**League of Legends Skin Sales Data Warehouse**

**Project Documentation**

---

## OVERVIEW

**Goal:** Build a production-grade data warehouse for analyzing League of Legends skin sales with advanced data quality framework.

**Architecture:** PostgreSQL Medallion (Bronze → Silver → Gold)
**Data Source:** Riot Games Data Dragon API + LoL Wiki (web scraping)
**Key Feature:** 10% intentional errors in sales data to demonstrate DQ monitoring

---

## ARCHITECTURE

### Medallion Architecture

#### BRONZE (Staging Layer)

- Purpose: Raw data staging - accept everything
- Tables: stg_skins, stg_players, stg_sales
- Data Types: NUMERIC, TEXT (to accept invalid data)

#### SILVER (Cleansed Layer)

- Purpose: Validated and cleaned data
- Tables: dim_date, dim_skin, dim_player, fact_sale
- Additional: fact_sale_quarantine, data_quality_log
- Process: Validation, cleansing, error detection

#### GOLD (Curated Layer)

- Purpose: Star schema for analytics
- Tables: Final dimensions and facts
- Data: Only valid records (is_valid = TRUE)

---

## DATA MODEL

### Star Schema (Gold Layer)

**Fact Table:**

- fact_sales: Grain = one purchase transaction

**Dimension Tables:**

- dim_player: Player attributes (region, segment, account_age)

- dim_skin: Skin attributes (champion, rarity, price_rp)
- dim_date: Calendar dimension (year, quarter, month, weekend flags)

**Relationships:**

- fact_sales.player_key → dim_player.player_key
- fact_sales.skin_key → dim_skin.skin_key
- fact_sales.date_key → dim_date.date_key

**Data Volumes**

**Dimensions:**

- dim_player: 5,000 records
- dim_skin: 1,474 records
- dim_date: 7,671 records (2010-01-01 to 2030-12-31)

**Facts:**

- fact_sales: 20,200 transactions
    - Valid: 18,324 (90.7%)
    - Invalid: 1,876 (9.3%)

---

**DATA SOURCES**

**1. Riot Data Dragon API**

**Endpoint:** https://ddragon.leagueoflegends.com/cdn/14.1.1/data/en_US/champion.json

**Method:** REST API call via Python requests library

**Script:** fetch_skins.py

**Output:** ddragon_skins.csv (approximately 2,000 skin records)

**Content:**

- Skin names
- Champion IDs
- Skin numbers

**Limitations:**

- No pricing information
- No rarity classification
- No release dates

**2. LoL Wiki (Web Scraping)**

**URL:** https://leagueoflegends.fandom.com/wiki/Module:SkinData/data?action=edit

**Method:** Playwright headless browser automation

**Reason for Playwright:** Direct HTTP requests blocked with 403 Forbidden

**Script:** scrape_wiki.py → parse_skins_from_wiki.py

**Output:** wiki_skins_clean.csv (approximately 1,700 skin records)

**Content:**

- Skin prices (Riot Points)
- Rarity classifications
- Release dates (real dates from Wiki)

**Data Extraction:**

- Source: Lua module code from Wiki edit view
- Parsing: Regular expressions to extract Lua table structure
- Normalization: String cleaning, rarity mapping

**3. Data Merge Process**

**Script:** merge_skins.py

**Input Files:**

- ddragon_skins.csv (from API)
- wiki_skins_clean.csv (from Wiki)

**Process:**

1. Filter event-exclusive skins (Prestige, Victorious, After Hours)
2. Preserve esports skins (T1, DRX - purchasable during events)
3. Normalize skin names (remove special characters, lowercase)
4. Match records by normalized name
5. Handle Default skins separately (set price = 0 RP)
6. Assign sequential skin_id

**Output:** dim_skins_final.csv

**Match Statistics:**

- Matched: 1,470 records (84.9%)
- Unmatched: 262 records (removed - newer skins not in Wiki)

- Total output: 1,474 records

**Critical Fix:** Default skins originally showed 880 RP (champion cost) from Wiki. Fixed to 0 RP as they are free.

---

**DATA QUALITY FRAMEWORK**

**Components**

**1. Quarantine Table (fact_sale_quarantine)**

Purpose: Store all rejected records for manual review

Columns:

- quarantine_id (SERIAL PRIMARY KEY)
- transaction_id, player_id, skin_id, purchase_date
- price_rp, quantity
- rejection_reason (TEXT - comma-separated issues)
- rejection_timestamp
- source_system

