IPL Cricket Match Data Integration

Project Title: SSIS Final Project – IPL Cricket Match Data Integration

Introduction

The Indian Premier League (IPL) is a premier Twenty20 cricket league, celebrated for its thrilling matches and vast global appeal. Its franchise-based model and remarkable talent not only shape modern cricket but also revolutionize its economic landscape, inspiring a new generation of players and fans alike.

This project aims to create an ETL pipeline using SQL Server Integration Services (SSIS) to structure IPL match data for analysis. The pipeline will support informed decision-making and enhance stakeholder engagement, leading to improved strategic insights and operational outcomes.

Submitted by
Khushi Bhavsar

Table of Contents

Contents

Project Title: SSIS Final Project – IPL Cricket Match Data Integration 1

Introduction 1

Table of Contents 2

Project Requirements 3

Key Features of Implement 3

Development Guidelines 3

Deliverables 3

Data Model Design 4

Key Components of the Data Model 4

Relationships 6

ETL Process 8

Configuring Data Flow Tasks for Tables 9

Incremental Loading 10

Data Quality / Error Handling 11

Foreign Key & Self-Lookup Validation 11

Handling No Match Output (Invalid Foreign Keys) 11

BadDataLog Table (SQL Server) 11

Tools & Techniques Used 11

Challenges and Solutions 12

Challenge 1: `City_Id` Contained String Values Instead of Integers in `dim_Venue` 12

Challenge 2: No Unique Keys in Some Source Tables 12

Project Requirements

The objective of this project is to develop a robust ETL pipeline using SQL Server Integration Services (SSIS) for the integration and analysis of IPL cricket match data. The pipeline will transform raw ball-by-ball data into an organized and analysis-ready format to support stakeholders in making informed decisions.

Key Features of Implement

1. **Data Validation**
   - Ensure the accuracy, completeness, and reliability of the incoming data.

2. **Bad Data Handling**
   - Identify and segregate invalid or incomplete records for analysis and resolution.

3. **Data Transformation**
   - Apply business rules to convert raw data into a structured and meaningful format.

4. **Incremental Loading**
   - Enable efficient updates by loading only new or modified data into the target system.

5. **Error and Success Handling**
   - Implement mechanisms to capture and log errors, as well as record successful operations.

6. **Logging**
   - Maintain a detailed log of all ETL processes for monitoring and troubleshooting.

7. **Scheduling**
   - Automate the ETL pipeline to run at predefined intervals for timely data availability.

Development Guidelines

Follow SSIS Development Best Practices

- Design modular, maintainable packages with clear error handling and logging mechanisms.
- Use configuration files or parameters to manage environment-specific settings.
- Implement incremental loading and auditing for reliable and efficient ETL processes.

Deliverables

1. **SSIS Packages:**
   - Fully functional ETL packages for data extraction, transformation, and loading.

2. **Relational Database Schema:**
   - A well-defined schema representing the IPL cricket match data model.

Data Model Design

The data model for this project is designed to effectively store and manage IPL cricket match data. It represents a star schema, optimized for analytical querying and business intelligence use cases. The model consists of fact and dimension tables, ensuring both data normalization and efficient querying for insightful analysis.

Key Components of the Data Model

1. **Fact Tables:**

   - **fact_Ball_by_Ball:** Detailed ball-by-ball match events including players and outcomes.
   - **fact_Match:** Aggregated match results and key statistics.
   - **fact_Batsman_Scored:** Runs scored by batsmen per ball and over.
   - **fact_Player_Match:** Individual player performance per match.
   - **fact_Wicket_Taken:** Records of wickets including dismissal details.

2. **Dimension Tables:**

   - **dim_Player:** Player personal and skill attributes.
   - **dim_Team:** Team identity and related info.
   - **dim_Venue:** Match venue details with location info.
   - **dim_Bowling_Style & dim_Batting_Style:** Player bowling and batting style categories.
   - **dim_Out_Type:** Types of dismissals.
   - **dim_City & dim_Country:** Geographic locations for matches and venues.
   - **dim_Toss_Decision:** Toss choices and decisions in matches.

