Extraction and Integration of MovieLens and IMDb Data

Verónika Peralta

Laboratoire PRiSM, Université de Versailles
45, avenue des Etats-Unis
78035 Versailles Cedex
FRANCE

veronika.peralta@prism.uvsq.fr

Technical Report [1]

July 2007

Abstract. This report describes the procedure followed for extracting and integrating MovieLens and IMDb data. We firstly focus on data extraction, describing source schemas, target schemas and the algorithms that perform data extraction, data cleaning and data transformation. We then describe data integration, describing the integrated schema, the algorithm that match movie titles and the construction of the integrated database. Finally, we present some statistics of the extracted and integrated data.

1. Introduction

This report describes the procedure followed for extracting and integrating IMDb and MovieLens data.

IMDb, the Internet Movie Database, is a huge collection of movie information (auto-claimed to be the earth's biggest movie database). It started as a hobby project by an international group of movie fans and currently belongs to the Amazon.com Company. IMDb tries to catalog every pertinent detail about a movie, from who was in it, to who made it, to trivia about it, to filming locations, and even where you can find reviews and fan sites on the web. The IMDb web site (http://www.imdb.com/ [4]) provides 49 text files in ad-hoc format (called lists) containing different characteristics about movies (e.g. actors.list or running-times.list). Lists content is continually updated and enlarged; at October 5th 2006, the movie list included more than 858.000 movies.

MovieLens is a movie recommender project, developed by the Department of Computer Science and Engineering at the University of Minnesota. MovieLens is a typical collaborative filtering system that collects movie preferences from users and then groups users with similar tastes. Based on the movie ratings expressed by all the users in a group it attempts to predict for each individual their opinion on movies they have not yet seen. Two data sets are available at the MovieLens web site (http://movielens.umn.edu/ [2]). The first one consists of 100,000 ratings for 1682 movies by 943 users. The second one consists of approximately 1 million ratings for 3883 movies by 6040 users.

Our goal is to build a relational database about movies, which includes movie descriptions and user ratings. To this end, we extract, transform and integrate data provided by IMDb and MovieLens sites. This database will be used, in future works, to evaluate the performance and pertinence of different techniques and algorithms for query large data sets taking into account user preferences and data quality.

The extraction and integration procedure has 4 main steps: (i) extraction of MovieLens data, (ii) extraction and transformation of IMDb data, (iii) matching of MovieLens and IMDb movie titles, and (iv) construction of the integrated database. Figure 1 shows an overview of these steps.

---

[1] This research was partially supported by the French Ministry of Research and New Technologies under the ACI program devoted to Data Masses (ACI-MD), project #MD-33.

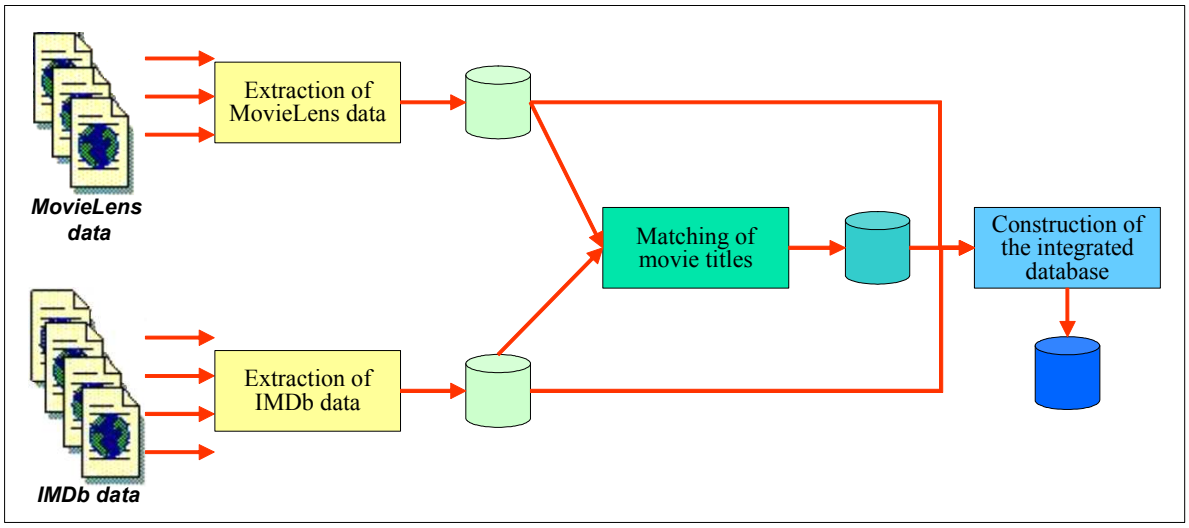

Figure 1 – Extraction and integration steps

Data extraction and integration processes were executed in a Microsoft Access database and then migrated to an Oracle database. In the remaining of the report we describe each step: Sections 2 and 3 present the extraction of MovieLens and IMDb data respectively, describing source schemas, target schemas and the algorithms that perform data extraction, data cleaning and data transformation. Section 4 presents data integration, describing the integrated schema, the algorithm that matches movie titles and the construction of the integrated database. Matching details are subject of a more detailed report [Peralta 2007a]. Finally, Section 7 concludes.

2. MovieLens data extraction

MovieLens data sets are provided in the form of tabular text files (comma separated text). Data extraction consisted in the loading of text data to a relational database, normalization of data and duplicates elimination. The following sub-sections describe source files, target schemas, extraction processes and cleaning processes.

2.1. MovieLens source schemas

MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota. They provide two data sets (with 100.000 and 1.000.209 evaluations respectively). Data was collected through the MovieLens web site [3] and was cleaned up, i.e. users who had less than 20 ratings or did not have complete demographic information were removed from data sets.

We concentrated in the largest one, however, the smallest one is helpful in for the matching of movie titles (because it provides the IMDb URL of each movie) so it was extracted too. The following sub-sections describe both schemas.

2.1.1. Source files of the large data set

The large data set consists in 3 text files, with tabular format, describing 1000209 anonymous ratings of 3883 movies made by 6040 MovieLens users who joined MovieLens in 2000. In the following, we describe the contents of each text file.

Ratings.dat

This file contains data about 1000209 ratings (1 evaluation, corresponding to 1 user and 1 movie, in each line) in the format: *UserID::MovieID::Rating::Timestamp* where:

- *UserID* is an integer, ranging from 1 to 6040, that identifies a user. Each user has rated at least 20 movies.
- *MovieID* is an integer, ranging from 1 to 3952, that identifies a movie.
- Rating is an integer, ranging from 1 to 5, made on a 5-star scale (whole-star ratings only).
- Timestamp is represented in seconds since 1/1/1970 UTC

Figure 2 shows an extract of the file corresponding to 5 ratings.

```
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
1::3408::4::978300275
1::2355::5::978824291
```

Figure 2 – Extract of the Ratings.dat file

Movies.dat

This file contains data about 3883 movies (1 movie in each line) in the format: *MovieID::Title::Genres* where:

- *MovieID* is an integer, ranging from 1 to 3952, that identifies a movie.
- *Title* is a String that concatenates movie title and year of release (between brackets).
- *Genres* is a pipe-separated list of genres. Provided genres are: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western.

Figure 3 shows an extract of the file corresponding to 5 movies.

```
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama
5::Father of the Bride Part II (1995)::Comedy
```

Figure 3 – Extract of the Movies.dat file

Users.dat

This file contains data about 6040 users (1 user in each line) in the format: *UserID::Gender::Age::Occupation::Zip-code* where:

- *UserID* is an integer, ranging from 1 to 6040, that identifies a user.
- *Gender* is denoted by a "M" for male and "F" for female
- *Age* is an integer identifying a range (the minimum age in the range). Provided ranges are: under 18, 18-24, 25-34, 35-44, 45-49, 50-55, over 56.
- *Occupation* is an integer, ranging from 0 to 20, indicating the following choices: 0: other or not specified, 1: academic/educator, 2: artist, 3: clerical/admin, 4: college/grad student, 5: customer service, 6: doctor/health care, 7: executive/managerial, 8: farmer, 9: homemaker, 10: K-12 student, 11: lawyer, 12: programmer, 13: retired, 14: sales/marketing, 15: scientist, 16: self-employed, 17: technician/engineer, 18: tradesman/craftsman, 19: unemployed, 20: writer
- *Zip-code* is a five-digits integer indicating user ZIP-code.

All demographic information was provided voluntarily by the users and was not checked for accuracy. Only users who have provided some demographic information are included in this data set. Figure 4 shows an extract of the file corresponding to 5 users.

```
1::F::1::10::48067
2::M::56::16::70072
3::M::25::15::55117
4::M::45::7::02460
5::M::25::20::55455
```

Figure 4 – Extract of the Users.dat file

2.1.2. Source files of the small data set

The small data set consists in 5 text files, with tabular format, describing 100000 anonymous ratings of 1682 movies made by 943 users during the seven-month period from September 19th, 1997 through April 22nd, 1998. In the following, we describe the contents of each text file.

u.data

This file contains data about 100000 ratings (1 evaluation, corresponding to 1 user and 1 movie, in each line). It is a tab-separated list of *UserID*, *MovieID*, *Rating* and *Timestamp*, where:

- *UserID* is an integer, ranging from 1 to 943, that identifies a user. Each user has rated at least 20 movies.
- *MovieID* is an integer, ranging from 1 to 1682, that identifies a movie.
- Rating is an integer, ranging from 1 to 5, made on a 5-star scale (whole-star ratings only).
- Timestamp is represented in seconds since 1/1/1970 UTC

Data is randomly ordered. Figure 5 shows an extract of the file corresponding to 5 ratings.

```
196    242    3      881250949
186    302    3      891717742
22     377    1      878887116
244    51     2      880606923
166    346    1      886397596
```

Figure 5 – Extract of the u.data file

u.item

This file contains data about 1682 movies (1 movie in each line). This is a pipe-separated list of *MovieID*, *MovieTitle*, *ReleaseDate*, *VideoReleaseDate*, *IMDbURL*, *unknown*, *Action*, *Adventure*, *Animation*, *Children's*, *Comedy*, *Crime*, *Documentary*, *Drama*, *Fantasy*, *Film-Noir*, *Horror*, *Musical*, *Mystery*, *Romance*, *Sci-Fi*, *Thriller*, *War* and *Western*, where

- *MovieID* is an integer, ranging from 1 to 1682, that identifies a movie.
- *MovieTitle* is a String that concatenates movie title and year of release (between brackets).
- *ReleaseDate* is a DD-Mon-YYYY date indicating movie release date
- *VideoReleaseDate* was destinated to video release date but it is always NULL
- *IMDbURL* indicates the URL of the movie in the IMDb site.
- The last 19 fields correspond to genres; a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once.