**2. Audit Log (data_quality_log)**

Purpose: Historical tracking of all data quality checks

Columns:

- log_id (SERIAL PRIMARY KEY)
- run_timestamp
- table_name
- issue_type
- issue_count
- issue_pct (percentage)
- threshold_pct (5.0%)
- alert_triggered (BOOLEAN)

**3. Monitoring Process**

Checks performed:

1. Missing player_key
2. Missing skin_key
3. Invalid price_rp (NULL or negative)

4. Invalid quantity (NULL or ≤0)
5. Total invalid records

Alert threshold: 5.0%

Alert trigger: When any check exceeds 5% error rate

**4. Cleansing Rules**

Rules applied automatically:

FIXABLE:

- Missing price_rp: Lookup from dim_skin using skin_key
- Incorrect total_rp: Recalculate as price_rp * quantity

NON-FIXABLE (rejected to quarantine):

- NULL or invalid player_id
- NULL or invalid skin_id
- Negative prices (no valid source)
- Zero quantity
- Invalid dates (not in dim_date range)

Validation criteria for is_valid = TRUE:

- player_key IS NOT NULL
- skin_key IS NOT NULL
- price_rp > 0
- quantity > 0
- total_rp = price_rp * quantity
- date_key exists in dim_date

---

**PROJECT STRUCTURE**

LOLDW/

```
├── data/
│   ├── raw/
│   │   ├── skindata_raw.lua
│   │   ├── ddragon_skins.csv
│   │   ├── wiki_skins_clean.csv
│   │   ├── dim_skins_final.csv
│   │   ├── dim_player.csv
│   │   └── fact_sales.csv
│   │
│   └── src_data/
│       ├── fetch_skins.py
│       ├── scrape_wiki.py
│       ├── parse_skins_from_wiki.py
│       ├── merge_skins.py
│       └── generate_players_sales.py
│
└── sql/
    ├── 00_reset_database.sql
    ├── 01_bronze_layer_NEW.sql
    └── 02_silver_SIMPLE.sql
```

---

**EXECUTION PIPELINE**

**Phase 1: Data Collection**

Execute in order:

cd LOLDW

python data/src_data/fetch_skins.py

# Output: data/raw/ddragon_skins.csv

python data/src_data/scrape_wiki.py

# Output: data/raw/skindata_raw.lua

python data/src_data/parse_skins_from_wiki.py

# Output: data/raw/wiki_skins_clean.csv

python data/src_data/merge_skins.py

# Output: data/raw/dim_skins_final.csv

python data/src_data/generate_players_sales.py

# Output: data/raw/dim_player.csv, data/raw/fact_sales.csv

**Phase 2: Database Build**

Execute in PostgreSQL:

-- Step 1: Clean slate

\i sql/00_reset_database.sql

-- Step 2: Create Bronze tables

\i sql/01_bronze_layer_NEW.sql

-- Step 3: Import CSV files

-- Use DataGrip or pgAdmin Import/Export Data feature

-- Import into: bronze.stg_skins, bronze.stg_players, bronze.stg_sales

-- Step 4: Build Silver layer with DQ framework

\i sql/02_silver_SIMPLE.sql

---

**DATA QUALITY RESULTS**

**Monitoring Summary**

| Check Type | Count | Error % | Status | Alert |
|---|---|---|---|---|
| Missing Player | 245 | 1.21 | OK | No |
| Missing Skin | 238 | 1.18 | OK | No |
| Invalid Price | 225 | 1.11 | OK | No |
| Invalid Quantity | 221 | 1.09 | OK | No |
| **Total Invalid** | **1,876** | **9.27** | **FAIL** | **YES** |

**Quarantine Analysis**

Top rejection reasons:

- missing_player: 245 records
- missing_skin: 238 records
- negative_price: 225 records
- invalid_quantity: 221 records
- Combined issues: 947 records

**Revenue Statistics (Valid Sales Only)**

- Total valid transactions: 18,324
- Total revenue: 24,567,890 RP
- Average transaction value: 1,341 RP
- Revenue by segment:
    - Whale: 45%
    - Core: 35%
    - Casual: 20%

---

**BUSINESS USE CASES**

**1. Revenue Analysis**

- Temporal trends (daily, weekly, monthly, quarterly)
- Weekend vs weekday sales patterns
- Month-end purchasing behavior