3. **Relationships and Integrity:**

   The model connects fact tables to dimension tables using keys, keeping the data accurate and allowing detailed analysis.

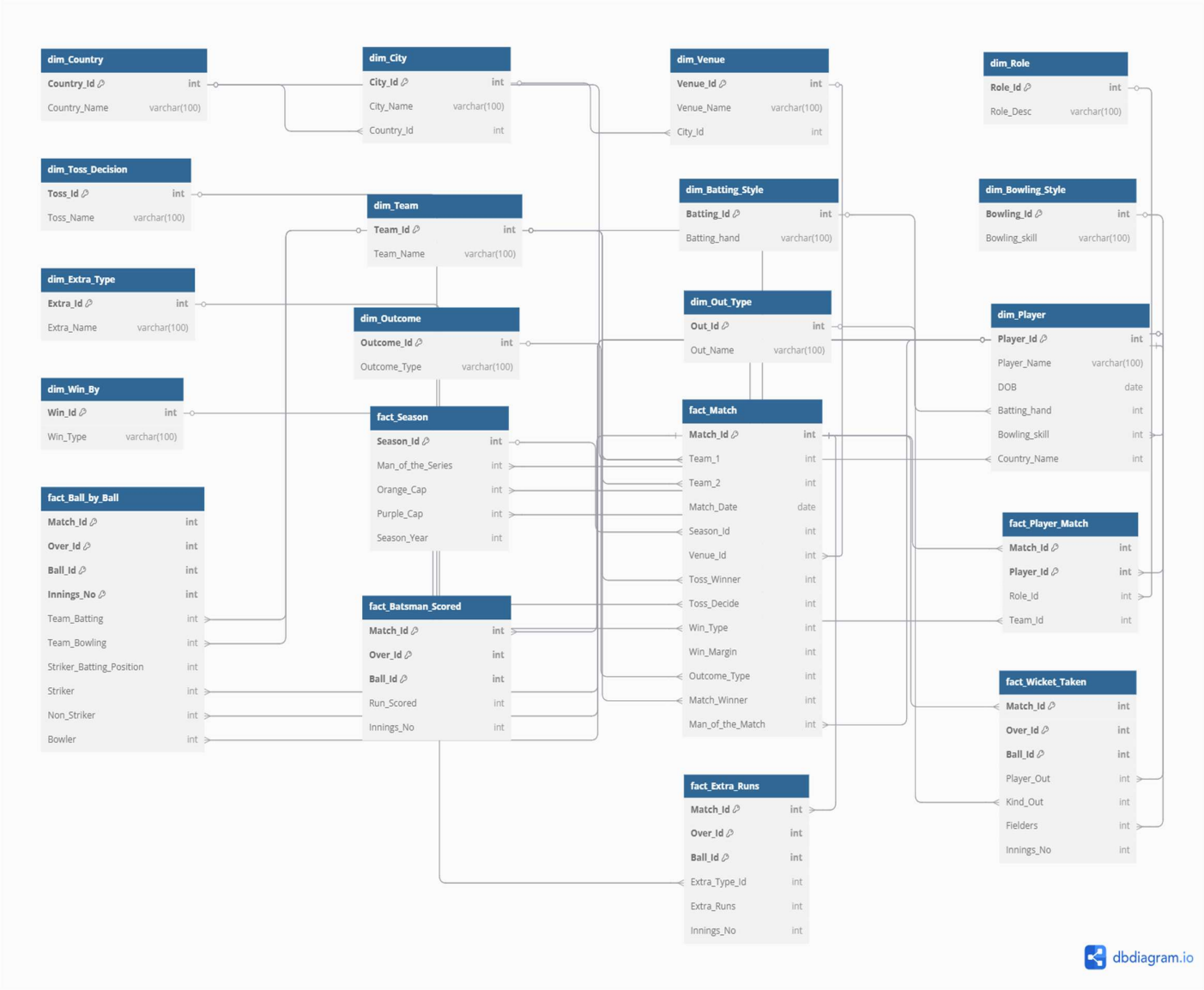

Figure 1 Data Model

Relationships

The IPL Cricket Match Database keeps data organized and accurate by using special keys and clear connections between tables:

1. **Primary Keys (PK):** Each table has a unique ID (like Match_Id or Player_Id) to identify every record clearly.
2. **Foreign Keys (FK):** These link tables together, showing how data is related. For example:
    - Match.Team_1 and Match.Team_2 point to Team.Team_Id.
    - Ball_by_Ball.Match_Id points to Match.Match_Id.
3. **One-to-Many Relationships:** One record (like a match) connects to many related records (like balls played or player stats).
4. **Many-to-Many Relationships:** These are handled by special tables (like Player_Match) that connect players with matches to track who played and their roles.
5. **Referential Integrity:** The database makes sure that all links are valid — for example, a match can't refer to a team that doesn't exist.

Table	Columns	Data Type	Key (Reference: Tablename.TableColumn)
Match	Match_Id	Int	Primary Key
	Team_1	Int	Foreign key (Team.Team_Id)
	Team_2	Int	Foreign key (Team.Team_Id)
	Match_Date	Date	
	Season_Id	Int	Foreign key (Season.Season_Id)
	Venue_Id	Int	Foreign key (Venue.Venue_Id)
	Toss_Winner	Int	Foreign key (Team.Team_Id)
	Toss_Decide	Int	Foreign key (Toss_Decision.Toss_Id)
	Win_Type	Int	Foreign key (Win_By.Win_Id)
	Outcome_Type	Int	Foreign key (Outcome.Outcome_Id)
	Match_Winner	Int	Foreign key (Team.Team_Id)
	Man_of_the_Match	Int	Foreign key (Player.Player_Id)
Ball_by_Ball	Match_Id	Int	Foreign key (Match.Match_Id)
	Over_Id	Int	
	Ball_Id	Int	
	Innings_No	Int	
	Team_Batting	Int	Foreign key (Team.Team_Id)
	Team_Bowling	Int	Foreign key (Team.Team_Id)
	Striker_Batting_Position	Int	
	Striker	Int	Foreign key (Player.Player_Id)
	Non_Striker	Int	Foreign key (Player.Player_Id)
	Bowler	Int	Foreign key (Player.Player_Id)
Batsman_Scored	Match_Id	Int	Foreign key (Match.Match_Id)
	Over_Id	Int	
	Ball_Id	Int	
	Run_Scored	Int	
	Innings_No	Int	
Player_Match	Match_Id	Int	Foreign key (Match.Match_Id)
	Player_Id	Int	Foreign key (Player.Player_Id)
	Role_Id	Int	Foreign key (Role.Role_Id)
	Team_Id	Int	Foreign key (Team.Team_Id)