Figure 6 shows an extract of the file corresponding to 5 movies.

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)|0|0|0|1|1|1|0|0|
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?GoldenEye%20(1995)|0|1|1|0|0|0|0|0|0|
0|0|0|0|0|0|0|1|0|0
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)|0|0|0|0|0|0|0|
0|0|0|0|0|0|0|0|0|1|0|0
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)|0|1|0|0|0|1|0|
0|1|0|0|0|0|0|0|0|0|0|0
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Copycat%20(1995)|0|0|0|0|0|0|1|0|1|0|0|
0|0|0|0|0|1|0|0
```

Figure 6 – Extract of the u.item file

u.genre

This file contains data about 19 movie genres (1 genre in each line). This is a pipe-separated list of *GenreID* and *GenreName*, where

- *GenreID* is an integer, ranging from 0 to 18, that identifies a genre.
- *GenreName* is a String describing the genre. Genre names corresponds to genre columns of the u.item file.

Figure 7 shows an extract of the file corresponding to 5 genres.

```
unknown|0
Action|1
Adventure|2
Animation|3
Children's|4
```

Figure 7 – Extract of the u.genre file

u.user

This file contains data about 943 users (1 user in each line). This is a pipe-separated list of *UserID*, *Age*, *Gender*, *Occupation* and *Zip-code*, where:

- *UserID* is an integer, ranging from 1 to 943, that identifies a user
- *Age* is an integer indicating user's age
- *Gender* is denoted by a "M" for male and "F" for female
- *Occupation* is an String indicating user occupation.
- *Zip-code* is a five-digits integer indicating user ZIP-code.

All demographic information was provided voluntarily by the users and was not checked for accuracy. Only users who have provided some demographic information are included in this data set. Figure 8 shows an extract of the file corresponding to 5 users.

```
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
```

Figure 8 – Extract of the u.user file

u.occupation

This file lists 21 user occupations (1 occupation in each line). Figure 9 shows an extract of the file corresponding to 5 occupations.

```
administrator
artist
doctor
educator
engineer
```

Figure 9 – Extract of the u.occupations file

2.2. MovieLens target schemas

Both MovieLens data set were extracted to a Microsoft Access® database. The following sub-sections describe the target schemas for both data sets.

2.2.1. Target tables of the large data set

The target schema for the large data set consists in 6 tables describing movies, users and ratings. Figure 10 shows the tables of the target schema, their primary keys (attributes in bold) and foreign keys (lines between tables). Table 1 describes each target table, its attributes and constraints.

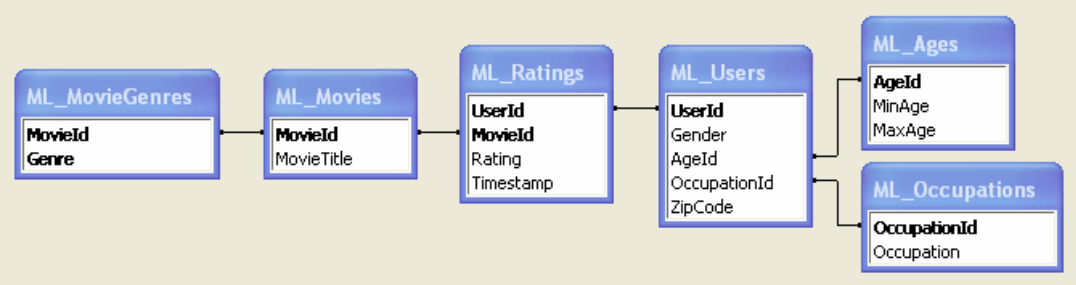

Figure 10 – MovieLens schema (large data set)

Table	Attributes	Constraints
ML_Users *Demographic information about users*	– UserId: Numeric(4) – Gender: Char(1); value in {M,F} – AgeId: Numeric(2) – OccupationId: Numeric(2) – ZipCode: String(10)	Primary key: UserId Foreign keys: – Age: ref. ML_Ages – Occupation: ref. ML_Occupation
ML_Movies *Titles of movies*	– MovieId: Numeric(4) – MovieTitle: String(100)	Primary key: MovieId Unique: MovieTitle
ML_Ratings *User ratings on movies*	– UserId: Numeric(4) – MovieId: Numeric(4) – Rating: Number(1), value in {1,2,3,4,5} – Timestamp: Numeric(9); represents milliseconds from an initial time	Primary key: UserId, MovieId Foreign keys: – UserId: ref. ML_Users – MovieId: ref. ML_Movies
ML_MovieGenres *Genres of movies*	– MovieId: Numeric(4) – Genre: String(12)	Primary key: MovieId, Genre Foreign keys: – MovieId: ref. SML_Movies
ML_Ages *Age intervals*	– AgeId: Numeric(2) – MinAge: Numeric(2) – MaxAge: Numeric(2)	Primary key: AgeId
ML_Occupations *Occupations*	– OccupationId: Numeric(2) – Occupation: String (25)	Primary key: OccupationId

Table 1 – MovieLens schema (large data set)

2.2.2. Target tables of the small data set

The target schema for the small data set consists in 5 tables describing movies, movie genres, users and ratings. Figure 11 shows the tables of the target schema, their primary keys (attributes in bold) and foreign keys (lines between tables). Table 2 describes each target table, its attributes and constraints.

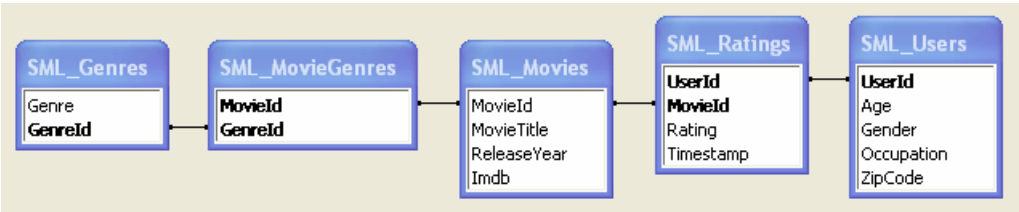

Figure 11 – MovieLens schema (small data set)

Table	Attributes	Constraints
SML_Users *Demographic information about users*	– UserId: Numeric(4) – Age: Numeric(2) – Gender: Char(1); value in {M,F} – Occupation: String(15); finite set – ZipCode: String(5)	Primary key: UserId
SML_Movies *Information about movies*	– MovieId: Numeric(4) – MovieTitle: String(100) – ReleaseYear: Numeric(4) – Imdb: String(150); represents the URL of an IMDb page	Primary key: MovieId Unique: MovieTitle
SML_Ratings *User ratings on movies*	– UserId: Numeric(4) – MovieId: Numeric(4) – Rating: Number(1), value in {1,2,3,4,5} – Timestamp: Numeric(9); represents milliseconds from an initial time	Primary key: UserId, MovieId Foreign keys: – UserId: ref. SML_Users – MovieId: ref. SML_Movies
SML_Genres *Description of genres*	– Genre: String(12) – GenreId: Numeric(2)	Primary key: GenreId
SML_MovieGenres *Genres of movies*	– MovieId: Numeric(4) – GenreId: Numeric(2)	Primary key: MovieId, GenreId Foreign keys: – MovieId: ref. SML_Movies – GenreId: ref. SML_Genres

Table 2 – MovieLens schema (small data set)

2.3. MovieLens extraction, cleaning and transformation processes

Both data sets were quite consistent; however, we detected and solved some kinds of anomalies:

- *Multi-valued attributes*: Some source attributes are provided as comma-separated (or pipe-separated) lists of values. As a solution, we normalise tables.
- *Null or dummy values in mandatory attributes*: Some mandatory attributes (especially key attributes) contains Null or dummy values (e.g. MovieTitle = 'Unknown'). As a solution, such tuples are eliminated.
- *Duplicates*: Some tuples (or tuple keys) are duplicated. As a solution, we keep a unique tuple, reconciling values of further attributes if necessary.
- *Orphans*: Some tuples do not satisfy referential constraints, i.e. their values are not included in a foreign table. As a solution, we eliminate orphan tuples.

In order to solve such anomalies we implemented an ETL process for extracting, cleaning and transforming data of each source file. ETL workflows consist of five major tasks as illustrated in Figure 12 (rectangles represent

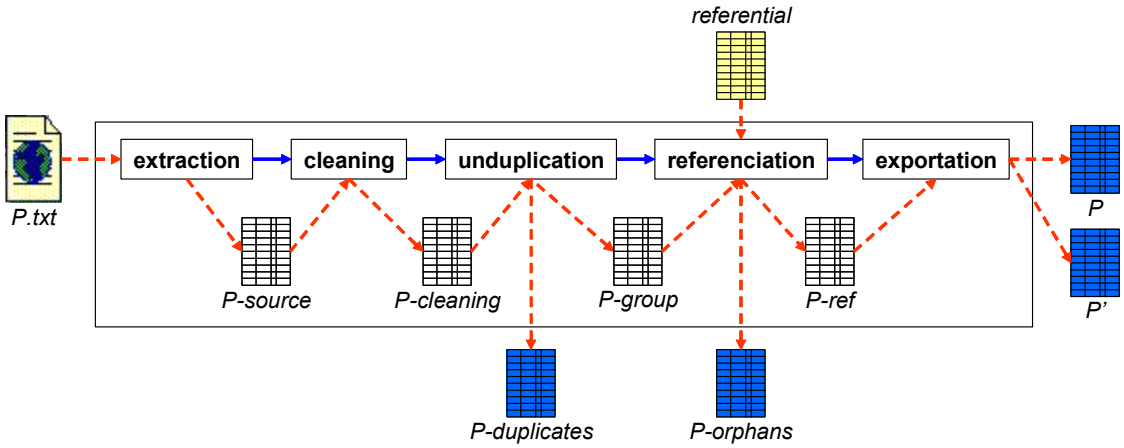

Figure 12 – High-level ETL process

tasks; continuous arrows represent control flow; dotted arrows represent data flow, indicating the tables that are input or output of tasks. ETL inputs are a text file containing the data to extract (P.txt in Figure 12) and (possibly) some referential tables (referential in Figure 12). ETL output is a relational table containing the extracted data (P in Figure 12) and (possibly) some aggregation tables (P' in Figure 12). Two additional tables keep trace of the anomalies found (P-duplicates and P-orphans in Figure 12). We stored the result of each task in a temporal table (white tables in Figure 12) in order to ease traceability and maintainability. For certain specific files, ETL workflows can be simpler car not all the tasks are necessary for loading and cleaning all tables. In the following, we describe the semantics of each task (details are illustrated in Figure 13).

Bulk copy tasks load data from text files to relational tables. We used Microsoft Access® loading interface, indicating file format and column format.

Cleaning tasks perform different data cleaning transformations, for example, elimination of Null values, denormalisation, calculation of derived attributes and so on. Specific cleaning tasks are described later for each table.

Unduplication tasks eliminate duplicate data. Several sub-tasks are performed, distinguishing normal flow (when there are no duplicates) and error flow (when there are duplicates):

- *Duplicate check* tasks verify the existence of duplicate values of key (or unique) attributes. Note that key (or unique) constraints are not defined yet; unduplication is needed in order to define the appropriate constraints. Duplicate tuples are inserted into temporal tables (P-duplicates in Figure 13), registering the key (or unique) value, the number of times the key (or unique) value is repeated, and the conflicting values of other attributes (minimum and maximum values). Tasks are implemented as SQL operations of the form:

```
INSERT INTO P-duplicates (key_attribute1,... key_attributeM, quantity,
                min_column1, max_column1,... min_columnN, max_columnN)
SELECT key_attribute1,... key_attributeM, COUNT(*), MIN(column1),
       MAX(column1),... MIN(columnN), MAX(columnN)
FROM P-cleaning
GROUP BY key_attribute1,... key_attributeM
HAVING COUNT(*)>1;
```

- *Reconciliation* tasks are executed only when some duplicates where found (in the corresponding duplicate check task). Inconsistencies are generally solved by arbitrarily selecting one of the values of conflictive attributes (e.g. the minimum one) or filling with Null values. Additional columns are added to the

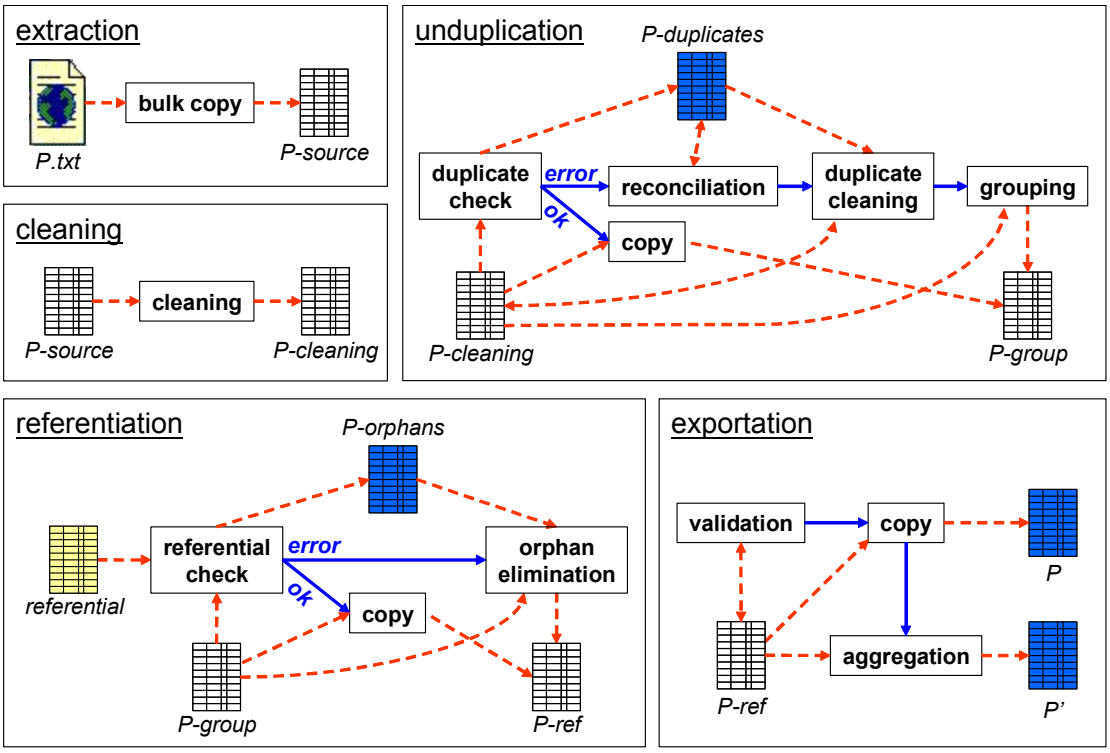

Figure 13 – Details of ETL tasks

temporal table storing duplicates (P-duplicates in Figure 13) in order to store reconciled values. Tasks are implemented as a sequence of SQL operations, each one solving one inconsistency. The following operations are examples of reconciliation operations:

```
UPDATE P-duplicates
SET column1 = min_column1;

UPDATE P-duplicates
SET column2 = NULL
WHERE min_column2 <> max_column2;
```

– *Duplicate cleaning* tasks set a unique value for each non-key attribute (those indicated during reconciliation task) in duplicated tuples. Tasks are implemented as SQL operations of the form:

```
UPDATE P-cleaning
SET column1 = P-duplicates.column1,...
    columnN = P-duplicates.columnN
WHERE P-cleaning.key_attribute1 = P-duplicates.key_attribute1 ...
AND P-cleaning.key_attributeM = P-duplicates.key_attributeM
```

– *Grouping* tasks keep a unique tuple for each key value. Note that there are conflicting values anymore, so grouping by all attributes is analogous to grouping by key attributes. Tasks are implemented as SQL operations of the form:

```
INSERT INTO grouped_table (key_attribute1,... key_attributeM, column1,...
                columnN)
SELECT key_attribute1,... key_attributeM, column1,... columnN
FROM cleaning_table
GROUP BY key_attribute1,... key_attributeM, column1,... column;
```

– *Copy* tasks duplicate tables in order to provide traceability in the case no duplicates were found. Tasks are implemented as SQL operations of the form:

```
INSERT INTO P-group (
SELECT *
FROM P-cleaning;
```

Referentiation tasks eliminate data not satisfying referential constraints. Several sub-tasks are performed, distinguishing normal and error flow:

– *Referential check* tasks verify the satisfaction of referential constraints. Orphan tuples (which do not match tuples of the referenced table) are stored in a temporal table (P-orphans in Figure 13). Tasks are implemented as SQL operations of the form:

```
INSERT INTO P-orphans (
SELECT *
FROM P-group
WHERE NOT EXISTS (
   SELECCT * FROM referential
   WHERE reference_attribute1 = foreign_attribute1 AND...
         reference_attributeN = foreign_attributeN));
```

– *Orphan elimination* tasks deletes tuples not satisfying referential constraints (those found by the corresponding referential check task). In other words, tasks keep only tuples that join referential tables. Tasks are implemented as SQL operations of the form:

```
INSERT INTO P-ref (
SELECT P-group.*
FROM P-group, referential
WHERE reference_attribute1 = foreign_attribute1 ...
AND   reference_attributeN = foreign_attributeN));
```

– *Copy* tasks are analogous to those described for unduplication.

Exportation tasks create result tables. Several sub-tasks are performed:

– *Validate* tasks perform manual user validations (e.g. control of some sampled tuples) or modifications (e.g. the insertion of new tuples). Specific validation tasks are described later for each table.

– *Copy* tasks are analogous to those described for unduplication.

– *Aggregate* tasks build aggregations (P' in Figure 13) for storing existing values of a specific attribute, i.e. master tables (e.g. movie genres). Tasks are implemented as SQL operations of the form:

```
INSERT INTO P' (
SELECT column1,... columnK
FROM P-ref
GROUP BY column1,... columnK;
```

The following sub-sections describe the specific tasks of each data set.

2.3.1. *ETL process for the large data set*

The large data set was quite consistent: no duplicates were found, no violations to referential constraints were detected. However, we detected and solved some minor anomalies:

– Genres were represented as multi-valued attributes (pipe-separated lists) in source files. Normalization routines were performed.

– ZIP codes contained letters and ZIP intervals. We ignored such errors and stored values as Strings.

– Master tables for ages and occupations were created from scratch in order to associate descriptions to age ranges and occupation ids.

Figure 14 shows the extraction workflow. Note that not all the tasks presented in previous section were necessary for extracting data from all text files. For ease of understanding, we omitted representing some data flows and temporary tables; copy tasks were also omitted.

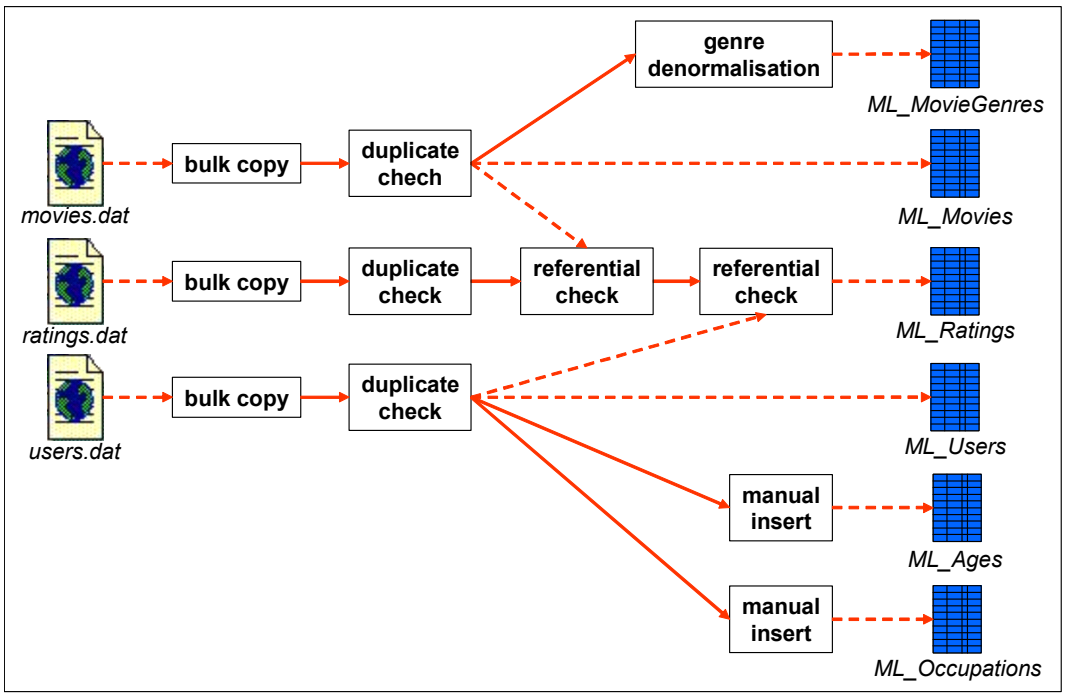

Figure 14 – ETL workflow for the large data set

The semantics of the *bulk copy*, *duplicate check* and *reference check* tasks was described previously; the semantics of specific validation and aggregation tasks is the following:

– *Genre denormalisation* task builds a normalized table expressing the relationship among movies and genres. Each tuple, which has a list of genres, is decomposed in several tuples, one per genre. For example, from the tuple <1,'Toy Story (1995)','Animation|Children's|Comedy'> (see Figure 3) we create 3 tuples <1,'Animation'>, <1,'Children's'> and <1,'Comedy'>. The task is implemented as a sequence of SQL operations that split the genre list according to the token '|'.

– *Manual insert* tasks create new tables by executing a list of manually-entered insert operations, for example:

```
INSERT INTO ML_Ages (AgeId, MinAge, MaxAge)
VALUES (18, 18, 24);
```

2.3.2. *ETL process for the small data set*

We detected and solved several anomalies in the small data set:

– We found 18 duplicate titles (with different MovieId). These tuples would cause duplicates when matching (by title) IMDb movies. We kept, arbitrarily, the smaller MovieId of each duplicate title.

– There was a movie with title='Unknown' and null values for ReleaseYear and IMDbURL. The movie was eliminated.

– Ratings of eliminated movies were also eliminated. Substituting their MovieId for those of the kept movies would carry to duplicates with contradictory ratings.

– Genres were denormalized in source files (one Boolean attribute for each genre). Normalization routines were performed.

Figure 15 shows the extraction workflow.

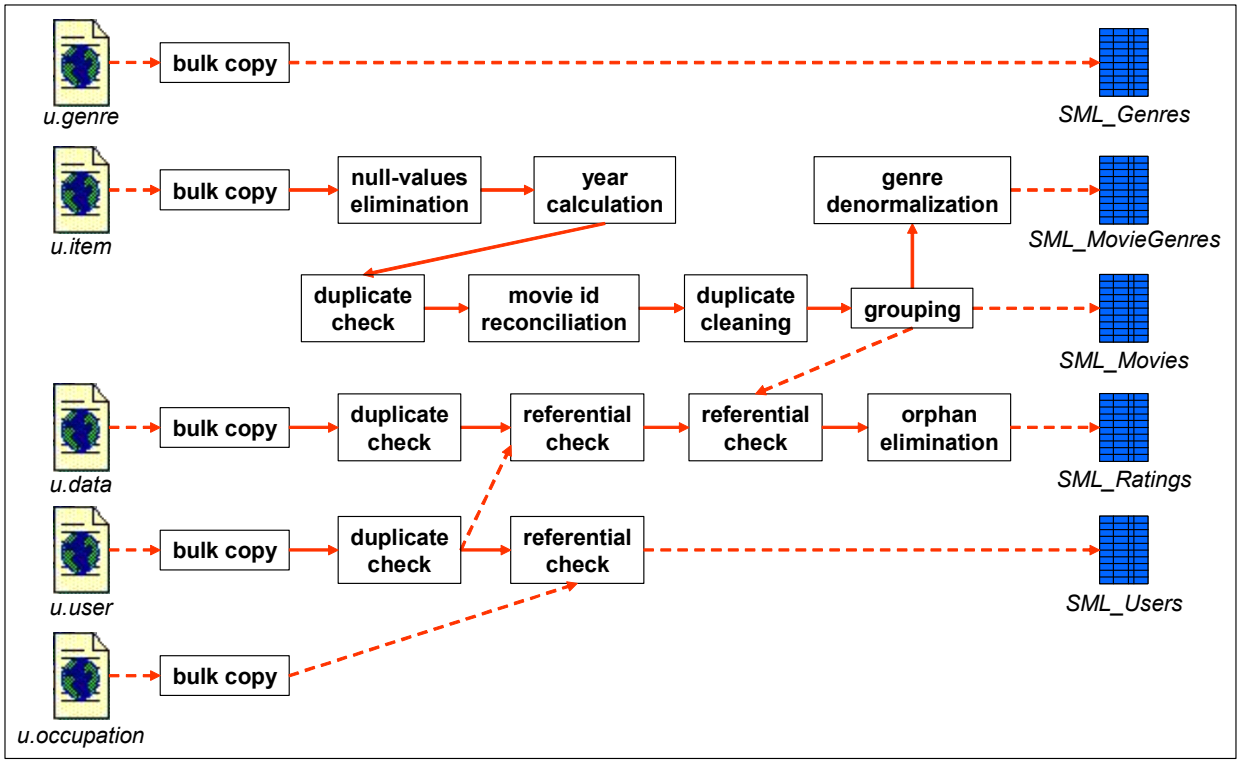

Figure 15 – ETL workflow for the small data set

The semantics of the *bulk copy*, *duplicate check*, *duplicate cleaning*, *grouping*, *reference check* and *orphan elimination* tasks was described previously; the semantics of specific cleaning, reconciliation and aggregation tasks is the following:

– *Null-values elimination* task deletes tuples having null or dummy values. Specifically, we deleted tuples having 'Unknown' as title. Task is implemented as follows:

```
INSERT INTO movies-cleaning (MovieId, MovieTitle, ReleaseDate, IMDb)
SELECT * FROM movies-source
WHERE MovieTitle <> 'Unknown';
```

– *Year calculation* task calculates movie release year from movie release date. A typical truncate function is used. The task is implemented as follows:

```
UPDATE movies-cleaning
SET ReleaseYear = Mid(ReleaseDate,8,4);
```

– *MovieId reconciliation* task keeps the smaller movie id when there are duplicated movid titles (supposed to be unique). The task is implemented as follows:

```
UPDATE movie-duplicates
SET MovieId = MinMovieId;
```

– *Genre denormalisation* task builds a normalized table expressing the relationship among movies and genres. Each tuple, which has several Boolean values each one corresponding to a genre, is decomposed in several tuples, one for each True value. For example, from the tuple <1,'Toy Story (1995)',…,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0> (see Figure 6) we create 3 tuples <1,3>, <1,4> and <1,5> indicating that genres 3, 4 and 5 correspond to the movie. The task is implemented as a sequence of SQL operations, one for each genre; for example, the query for Adventure movies (genre 2) is:

```
INSERT INTO SML_MovieGenres (MovieId, GenreId)
SELECT MovieId, 2
FROM Movies-ref
WHERE Adventure=1;
```

2.3.3. Summary and statistics

As summary of ETL tasks, the following tables show the number of extracted tuples, the number of anomalies, duplicates and orphans found, and the number of exported tuples for both data sets.

Source file	Extracted tuples	Cleaning anomalies	Duplicates	Orphans	Exported tuples	Resulting table
Movies.dat	3883	0	0	0	3883	ML_Movies
					6407	ML_MovieGenres
Ratings.dat	1000209	0	0	0	1000209	ML_Ratings
Users.dat	6040	0	0	0	6040	ML_Users
					7	ML_Ages
					21	ML_Occupations

Table 3 – Statistics of data extraction, cleaning and transformation of the large data set

Source file	Extracted tuples	Cleaning anomalies	Duplicates	Orphans	Exported tuples	Resulting table
u.genre	19	0	0	0	19	SML_Genres
u.item	1682	1	18	0	1663	SML_Movies
					2906	SML_MovieGenres
u.data	100000	0	0	617	99383	SML_Ratings
u.user	943	0	0	0	943	SML_Users
u.occupation	21					

Table 4 – Statistics of data extraction, cleaning and transformation of the small data set

3. IMDb data extraction

IMDb data set consists in ad-hoc text files, having different formats, including tabular lists, tagged text and hierarchical-organized text. Data extraction consisted in the loading of text data to a relational database, normalization of data and duplicates elimination. The following sub-sections describe source files, target schemas, extraction processes and cleaning processes.

3.1. IMDb source schemas

IMDb data set [5] consists of 49 lists that catalog different details about movies. At October 5th 2006, list sizes varied from 25000 to 5000000 tuples. Each list has a list manager, who is responsible for updating, maintaining and publishing it (http://www.imdb.com/helpdesk/contact). List managers rely on users of IMDb to submit

corrections and additions to keep lists accurate and complete as possible. List updates are mostly submitted via the movie mail-server's central collection service, following specific submission protocols (http://www.imdb.com/mailing_lists).

We extracted data from 23 lists, representing the most relevant tabular features about movies. We discarded lists providing textual data (10 lists), such as movie plots or trivia, because it was useless for our database-oriented goals. The remaining 16 lists will be possibly extracted in near future.

We treated four file formats:

Fixed-length columned files present data in several columns, each one starting at a fixed position. They are very simple to treat with a database loading interface like the one used for extracting MovieLens data sets. Figure 16 shows an extract of the ratings.list file. It is the unique file having this format:

- *ratings.list* has 4 columns (rating distribution, number of votes, average rating, movie title).

```
0000000015  178183   9.1  Godfather, The (1972)
0000000115  214539   9.1  Shawshank Redemption, The (1994)
0000000124  101253   9.0  Godfather: Part II, The (1974)
0000000015  161827   8.9  Lord of the Rings: The Return of the King, The (2003)
0000000124   87270   8.8  Casablanca (1942)
0000000124  129696   8.8  Schindler's List (1993)
```

Figure 16 – Extract of the ratings.list file

Tab-separated columned files also present data in several columns, but columns are separated by tabulations. The number of tabulations between two consecutive columns is variable (responding to visual presentation) which unables the direct use of database loading interfaces. Files need to be pre-processed in order to substitute several tabulations by a unique one. In addition, as some values can be omitted (null values), in some cases, a double tabulation is expected. In particular, most lists contain optional comments. As anomalies, we found occurrences of double spaces instead of a tabulation. Figure 17 shows an extract of the genres.list file, which presents movie titles and their genres separated by several tabulations. The files having this format are:

- *color-info.list* has 3 columns (movie title, color, comments).
- *countries.list* has 2 columns (movie title, country).
- *genres.list* has 2 columns (movie title, genre).
- *keywords.list* has 2 parts. The former contains a list of keywords (with the number of films including such keywords between brackets), organized in several tab-separated columns. The latter contains two columns (movie title, keyword).
- *language.list* has 3 columns (movie title, language, comments).
- *locations.list* has 2 columns (movie title, location, comments), where *location* is a comma-separated list of geographical names, for example: “Sevilla, Andalucía, Spain”.
- *movies.list* has 3 columns (movie title, release year, comments).
- *production-companies.list* has 3 columns (movie title, production company, comments), where *production company* may contain the code of one or several countries (between square brackets), for example: “Taxi Films [uy]”
- *release-dates.list* has 3 columns (movie title, release date, comments), where *release date* embeds release country.
- *running-times.list* has 2 columns (movie title, running time, comments), where *running time* may embed the running country.
- *sound-mix.list* has 2 columns (movie title, sound mix).

