BrickLink LEGO Catalog Web Scraper

Scraping Project Documentation

Executive Summary

This project implements a comprehensive web scraping solution for BrickLink.com, extracting and organizing data about LEGO sets, minifigures, and parts into a normalized relational database structure. The system processes thousands of catalog pages, downloads product images, and creates complex relationship mappings between different item types.

Code Files - One-Line Descriptions

- **index.py**: Orchestrates the entire scraping workflow by sequentially executing item discovery, data collection, parsing, and relationship normalization stages.
- **item_scraper.py**: Extracts item numbers from BrickLink catalog listing pages using concurrent requests across all categories (sets, minifigures, parts).
- **data_scraper.py**: Downloads main product pages, price tables, inventory pages, and product images for each item, storing raw HTML in database.
- **parser.py**: Transforms raw HTML into structured data by extracting item details, pricing information, and relationship data using BeautifulSoup and regex.
- **db_connection.py**: Creates and populates a normalized connection table by converting comma-separated relationships into individual many-to-many database records.

1. Project Overview

1.1 Business Objectives

- Create a complete database of BrickLink catalog items
- Establish relationships between sets, minifigures, and parts
- Enable complex queries for pricing analysis and inventory management
- Support decision-making for collectors and resellers

1.2 Technical Goals

- Implement scalable web scraping architecture
- Normalize data into 3rd Normal Form (3NF)
- Optimize for performance with large datasets
- Ensure data integrity and relationship accuracy

1.3 Data Scope

- **Sets**: 409 catalog pages

- **Minifigures**: 352 catalog pages
- **Parts**: 1,757 catalog pages
- **Total Items**: Approximately 50,000+ unique items

---

2. System Architecture

2.1 Four-Stage Pipeline

Stage 1: Discovery

- Identifies all item numbers across three categories
- Navigates catalog listing pages systematically
- Extracts unique identifiers for each item

Stage 2: Collection

- Retrieves detailed information for each item
- Downloads multiple data points per item
- Stores raw HTML for future processing
- Downloads and archives product images

Stage 3: Extraction

- Parses raw HTML into structured data
- Extracts pricing, metadata, and relationships
- Handles different data formats per category
- Validates and cleans extracted information

Stage 4: Normalization

- Transforms comma-separated relationships into normalized records
- Creates many-to-many relationship mappings
- Implements referential integrity
- Optimizes for query performance

2.2 Database Design

Two-Database Approach:

1. **Raw Data Storage** (bricklink_data_raw.db)
    - Preserves original HTML responses
    - Enables re-parsing without re-scraping
    - Maintains data provenance

2. **Normalized Data Storage** (bricklink_parse.db)
    - Contains parsed, structured data
    - Implements relationship tables
    - Optimized for querying and analysis

---

3. Process Flow

3.1 Item Discovery Process

1. Access category-specific catalog pages
2. Parse pagination to determine total pages
3. Extract item numbers from catalog listings
4. Aggregate results by category type
5. Return consolidated item lists

3.2 Data Collection Process

1. For each discovered item:
    - Fetch main product page
    - Extract internal system ID
    - Retrieve pricing data via AJAX endpoints
    - Collect inventory/relationship information
    - Download product images
2. Handle different URL patterns:
    - Sets and Minifigures use inventory endpoints
    - Parts use appearance tracking endpoints
3. Store all responses with metadata

3.3 Data Extraction Process

1. Parse main product pages for:
    - Basic item information (name, number, category)
    - Physical attributes (dimensions, weight)
    - Metadata (year released, instructions availability)
    - Related items and cross-references
2. Extract pricing data:
    - Historical sales averages

    - Current market prices
    - Condition-based pricing (new/used)
    - Multiple time period analyses
3. Process relationship data:
    - Sets contain specific minifigures and parts
    - Minifigures contain specific parts
    - Parts appear in specific sets and minifigures

3.4 Relationship Normalization Process

1. Read comma-separated relationship strings
2. Split into individual item references
3. Create connection records for each relationship
4. Establish three relationship types:
    - Set-to-Part connections
    - Set-to-Minifigure connections
    - Minifigure-to-Part connections
5. Insert records in batches for efficiency

---

4. Data Handling Strategies

4.1 Relationship Management

Input Format:

- Relationships stored as comma-separated values
- Example: "3023, 3024, 3069b, 6636"

Normalization Process:

- Each comma-separated value becomes a separate record
- Maintains referential integrity
- Enables bidirectional queries

Query Capabilities:

- Find all parts in a specific set
- Identify sets containing specific minifigures
- Determine cheapest set for obtaining specific parts
- Calculate total part counts across sets

4.2 Data Quality Assurance

- Validation of extracted data
- Handling of missing or incomplete information
- Consistency checks across related items
- Duplicate detection and resolution

4.3 Special Handling Cases

- Parts don't have "consist of" relationships
- Some items may have empty relationship fields
- Price data may be unavailable for rare items
- International character encoding requirements

---

5. Performance Optimization

5.1 Concurrency Strategy

- Parallel processing for HTTP requests
- Multi-core utilization for data parsing
- Batch processing for database operations
- Resource pooling for connections

5.2 Database Optimization

- Strategic indexing on foreign keys
- Batch insertions to reduce overhead
- SQLite pragma optimizations
- Connection pooling

5.3 Memory Management

- Streaming processing for large datasets
- Batch-based data handling
- Garbage collection optimization
- Resource cleanup procedures

5.4 Network Efficiency

- Request throttling to respect server limits
- Proxy support for distributed requests
- Timeout management

- Retry mechanisms with backoff

---

6. Error Handling and Resilience

6.1 Network Resilience

- Automatic retry for failed requests
- Timeout handling for slow responses
- Proxy rotation for blocked requests
- Connection error recovery

6.2 Data Integrity

- Validation of parsed data
- Handling of malformed HTML
- Recovery from partial downloads
- Transaction rollback on errors

6.3 Progress Monitoring

- Real-time progress indicators
- Batch completion tracking
- Error logging and reporting
- Performance metrics collection

---

7. Output and Deliverables

7.1 Database Outputs

- Complete SQLite database with normalized data
- Connection table for relationship queries
- Indexed tables for performance
- Export capability to Excel format

7.2 File Outputs

- Downloaded product images
- Excel reports for each category
- Processing logs
- Error reports

7.3 Query Capabilities

- Complex relationship queries
- Price analysis across items
- Inventory optimization
- Collection completeness tracking

---

8. Technical Achievements

8.1 Scale Management

- Processes 50,000+ items efficiently
- Handles gigabytes of raw data
- Manages millions of relationships
- Maintains performance at scale

8.2 Data Quality

- Achieves 95%+ success rate
- Maintains referential integrity
- Provides clean, normalized data
- Enables complex analytics

8.3 Architecture Benefits

- Modular design for maintainability
- Separation of concerns
- Reusable components
- Extensible framework

This documentation provides a comprehensive overview of the BrickLink scraping project, focusing on the architectural decisions, process flows, and technical strategies that enable efficient extraction and organization of one of the world's largest LEGO marketplace datasets.