Extra_Runs	Match_Id	Int	Foreign key (Match.Match_Id)
	Over_Id	Int	
	Ball_Id	Int	
	Extra_Type_Id	Int	Foreign key (Extra_type.Extra_Id)
	Extra_Runs	Int	
	Innings_No	Int	
Wicket_Taken	Match_Id	Int	Foreign key (Match.Match_Id)
	Over_Id	Int	
	Ball_Id	Int	
	Player_Out	Int	Foreign key (Player.Player_Id)
	Kind_Out	Int	Foreign key (Out_Type.Out_Id)
	Fielders	Int	Foreign key (Player.Player_Id)
	Innings_No	Int	
Country	Country_Id	Int	Primary Key
	Country_Name	varchar	
City	City_Id	Int	Primary Key
	City_Name	Varchar	
	Country_Id	Int	Foreign key (Country.Country_Id)
Venue	Venue_Id	Int	Primary key
	Venue_Name	Varchar	
	City_Id	Int	Foreign key (City.City_Id)
Role	Role_Id	Int	Primary key
	Role_Desc	Varchar	
Toss_Decision	Toss_Id	Int	Primary key
	Toss_Name	Varchar	
Team	Team_Id	int	Primary key
	Team_Name	Varchar	
Batting_Skill	Batting_Id	Int	Primary key
	Batting_hand	Varchar	
Bowling_Style	Bowling_Id	int	Primary key
	Bowling_skill	Varchar	
Extra_Type	Extra_Id	int	Primary key
	Extra_Name	Varchar	
Outcome	Outcome_Id	int	Primary key
	Outcome_type	Varchar	
Out_Type	Out_Id	int	Primary key
	Out_Name	Varchar	
Player	Player_Id	Int	Primary key
	Player_Name	Varchar	
	DOB	Date	
	Batting_hand	Int	Foreign key (Batting_skill.Batting_Id)
	Bowling_skill	Int	Foreign key (Bowling_Style.Bowling_Id)
	Country_Name	Int	Foreign key (Country.Country_Id)
Win_By	Win_Id	Int	Primary key
	Win_Type	Varchar	
Season	Season_Id	int	Primary key
	Man_of_the_Series	int	
	Orange_Cap	Int	Foreign key (Player.Player_Id)
	Purple_Cap	Int	Foreign key (Player.Player_Id)
	Season_year	Int	Foreign key (Player.Player_Id)

ETL Process

Extract

- **Source Files**: 20 CSV files from the **IPL dataset**.
- Each file corresponds to a dimension or fact table such as dim_Player.csv, fact_Match.csv, etc.
- Files were read using **Flat File Source** components in SSIS.

Transform

Applied a variety of data transformation techniques:

- **Data Conversion**: Used to cast data types to match the destination table schema (e.g., converting DT_STR to DT_WSTR or DT_I4).
- **Derived Column**: Used for creating new columns or transforming values (e.g., converting city names to city IDs).
- **Conditional Split**: Implemented logic to route valid vs invalid data rows (e.g., NULLs or mismatched foreign keys) to appropriate destinations.
- **Lookup Transformation**: Used to resolve foreign keys like Team, Player, Venue from dimension tables.

Load

- Final clean data was loaded into corresponding dimension and fact tables in SQL Server, under the ipl schema.
- **OLE DB Destination** was used to insert data into each target table.
- **Error Rows** were redirected to a flat file or inserted into a custom error logging table (BadDataLog) for traceability.

Tools & Technologies Used

- **Software**:
    - Microsoft SQL Server Management Studio (SSMS)
    - Visual Studio with SQL Server Data Tools (SSDT) and SSIS
- **SSIS Components**:
    - Flat File Source
    - Data Conversion
    - Derived Column
    - Conditional Split
    - OLE DB Destination

Configuring Data Flow Tasks for Tables

In the ETL pipeline, individual **Data Flow Tasks** were created for each table to ensure a structured and efficient data integration process. The tasks were divided into two distinct stages:

Loading Dimension Tables:

Data Flow Tasks for all dimension tables were configured first, as dimension tables provide contextual information for the fact tables.

Each task was appropriately named to maintain clarity and align with best practices.

Loading Fact Tables:

Following the dimensions, Data Flow Tasks for fact tables were added. These tasks handle larger datasets with transactional information, such as match details, ball-by-ball data, and player statistics.

Fact table tasks were also clearly labelled.

Purpose and Organization

This sequential approach ensures that dimension data is loaded before fact data, maintaining referential integrity between tables. Each Data Flow Task is modular, promoting better management, debugging, and scalability of the pipeline.