```
Walking with the Dead (1998)                    Short
Walking with Walken (2001)                      Short
Walkin' on Sunshine: The Movie (1997)                 Comedy
Walkin' on Sunshine: The Movie (1997)                 Sci-Fi
Walk in the Clouds, A (1995)                    Drama
Walk in the Clouds, A (1995)                    Romance
```

Figure 17 – Extract of the genres.list file

Tagged files precedes data with tags, which indicates the nature of data. Figure 18 shows an extract of the biographies.list file; tags are placed at the start of each line, indicating for example, an artist name (NM), his real name (RN), his date and place of birth (DB), etc. Tags are prefixed for each list but some tags are not explained. This type of file needs ad-hoc parsing. The files having this format are:

- *biographies.list* includes as tags: artist name, real name, birth (date and place), decease (date, place and cause) and height. Other tags are ignored.
- *business.list* includes as tags: movie title, budget (currency and amount), and revenue (currency, amount and country). Other tags are ignored.

```
NM: Noth, Chris

RN: Noth, Christopher David

DB: 13 November 1954, Madison, Wisconsin, USA

BG: Chris Noth was born in Madison, Wisconsin on 13 November 1954.
In his
BG: youth he travelled and lived in England, Yugoslavia and Spain…
```

Figure 18 – Extract of the biographies.list file

Hierarchical-structured files organize data into two hierarchical levels: the outside level describes a movie feature (e.g. an actor or a director) and the inside level lists all movies corresponding to such feature. Figure 19 shows an extract of the actors.list file, presenting the list of movies played by an actor. Some files contains other attributes and comments, which follow movie titles (in the same line) separated by one or more tabulations. This type of file is the most difficult to extract; it needs ad-hoc extractors to iterate between hierarchical levels, and pre-processing for solving tabulations problems (see tab-separated columned files). The files having this format are:

- *actors.list* lists actors and 4 columns (movie title, role, casting position, comments). Columns (excepting movie title) are optional.
- *actresses.list* lists actresses and 4 columns (movie title, role, casting position, comments). Columns (excepting movie title) are optional.
- *costume-designers.list* lists costume designers and 2 columns (movie title, comments).
- *directors.list* lists directors and 2 columns (movie title, comments).
- *distributors.list* lists movie titles and 2 columns (distribution company, comments), where *distribution company* may contain acronyms (between brackets) and the code of one or several countries (between square brackets), for example: “France 2 (FR 2) [fr]”
- *movie-links.list* lists movie titles and 1 columns (link to another movie title), which embeds the link type (e.g. “featured in”, “alternate language version of”, “referenced in”, etc).
- *producers.list* lists producers and 2 columns (movie title, comments).
- *production-designers.list* lists producion designers and 2 columns (movie title, comments).
- *writers.list* lists custome designers and 2 columns (movie title, comments).

