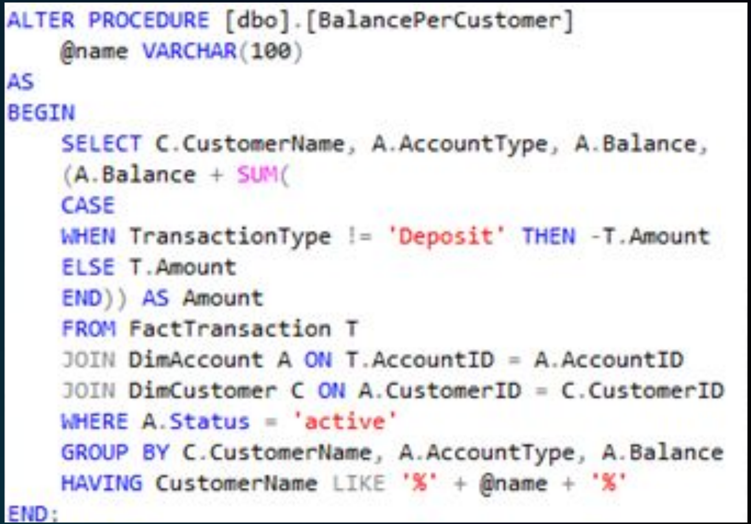
Output



	Date	TotalTransaction	TotalAmount
1	2024-01-18	4	11250000
2	2024-01-19	3	5400000
3	2024-01-20	4	4000000

Store Procedure

BalancePerCustomer Stored Procedure



```
ALTER PROCEDURE [dbo].[BalancePerCustomer]
    @name VARCHAR(100)
AS
BEGIN
    SELECT C.CustomerName, A.AccountType, A.Balance,
    (A.Balance + SUM(
    CASE
    WHEN TransactionType != 'Deposit' THEN -T.Amount
    ELSE T.Amount
    END)) AS Amount
    FROM FactTransaction T
    JOIN DimAccount A ON T.AccountID = A.AccountID
    JOIN DimCustomer C ON A.CustomerID = C.CustomerID
    WHERE A.Status = 'active'
    GROUP BY C.CustomerName, A.AccountType, A.Balance
    HAVING CustomerName LIKE '%' + @name + '%'
END;
```

2. BalancePerCustomer

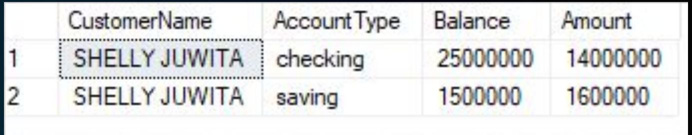
Purpose:

1. Computes the remaining balance for each customer by considering transactions.
2. Accounts for whether a transaction is a deposit (adds to the balance) or withdrawal/other (deducts from the balance).

Call Store Procedure:

```
EXEC [BalancePerCustomer] @name = 'Shelly';
```

Output



	CustomerName	AccountType	Balance	Amount
1	SHELLY JUWITA	checking	25000000	14000000
2	SHELLY JUWITA	saving	1500000	1600000

THANKS

Do you have any questions?

hijirdw@gmail.com
https://github.com/hijirdella