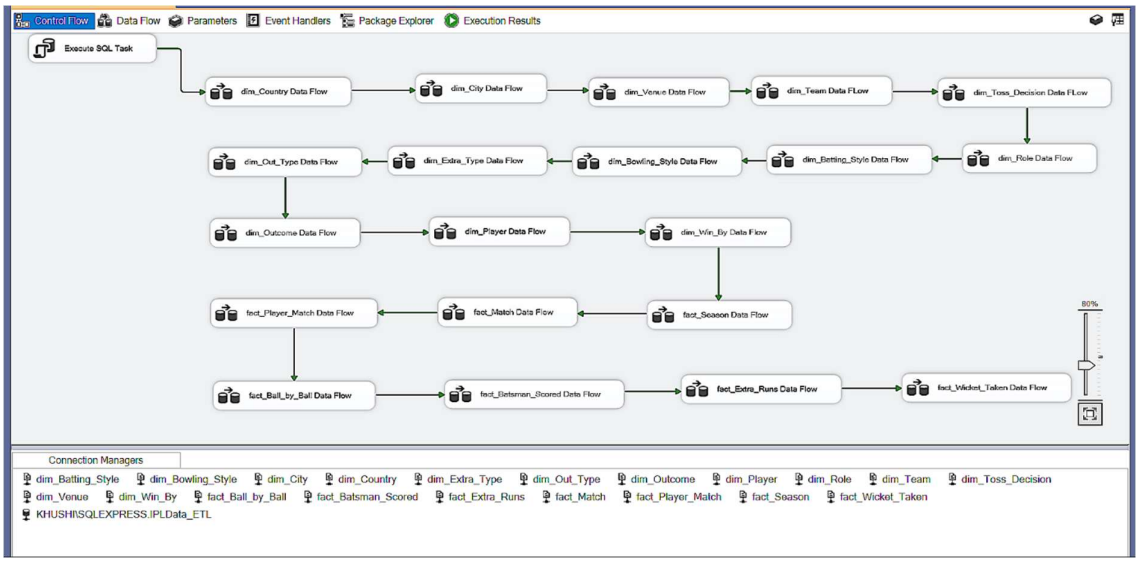

Figure 2 Pipeline Architecture

Incremental Loading

To optimize performance and avoid loading duplicate data, Lookup Transformations were implemented in the SSIS package to support incremental data loading. This approach ensures only new or updated records are processed.

Lookup Configuration

1. No Match Output

- Purpose: Detects new records that are not already present in the destination table.
- Configuration:
  - The Lookup transformation is set to compare incoming source data against the destination table using a unique key column (e.g., Player_Id, Match_Id, etc.).
  - If no match is found:
    - The row is considered a new record.
    - It is redirected to an OLE DB Destination, where it is inserted into the target table.
- Result: Ensures that only new records are inserted, maintaining data integrity and preventing duplicates.

2. Match Output

- Purpose: Detects existing records that may require an update.
- Configuration:
  - If a match is found based on the lookup key:
    - The row is redirected to an OLE DB Command.
    - This component executes a inline SQL UPDATE statement to update the corresponding record in the target table.
- Result: Enables dynamic and efficient updates to existing records when changes are detected in the source data.