- Seasonal variations

**2. Player Segmentation**

- Lifetime Value (LTV) by segment
- RFM analysis (Recency, Frequency, Monetary)
- Segment migration patterns
- Churn prediction inputs

**3. Skin Performance**

- Top-selling champions
- Rarity tier performance (Legacy vs Epic vs Legendary)
- Price elasticity analysis
- New release performance tracking

**4. Pricing Strategy**

- Optimal price point identification
- Discount impact analysis
- Bundle opportunity detection
- Price sensitivity by segment

**5. Regional Insights**

- Regional preference patterns (EU vs NA vs KR)
- Regional revenue contribution
- Regional skin catalog gaps
- Localization opportunities

---

**TECHNICAL SPECIFICATIONS**

**Technologies**

**Database:**

- PostgreSQL 18
- Schemas: bronze, silver, gold

**Python Environment:**

- Python 3.x
- Libraries: pandas, requests, playwright, re, datetime, random

**Development Tools:**

- DataGrip (SQL IDE)
- VS Code (Python development)
- Git (version control)

**APIs and Sources:**

- Riot Games Data Dragon REST API
- League of Legends Wiki (Fandom)

**Database Specifications**

**Bronze Layer:**

- Column Types: NUMERIC, TEXT (permissive for error handling)
- Constraints: None (accept all data)
- Indexes: None (staging only)

**Silver Layer:**

- Column Types: INTEGER, VARCHAR, DATE, BOOLEAN, NUMERIC
- Constraints: CHECK constraints on rarity, region, segment
- Indexes: PRIMARY KEY on surrogate keys, UNIQUE on business keys
- Foreign Keys: Not enforced (to allow invalid references for DQ testing)

**dim_date Attributes:**

- date_key (INTEGER): Format YYYYMMDD (e.g., 20250315)
- Date components: year, quarter, month, week, day
- Business flags: is_weekend, is_month_start, is_month_end
- Coverage: 2010-01-01 to 2030-12-31 (7,671 days)

---

**KEY ACHIEVEMENTS**

**Data Quality:**

- Production-grade DQ framework with quarantine mechanism
- Automated monitoring with configurable alert thresholds
- Audit trail for all data quality checks
- Intentional error injection for framework validation

**Data Accuracy:**

- Real skin prices from LoL Wiki (not estimated)
- Actual release dates (not randomized)
- Correct Default skin pricing (0 RP)
- Esports skins preserved (purchasable content)

**Architecture:**

- Medallion pattern (Bronze/Silver/Gold)
- Proper staging layer for raw data
- Separation of concerns (raw/clean/curated)
- Scalable structure for future enhancements

**Data Model:**

- Star schema optimized for analytics
- Calendar dimension for time intelligence
- Player segmentation for cohort analysis
- Surrogate keys for SCD support

---

**IMPORTANT NOTES**

**Data Corrections**

**Default Skins Pricing:**

- Issue: Wiki Lua module shows cost = 880 (champion price)
- Fix: Explicitly set Default rarity skins to price_rp = 0
- Impact: 4 records corrected

**Release Dates:**

- Source: Extracted from Wiki Lua module "release" field
- Format: YYYY-MM-DD
- Coverage: Approximately 60% of skins have dates
- Missing dates: Left as NULL (not randomized)

**Design Decisions**

**dim_date Independence:**

- Not derived from transaction dates
- Pre-generated complete calendar 2010-2030

- Enables analysis of dates without transactions
- Supports future date planning

**Error Injection Strategy:**

- 10% error rate intentionally high for demonstration
- Real production would target <1%
- Mix of error types mirrors real-world issues
- Enables comprehensive DQ framework validation

**Esports Skins:**

- Kept in dataset despite filtering other special skins
- Rationale: Purchasable during World Championship events
- Examples: T1, DRX, EDG team skins

---

**CURRENT STATUS**

**Completed:**

- Bronze layer: Created and populated
- Silver layer: Created with DQ framework active
- Data quality monitoring: Operational with 5% threshold
- Quarantine system: Active and collecting invalid records
- Audit logging: Recording all DQ checks

**Next Steps:**

- Gold layer: Create final star schema
- Analytics views: Build business intelligence layer
- Performance optimization: Add indexes and partitioning
- Documentation: ER diagrams and data dictionary

---

**Project Date:** January 31, 2026
**Database:** lol_skins_dw (PostgreSQL 18)
**Current Layer:** Silver (Complete)
**Data Quality Status:** Monitoring Active (Alert: 9.27% error rate)