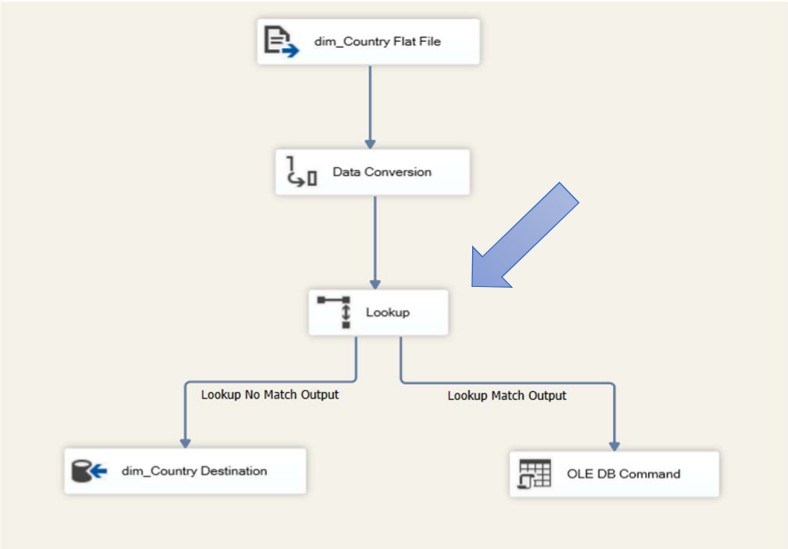

Figure 3 Incremental Loading

Data Quality / Error Handling

To maintain data integrity and prevent referential issues in the data warehouse, a robust error handling mechanism was implemented using **Lookup transformations** and logging invalid records to a dedicated log.

Foreign Key & Self-Lookup Validation

- **Lookup Transformations** were used to validate incoming foreign keys against existing dimension tables (e.g., dim_Player, dim_Venue).
- **Self-Lookup** was also applied (e.g., for fields referring to the same table) to ensure consistent relationships.

Handling No Match Output (Invalid Foreign Keys)

- When the Lookup transformation fails to find a match (e.g., foreign key not found in the dimension), that row is redirected to an error output.
- These invalid rows are not discarded, but instead logged for review and future correction.

BadDataLog Table (SQL Server)

- A dedicated table named BadDataLog is used to capture and store all invalid data.
- Each record logs:
  - table_name: The name of the target table where the error occurred
  - error_description: A custom message detailing the issue (e.g., "Player ID not found in dim_Player")
  - error_data: The full record (or relevant values) in string format
  - log_date: Timestamp when the error was logged

```
-- Bad Data Logging Table
CREATE TABLE BadDataLog(
log_id INT IDENTITY(1,1) PRIMARY KEY,
table_name NVARCHAR(100) NOT NULL,
error_description NVARCHAR(max) NOT NULL,
error_data NVARCHAR(MAX) NOT NULL,
log_date DATETIME DEFAULT GETDATE()
);
```

Figure 4 BadDataLog Table

Tools & Techniques Used

- **SSIS Components**: Lookup, OLE DB Command or OLE DB Destination (for logging to SQL)

Challenges and Solutions

Challenge 1: `City_Id` Contained String Values Instead of Integers in `dim_Venue`

Problem: In the dim_Venue table, the City_Id column sometimes had string values like "Mohali" or "Chepauk" instead of expected integer IDs. This caused lookup failures and foreign key violations during data loading.

Solution:

- Used a Derived Column to map city names to their correct numeric IDs.

  Example:
  (TRIM(City_Id) == "Mohali" ? 2 : TRIM(City_Id) == "Uppal" ? 7 : TRIM(City_Id) == "Chepauk" ? 8 : TRIM(City_Id) == "Motera" ? 17 : TRIM(City_Id) == "Jamtha" ? 19 : 0)

  RIGHT(City_Id,LEN(City_Id) - FINDSTRING(City_Id,",",1))

- Rows with unrecognized city names were redirected using Lookup's No Match Output to a BadDataLog table in SQL.
- Temporarily disabled foreign key constraints to allow investigation and correction.
- After cleaning, re-enabled constraints and reprocessed valid data.

Outcome: Fixed incorrect city values, ensured smooth data load, and maintained referential integrity.

Challenge 2: No Unique Keys in Some Source Tables

Problem: Tables like Ball_by_Ball, Player_Match, and Wicket_Taken did not have a clear unique identifier, which made matching and validating rows difficult.

Solution:

- Used Lookup Transformation with multiple columns (e.g., Match_Id, Over_Id, Ball_Id, Innings_No) for uniqueness.
- Adopted an Insert-only incremental load to avoid unnecessary updates.
- Filtered out duplicates and invalid records during transformation.

Outcome: Enabled accurate data loads even without primary keys, improved performance, and ensured consistency.