```
Noth, Chris   Abducted: A Father's Love (1996) (TV)  [Larry Coster]  <1>
              Acting Class, The (2000)  [Martin Ballsac]  <8>
              Apology (1986) (TV)  [Roy Burnette]  <12>
              At Mother's Request (1987) (TV)  [Steve Klein]
              Baby Boom (1987)  (as Christopher Noth)  [Yuppie Husband]  <44>
              Bad Apple (2004) (TV)  [Tozzi]  <1>
```

Figure 19 – Extract of the actors.list file

Table 5 summarizes file formats, optional attributes and other particularities.

File	Format	Optional att.	Particularities
movies.list	Tab-separated columned		
actors.list	Hierarchical-structured	comments, role, casting	
actresses.list	Hierarchical-structured	comments, role, casting	
biographies.list	Tagged		Columns *birth* and *decease* are comma-separated lists of dates and places.
business.list	Tagged		Columns *budget* and *revenue* embed currencies, amounts and countries.
color-info.list	Tab-separated columned	comments	
costume-designers.list	Hierarchical-structured	comments	
countries.list	Tab-separated columned		
directors.list	Hierarchical-structured	comments	
distributors.list	Hierarchical-structured	comments	Column *distribution company* embeds company acronyms and country code
genres.list	Tab-separated columned		
keywords.list	Tab-separated columned		Contains two lists: (i) list of keywords, (ii) lists of movies and their keywords.
language.list	Tab-separated columned	comments	
locations.list	Tab-separated columned	comments	Column *location* is a comma-separated list of geographical places.
movie-links.list	Hierarchical-structured		Column *link* embeds link type.
producers.list	Hierarchical-structured	comments	
production-companies.list	Tab-separated columned	comments	Column *production company* embeds country code
production-designers.list	Hierarchical-structured	comments	
ratings.list	Fixed-length columned		
release-dates.list	Tab-separated columned	comments	Column *release date* embeds country.
running-times.list	Tab-separated columned	comments	Column *running time* embeds country.
sound-mix.list	Tab-separated columned		
writers.list	Hierarchical-structured	comments	

Table 5 – List formats and particularities

Movies correspond to one of 6 types: cinema film, series episode, mini-series, TV series, video and video game. Movie identifiers (MovieTitle columns) are Strings that concatenates different data depending on movie types:

- *Cinema film* identifiers concatenate movie title and release year (between brackets). E.g.: Walk in the Clouds, A (1995)
- *Series episode* identifiers concatenate movie title (quoted), release year (between brackets) and episode identification (between braces). The latter may include episode name, episode number (between brackets), or both. E.g.: "Sex and the City" (1998) {All or Nothing (#3.10)}
- *Mini-series* identifiers concatenate movie title (quoted), release year (between brackets) and the word *mini* (between brackets). E.g.: "Bon Voyage" (2006) (mini)
- *TV series* identifiers concatenate movie title, release year (between brackets) and the word *TV* (between brackets). E.g.: Aladdin on Ice (1995) (TV)
- *Video* identifiers concatenate movie title, release year (between brackets) and the letter *V* (between brackets). E.g.: Elton John: Live in Barcelona (1992) (V)
- *Video game* identifiers concatenate movie title, release year (between brackets) and the word *VG* (between brackets). E.g.: Harry Potter: Quidditch World Cup (2003) (VG)

3.2. IMDb target schema

The target schema for the IMDb data set consists in 24 tables describing movies and companies or persons related to movies. Figure 20 shows the tables of the target schema, their primary keys (underlined attributes) and foreign keys (arrows between tables). Additional dotted lines indicate *is-a* relationships among some tables and a (not implemented) relation describing persons. Table 6 describes each target table, its attributes and constraints.

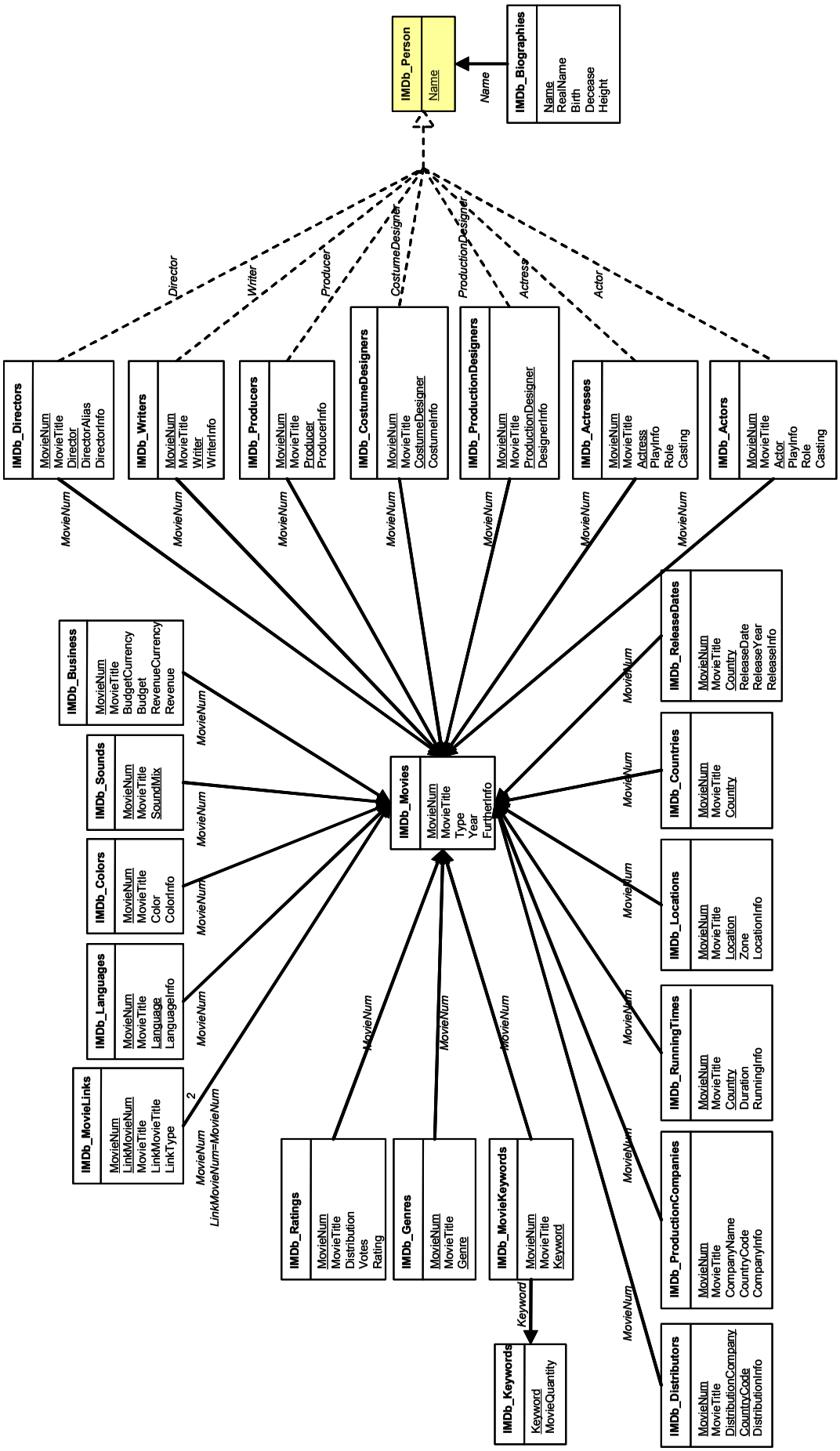

Figure 20 – IMDb schema

Table	Attributes	Constraints
IMDb_Movies *Information about movies*	– MovieNum: Numeric(7); autogenerated id – MovieTitle: String(250) – Type: String(2); {C=cinema, TV=television, V=video, VG=video game, S=serie, M=miniserie} – Year: String(45); an year, a range or a list of years – FurtherInfo: String(30)	Primary key: MovieNum Alternative key: MovieTitle
IMDb_Actors *Actors of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Actor: String(75) – PlayInfo: String(110); further info (uncredited, alias, special roles…) – Role: String(256); role name and/or description – Casting: Numeric(3); casting position	Primary key: MovieNum, Actor Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Actresses *Actresses of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Actress: String(75) – PlayInfo: String(110); further info (uncredited, alias, special roles…) – Role: String(256); role name and/or description – Casting: Numeric(3); casting position	Primary key: MovieNum, Actress Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Biographies *Personal data about actors, actresses, directors, producers and other people involved in movies*	– Name: String(70) – RealName: String(220) – Birth: String(130); date and place of birth – Decease: String(160); date, place and cause of decease – Height: String(15)	Primary key: Name
IMDb_Business *Budget and revenue of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – BudgetCurrency: String(3) – Budget: Numeric(15,2) – RevenueCurrency: String(3) – Revenue: Numeric(15,2)	Primary key: MovieNum Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Colors *Colors of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Color: String(20) – ColorInfo: String(50); further info (versions)	Primary key: MovieNum, Color Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Costume-Designers *Costume designers of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – CostumeDesigner: String(50) – CostumeInfo: String(100); further info (roles, alias, …)	Primary key: MovieNum, CostumeDesigner Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Countries *Countries of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Country: String(30)	Primary key: MovieNum, Country Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Directors *Directors of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Director: String(40) – DirectorAlias: String(80); director's appearing name in the movie – DirectorInfo: String(100); further info e.g. uncredited movies, co-direction, direction of episodes, segments or videos…	Primary key: MovieNum, Director Foreign keys: – MovieNum: ref. IMDb_Movies

IMDb_Distributors *Companies that distributed movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – DistributionCompany: String(110) – CountryCode: String(15) – DistributorInfo: String(140); further info (years, countries, media, …)	Primary key: MovieNum, DistributionCompany, CountryCode Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Genres *Genres of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Genre: String(12)	Primary key: MovieNum, Genre Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Keywords *Master of keywords*	– Keyword: String(60) – MovieQuantity: Numeric(5); the number of movies having the keyword	Primary key: Keyword
IMDb_Languages *Languages of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Language: String(30) – LanguageInfo: String(70); further info (versions)	Primary key: MovieNum, Language Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Locations *Running locations of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Location: String(170) – Zone: String(50) – LocationInfo: String(256); further info (years, countries, media, …)	Primary key: MovieNum, Location Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_MovieKeywords *Keywords of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Keyword: String(60)	Primary key: MovieNum, Keyword Foreign keys: – MovieNum: ref. IMDb_Movies – Keyword: ref. IMDb_Keywords
IMDb_MovieLinks *Links between movies*	– MovieNum: Numeric(7) – LinkMovieNum: Numeric(7); linked movie – MovieTitle: String(250) – LinkMovieTitle: String(250); linked movie – LinkType: String(30); references, remakes, etc.	Primary key: MovieNum, LinkMovieNum, LinkType Foreign keys: – MovieNum: ref. IMDb_Movies – LinkMovieNum: ref. IMDb_Movies
IMDb_Producers *Producers of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Producer: String(60) – ProducerInfo: String(90); further info (role, awards, …)	Primary key: MovieNum, Producer Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Production-Companies *Companies that produced movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – CompanyName: String(170) – CountryCode: String(15) – CompanyInfo: String(140); further info (years, places, participations, …)	Primary key: MovieNum, CompanyName, CountryCode Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Production-Designers *Production designers of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – ProductionDesginer: String(35) – DesignerInfo: String(140); further info (co-productions, roles, …)	Primary key: MovieNum, ProductionDesginer Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Ratings *Global ratings on movies (not differentiated par user)*	– Distribution: String(10); unknown meaning – Votes: Numeric(8,2); quantity of votes (decimals are always zero) – Rating: Number(4,2), value between 0 and 10 – MovieNum: Numeric(7) – MovieTitle: String(250)	Primary key: MovieNum Foreign keys: – MovieNum: ref. IMDb_Movies

IMDb_ReleaseDates *Dates of movie releases*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Country: String(40) – ReleaseDate: String(20) – ReleaseYear: Numeric(4) – ReleaseInfo: String(90); further info (release city/festival)	Primary key: MovieNum, Country, ReleaseDate Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_RunningTimes *Running times of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Country: String(40) – Duration: Numeric(5) – RunningInfo: String(75); further info (version, episodes, media, …)	Primary key: MovieNum, Country Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Sounds *Sound mix of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – SoundMix: String(40)	Primary key: MovieNum, SoundMix Foreign keys: – MovieNum: ref. IMDb_Movies
IMDb_Writers *Writers of movies*	– MovieNum: Numeric(7) – MovieTitle: String(250) – Writer: String(45) – WriterInfo: String(110); further info (role, awards, …)	Primary key: MovieNum, Writer Foreign keys: – MovieNum: ref. IMDb_Movies

Table 6 – IMDb schema

3.3. IMDb extraction processes

Contrarily to MovieLens data sets, the IMDb data set was very hard to extract and load in a relational database. The reason is that the data set is not conceived to be manipulated by DBMSs but by research engines. As a consequence, data is not presented in a tabular way and it is necessary to develop ad-hoc data extractors for loading tagged and hierarchical data and pre-processors for formatting data when it is presented in a pseudo-tabular format. This sub-section describes data extraction processes and next sub-section presents data cleaning and data transformation processes.

Data extraction includes pre-processing tasks for: (i) normalizing hierarchical structures, (ii) substituting multiple tabulations and blanks, and (iii) normalizing tagged structures. Then, pre-processed files are loaded into a relational database. We describe each task as follows:

3.3.1. Normalization of hierarchical structures

Many files organize data into two hierarchical levels. The outside level describes a movie feature (e.g. an actor) and the inside level lists all movies corresponding to such feature (see Figure 19). Consequently, only the first line corresponding to a movie feature contains it explicitly; the following lines only contain movie titles (and possible optional attributes). In order to store data in a tabular way, we need to iterate among the list of movies corresponding to each feature and insert it at each line. We do so in a pre-processing stage.

Pre-processing program is written in Java (jdk 1.4). It takes as input a list file, processes the file line per line and stores resulting lines in an output text file. We recognize lines containing a new feature from those containing movies of a previous feature because the formers start by a String (the feature value) and the latter start by a tabulation mark. A variable, storing the current feature value, is updated for each new feature found. White lines are ignored.

Table 7 (column hierarchies) indicates which source files need pre-processing for normalizing hierarchical structures.

3.3.2. Substitution of multiple tabulations and blanks

An important problem, present in most source files, is the use of a variable number of tabulations for separating columns. This disconcerts database loading programs and causes the storage of data at inappropriate attributes.

In order to prepare source files for being read by database loading programs, we pre-processed files, substituting multiple tabulations and blanks by a unique tabulation.

Pre-processing programs are written in Java (jdk 1.4). They take as input a list file, process the file line per line and store resulting lines in an output text file. The substitution function is quite simple; it searches for multiple occurrences of tabulations and/or blanks and substitutes them by a single tabulation.

Optional attributes need more attention. When there is a unique optional attribute and it is placed at the end of the line, the database loading interface fills null values automatically. However, when there are several optional attributes, we need to analyze to which attribute corresponds each value and (possibly) insert additional tabulations in order to signal the presence of a null value. Most files only contained comments as optional attributes, which were placed at the end of the line, so no further processing was needed. Conversely, actors.list and actresses.list files present three optional attributes: comments, role and casting position. We recognized their values because of special characters enclosing them. Specifically, comments, roles and casting positions are enclosed by brackets, square brackets and angle brackets respectively (see Figure 19). In all cases (including files having only comments), we did not eliminate enclosing characters; we deferred it to data cleaning steps.

Table 7 indicates which source files needed pre-processing for substituting multiple tabulations or blanks (column tabulations) and which ones needed the treatment of enclosing characters (we indicate attribute names enclosed by the corresponding special characters).

3.3.3. Normalization of tagged structures

Two files utilize tags for organizing data: biographies.list and business.list. Both files start each line with a two-letters tag that indicates the meaning of the line (the attribute to which it corresponds).

Biographies.list starts describing a new person by presenting a line with person name (tag NM). Subsequent lines describe different characteristics of a person. We extract real names (tag RN), date and place of birth (tag DB), date, place and cause of decease (tag DD) and height (tag HT). We ignore other tags and white lines. Analogously, business.list starts describing a new movie by presenting a line with its title (tag MV). Subsequent lines describe different characteristics of a movie. We extract budget currency and amount (tag BT), revenue currency and amount (tag RT). We ignore other tags and white lines.

File	Hierarchies	Tabulations	Enclosing characters	Tags
movies.list	no	yes		no
actors.list	yes	yes	(comments), [role], <casting>	no
actresses.list	yes	yes	(comments), [role], <casting>	no
biographies.list	no	no		yes
business.list	no	no		yes
color-info.list	no	yes	(comments)	no
costume-designers.list	yes	yes	(comments)	no
countries.list	no	yes		no
directors.list	yes	yes	(comments)	no
distributors.list	yes	yes	(comments)	no
genres.list	no	yes		no
keywords.list	no	yes		no
language.list	no	yes	(comments)	no
locations.list	no	yes	(comments)	no
movie-links.list	yes	yes		no
producers.list	yes	yes	(comments)	no
Production-companies.list	no	yes	(comments)	no
Production-designers.list	yes	yes	(comments)	no
ratings.list	no	no		no
release-dates.list	no	yes	(comments)	no
running-times.list	no	yes	(comments)	no
sound-mix.list	No	yes		no
writers.list	Yes	yes	(comments)	no

Table 7 – Pre-processing of source files

In order to store data in a tabular way, we need to iterate among lines corresponding to each person/movie and write a unique line with all available characteristics. We do so in a pre-processing stage.

Pre-processing program was written in Java (jdk 1.4). It takes as input a list file, processes the file line per line and stores resulting lines in an output text file. We store extracted data (corresponding to desired tags) in local variables. In both files, the desired tags are contained in a single line (other ignored tags are split in several lines). When we recognize a new person/movie (because of reading the corresponding tags), we store the line corresponding to the previous one.

Table 7 (column tags) indicates which source files needed pre-processing for normalizing tagged structures.

3.3.4. Loading into a relational database

After pre-processing tasks, files have a tabular form and can be treated by a database loading interface. We used Microsoft Access® loading interface, indicating file format and column format.

Note that the only file that did not need pre-processing is ratings.list, which is structured as fixed-length columns.

3.4. IMDb cleaning and transformation processes

Contrarily to MovieLens data sets, the IMDb data set presented a great number of anomalies and consequently needed the execution of several data cleaning and data transformation processes. The types of anomalies that we detected and solved are:

- We found 6 duplicate titles in movies.list. We keep only one occurrence.
- There were a great number of duplicate pairs of the form <movie, feature> for most features (specific quantities for each source file are shown in Table 11). In most cases, duplicates tuples had inconsistent values for remaining attributes. Our police was to substitute inconsistent values by null values (especially for comments), i.e. we considered inconsistent data as unknown. We did special treatment to some feature: in the case of budget and revenues (business.list) we summed inconsistent values (they were consider as partial data) and in the case of personal data (biographies.list) we kept the maximum value (in all inconsistencies, one of the values was null).
- There were a great number of violations to referential constraints (specific quantities for each source file are shown in Table 11). We eliminated such tuples.
- Some columns embedded multiple attributes. For example, the value “Taxi Films [uy]” of the production-companies.list embeds a production company (“Taxi Films”) and the code of its country (“Uruguay”). Eight files contained this type of anomalies (see Table 5). We solved them by splitting columns.
- Optional columns are delimited by special characters, e.g. square brackets or braces (specific enclosing characters are listed in Table 7). We eliminated enclosing characters in most cases. Exceptions were the occurrences of multiple comments, for example “(1963-1964) (USA) (TV) (original airing)”; we kept brackets because they delimitate each comment.
- Country names and codes, appearing in different source files (distributors.list, production-companies.list, running-times.list, locations.list, countries.list and release-dates.list) were not consistent (for example, we found the names ‘U.K.’ and ‘United Kindom’ for referring to the same country). We deferred the problem to data integration stage.
- As we did not extract all tags in business.list and biographies.list, we found an important number of tuples having null values for all columns excepting key (specific quantities for each source file are shown in Table 11). We filtered such tuples.
- The type of a movie (film, series episode, etc.) was not explicitly provided, however, IMDb site explains how titles are build (see Section 3.1). We calculated movie type following such explanations.
- Release year was calculated from release date (release-dates.list) by using typical date manipulation functions.
- The list of keyword included at the start of the keywords.list file, which is split in several columns, was hard to extract. We calculated the same information by grouping keywords of all movies and counting the number of movies that references each one.

ETL workflows consisted in the same tasks presented for the MovieLens data set but providing different implementations for specific cleaning and reconciliation tasks. The general workflow is illustrated in Figure 21 (P.txt represents either one of the text files obtained after pre-processing source files or ratings.list, which does not need pre-processing). However, for certain files some tasks were not necessary, specifically, if there were no transformations to do, or no duplications nor orphans were detected. Table 8 shows the tasks that are omitted for each file; bulk copy and duplicate check tasks are executed for all files.

In addition, as indexes by movie titles consumed much space and were less efficient, we auto generated new integer identifiers for movies. We included a new column (MovieNum), of auto-numeric type to the *movies-group* table (which stores the results of the grouping task). Each tuple inserted by the grouping task is set with a new sequential identifier generated automatically. We also maintained movie titles in tables referencing movies, letting users the possibility of choosing the identifier that is best adapted to their applications.

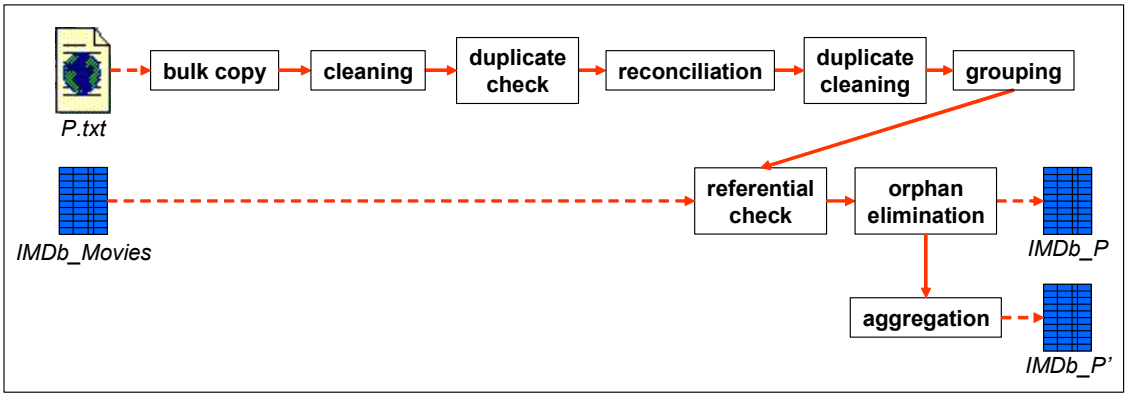

Figure 21 – ETL workflow for IMDb files

Source file	Cleaning	Reconc., dupl. cleaning and grouping	Ref. check	Orphan elimination	Aggregation
movies.txt			X	X	X
actors.txt	X				X
actresses.txt	X				X
biographies.txt			X	X	X
business.txt					X
color-info.txt	X				X
costume-designers.txt	X				X
countries.txt	X				X
directors.txt	X				X
distributors.txt					X
genres.txt	X				X
keywords.txt	X				
language.txt	X				X
locations.txt					X
movie-links.txt		X			X
producers.txt	X				X
production-companies.txt					X
production-designers.txt	X				X
ratings.list	X	X			X
release-dates.txt					X
running-times.txt					X
sound-mix.txt	X				X
writers.txt	X				X

Table 8 – ETL tasks that where not necessary to execute

In the following we describe such specific cleaning and reconciliation tasks, as well as special remarks:

– *Movie type calculation* task (cleaning for movies.txt) calculates movie types as derived attributes by looking for some special characters in the movie title (see Table 9). The task is implemented as a sequence of SQL operations (one for each movie type) that update the *Type* attribute according to Table 9, for example:

```
UPDATE movies-cleaning
SET Type=M
WHERE Mid(MovieTitle, 1, 2) = '"'
AND   Mid(MovieTitle, Len(MovieTitle)-5, 6) = '(mini)'
```

Movie type	Title is quoted	Special ending word	Generated attribute
Cinema film	No	none	C
Series episode	Yes	(mini)	S
Mini-series	Yes	none	M
TV series	No	(TV)	TV
Video	No	(V)	V
Video game	No	(VG)	VG

Table 9 – Movie type identification

– *Null values elimination* tasks (initial cleaning for biographies.txt and business.txt) eliminate tuples having null values for all attributes (excepting keys). Tasks are implemented as SQL operations like:

```
INSERT INTO business-cleaning (MovieTitle, Budget, Revenue)
SELECT *
FROM business-source
WHERE Bugnet IS NOT NULL OR Revenue IS NOT NULL;
```

The implementation for biographies.txt is analogous.

– *Splitting* tasks (cleaning for several tasks) update several attributes from one concatenating several concepts. For example, the ProductionCompany column of the productioncompanies.txt file embeds company names and country codes. The tasks search for special characters (e.g. brackets, square brackets, braces) and split attribute values according to such characters. Tasks are implemented as a sequence of SQL operations, for example:

```
INSERT INTO productioncompanies-cleaning
   (MovieTitle, ProductionCompany, CompanyInfo, Aux1)
SELECT MovieTitle, ProductionCompany, Comments,
       InStr(1,ProductionCompany,'[')
FROM productioncompanies-source;

UPDATE productioncompanies-cleaning
SET CompanyName = Mid(ProductionCompany, 1, Aux1-2),
   CountryCode = Mid(ProductionCompany,Aux1+1,Len(ProductionCompany)-Aux1-1)
WHERE Aux1 > 0;

UPDATE productioncompanies-cleaning
SET CompanyName = ProductionCompany
WHERE Aux1 = 0;
```

The first operation calculates the position of the special character and stores it in the Aux1 attribute. The second operation performs the split. The third operation is necessary when some values are optional (in the example, if country code can be omitted). It sets the first attribute and less the second as Null.

The implementation of splitting tasks for business.txt, distributors.txt, releasedates.txt and runningtimes.txt is similar. Table 10 lists attributes to split and special characters for those files. Columns of biographies.txt were not split because of their format diversity.

As a special case, the splitting of movielinks.txt is performed by table look-up, i.e. the first attribute (movie type) is looked up in a table; the remaining of the text corresponds to the second attribute. The look-up table contains the fifteen link types, namely: “alternate language version of”, “edited from”, “edited into”, “featured in”, “features”, “followed by”, “follows”, “referenced in”, “references”, “remade as”, “remake of”, “spin off”, “spoofed in”, “spoofs”, “version of”. The tasks is implemented as follows:

```
INSERT INTO movielinks-cleaning
   (MovieTitle, LinkMovieTitle, LinkType)
SELECT MovieTitle, Mid(Link, Len(LinkType), Len(Link)), LinkType
FROM movielinks-source X, MovieTypes Y
WHERE Mid(Link, 1, Len(LinkType)) = LinkType;
```

File	Multi-purpose column	First attribute	Second attribute	Separator
business.txt	Budget	BudgetCurrency	Budget	first blank
	Revenue	RevenueCurrency	Revenue	first blank
distributors.txt	DistributionCompany	DistributionCompany	CountryCode	[
movielinks.txt	Link	LinkType	LinkMovieTitle	look up
productioncompanies.txt	ProductionCompany	CompanyName	CountryCode(opt.)	[
releasedates.txt	ReleaseDate	Country	ReleaseDate	colon
runningtimes.txt	RunningTime	Country (optional)	Duration	colon

Table 10 – Splitting attributes

- *Enclosing character elimination* tasks (cleaning for several tables, listed in Table 7) deletes initial and ending character of a column, which enclose the value. The task is implemented as follows:

```
INSERT INTO languages-cleaning
   (MovieTitle, Language, LanguageInfo)
SELECT MovieTitle, Language, Mid(LanguageInfo,2,Len(LanguageInfo)-2)
FROM languages-source;
```

  The implementation for other files is analogous.

- *Zone calculation* task (cleaning for location.txt) extracts zone from location attribute. Specifically, location values are comma-separated lists of geographic places, from most precise to largest (e.g. "Toronto, Ontario, Canada" and "Biltmore Hotel - 506 S. Grand Ave., Downtown, Los Angeles, California, USA"). The number of items in each location value is variable. The Zone attribute will store the last item of a location, which is generally a country name. In order to calculate it, we need to iterate in the location value, searching the position of a comma and splitting the location value as described for previous task.

  The task is implemented as a sequence of SQL operations, as follows:

```
INSERT INTO locations-cleaning
   (MovieTitle, Location, Zone, LocationInfo)
SELECT MovieTitle, Location, Location, Comments
FROM locations-source;

UPDATE locations-cleaning
SET Zone = Mid(Zone, InStr(1,Zone,',')+2, Len(Zone))
WHERE InStr(1,Zone,',') > 0;
```

  The first operation sets zone as the complete location. The second operation splits zone according to the position of the first comma, storing the second part. Note that only tuples containing a comma are updated. The second operation is repeated until no tuple is updated.

- *Release year calculation* task (further cleaning for releasedates.txt) calculates release year from release date. A typical truncate function is used. The implementation of the task was presented in Section 2.3.2.

- *Reconciliation* tasks set conflictive values to Null. The task is implementation was discussed in Section 2.3.

- *Orphan elimination* tasks keep tuples joining the IMDb_Movies table, as described in Section 2.3. At this moment, we copy the auto-generated movie identifier (MovieNum) obtained from the IMDb_Movies table. The implementation for the genres.txt file is:

```
INSERT INTO genres-ref (MovieNum, MovieTitle, Genre)
SELECT Y.MovieNum, X.MovieTitle, X.Genre
FROM genres-group X, IMDb_Movies Y
WHERE X.MovieTitle = Y.MovieTitle;
```

The implementation for other files is analogous. Note that we execute the task even if no orphans were found, in order to obtain movie identifiers.

- *Referential check* task (for movielinks.txt) is executed twice in order to check that both, movie titles and link movie titles, are included in the IMDb_Movies table, generating two orphan tables. Implementation was described in Section 2.3.
- *Orphan elimination* task (for movielinks.txt) also considers that both, movie titles and link movie titles, are included in the IMDb_Movies table, i.e. it joins twice the IMDb_Movies table. The task is implemented as follows:

```
INSERT INTO movielinks-ref (MovieNum, LinkMovieNum, MovieTitle,
                            LinkMovieTitle, LinkType)
SELECT X.MovieNum, Y.MovieNum, M.MovieTitle, M.LinkMovieTitle, M.LinkType
FROM movielinks-group M, IMDb_Movies X, IMDb_Movies Y
WHERE M.MovieTitle = X.MovieTitle
AND   M.LinkMovieTitle = Y.MovieTitle;
```

- *Aggregation task* (for keywords.txt) stores the list of existing keywords and the number of movies related to each keyword. It is implemented as follows:

```
INSERT INTO IMDb_Keywords (Keyword, MovieQuantity)
SELECT Keyword, count(*)
FROM IMDb_MovieKeywords
GROUP BY Keyword;
```

As a summary of ETL tasks, Table 11 shows the number of extracted tuples, the number of anomalies, duplicates and orphans found, and the number of exported tuples.

Source file	Extracted tuples	Cleaning anomalies	Duplicates	Orphans	Exported tuples	Resulting table
movies.list	858.967	0	6		858.961	IMDb_Movies
actors.list	5.014.897	0	40	486.875	4.527.982	IMDb_Actors
actresses.list	2.743.802	0	35	327.935	2415832	IMDb_Actresses
biographies.list	312.837	54.011	8		258818	IMDb_Biographies
business.list	57.364	31.847	1	0	25516	IMDb_Business
color-info.list	433.117	0	846	44.541	387730	IMDb_Colors
costume-designers.list	99.192	0	35	7.644	91513	IMDb_CostumeDesigners
countries.list	530.295	0	681	0	529614	IMDb_Countries
directors.list	513.344	0	295	34	513015	IMDb_Directors
distributors.list	403.271	0	15.023	0	388248	IMDb_Distributors
genres.list	637.976	0	606	57.519	579851	IMDb_Genres
keywords.list	1.124.778	0	507	330	32704	IMDb_Keywords
					1123941	IMDb_MovieKeywords
language.list	445.840	0	1.355	0	444485	IMDb_Languages
locations.list	216.951	0	45	0	216906	IMDb_Locations
movie-links.list	533.428	0	0	36.572	496856	IMDb_MovieLinks
producers.list	1.038.128	0	422.088	172.775	443265	IMDb_Producers
production-companies.list	478.346	0	831	0	477515	IMDb_ProductionCompanies
production-designers.list	103.347	0	12	7509	95826	IMDb_ProductionDesigners
ratings.list	159.957	0	0	66	159891	IMDb_Ratings
release-dates.list	830.416	0	2.007	53.556	774853	IMDb_ReleaseDates
running-times.list	327.312	0	4.405	49.901	273006	IMDb_RunningTimes
sound-mix.list	231.318	0	496	8.512	222310	IMDb_Sounds
writers.list	775.660	0	25.260	130.812	619588	IMDb_Writers

Table 11 – Statistics of data extraction, cleaning and transformation

4. Data integration

This section describes the construction of an integrated database from the extracted data. We start describing the difficulty of matching movie identifiers and describing the matching process. We then describe the integrated schema and its construction.

4.1. Matching processes

We extracted 858.961 movies from IMDb and 3.883 movies from MovieLens. After integration, we constated that all MovieLens movies were included in the IMDb data set. However, matching titles was not trivial. Furthermore, only 79% of matches had identical movie titles in both data sets. Other matching strategies were implemented, including manual look-up, in order to match the remaining titles. This section presents an overview of the matching process; details can be found in [Peralta 2007a].

Both, MovieLens and IMDb identify movies by ad-hoc identifiers (called movie titles) that concatenate title and running year (see Sections 2.1.1and 3.1). Note that we do not consider series episodes, mini-series, TV series, videos or video games (which have different identifiers) because MovieLens data set only includes cinema movies. But even identifiers are built in the same manner, we found discrepancies in titles. Most typical causes are inclusion of special characters, typing errors, transposition or omission of articles, use of different running years, translation of foreign titles and use of alternative titles. Table 12 illustrates differences in movie titles.

IMDb titles	MovieLens titles	Causes
¡Three Amigos! (1986)	Three Amigos! (1986)	Special characters
Shall We Dance (1937)	Shall We Dance? (1937)	
8 ½ Women (1999)	8 1/2 Women (1999)	
One Hundred and One Dalmatians (1961)	101 Dalmatians (1961)	
Two Moon Junction (1988)	Two Moon Juction (1988)	Typing errors
La Bamba (1987)	Bamba, La (1987)	Transposed articles
El Dorado (1966)	Dorado, El (1967)	
Three Ages (1923)	Three Ages, The (1923)	Omitted articles
Story of G.I. Joe (1945)	Story of G.I. Joe, The (1945)	
Tarantella (1996)	Tarantella (1995)	Different years
Supernova (2000/I)	Supernova (2000)	
Abre los ojos (1997)	Open Your Eyes (Abre los ojos) (1997)	English titles
Caro diario (1994)	Dear Diary (Caro Diario) (1994)	
Huitième jour, Le (1996)	Eighth Day, The (Le Huitième jour) (1996)	
Historia oficial, La (1985)	Official Story, The (La Historia Oficial) (1985)	
Cité des enfants perdus, La (1995)	City of Lost Children, The (1995)	
Star Wars (1977)	Star Wars: Episode IV - A New Hope (1977)	Alternative titles
Santa Claus (1985)	Santa Claus: The Movie (1985)	
Sunset Blvd. (1950)	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	
Sugar Hill (1994)	Harlem (1993)	

Table 12 – Examples of movies having different titles

In order to match MovieLens titles with IMDb titles (included in ML_Movies and IMDb_Movies tables), we followed several matching strategies. Each strategy tries to match MovieLens titles not matched by previous strategies as shown in Figure 22. To this end, each strategy considers a different heuristic for building candidate (alternative) titles for unmatched movies and compares them with IMDb titles. We implemented the following strategies:

- *Join by movie title*: This strategy computes the exact match. We joined IMDb_Movies and ML_Movies tables by the MovieTitle attribute. We obtained 3086 matches (79%).
- *Match using the MovieLens small data set*: This strategy uses the SML_Movies table (small data set) as a mapping table between unmatched MovieLens titles and IMDb titles. Specifically, the SML_Movies table contains the URL of the IMDb web page corresponding to each movie. We formatted the URL and replaced special characters (e.g. %20 represents a blank) obtaining a candidate title. We joined this table with unmatched MovieLens movies (by the MovieTitle attribute) and with IMDb_Movies (by the just

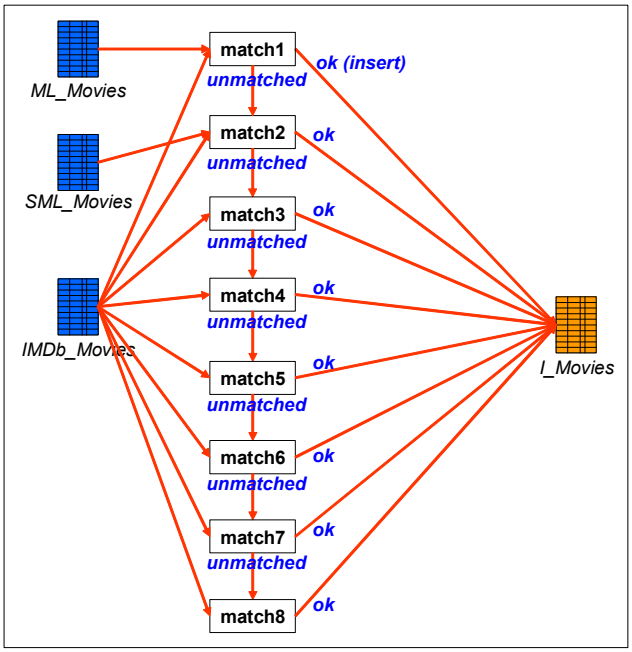

Figure 22 – Matching process

built candidate title). We had several difficulties: First, the intersection of both MovieLens collections is not very large. Second, we cannot replace all special characters. As a result, we obtained 83 new matches, totalizing 3169 matches (82%).

- *Match extracting foreign title*: Some MovieLens titles are translated to English but include, between brackets, the original title (see examples in Table 12). We extracted original titles (text between brackets) of unmatched MovieLens movies obtaining a candidate title. Then, we joined them with IMDb_Movies. We obtained 92 new matches, totalizing 3261 matches (84%).
- *Match ignoring running year*: Sometimes, movie titles embeds running years, sometimes they embed diffusion years and sometimes they include additional characters (e.g. '1999/I'). This strategy consists in ignoring years for joining movie titles and manually verifying the obtained matches. We removed years from movie titles of unmatched MovieLens movies and IMDb_Movies. Then, we joined these auxiliary titles storing matches in a temporal table. A human validated matches (e.g. those differentiating in only one year) and eliminated erroneous ones. As a result, we obtained 295 new matches, totalizing 3556 matches (92%).
- *Matching of 20 first characters*: This strategy consists in truncating titles to 20 characters, joining them and manually verifying the obtained matches. We tried to solve some kinds of alternative titles (e.g. "Friday the 13th Part III (1982)" and "Friday the 13th Part 3: 3D (1982)") or special characters or truncations (e.g. "Why Do Fools Fall In Love? (1998)" and "Why Do Fools Fall In Love (1998)"). We truncated titles of unmatched MovieLens movies and IMDb_Movies. Then, we joined these auxiliary titles storing matches in a temporal table. A human validated matches by comparing whole titles and eliminated erroneous matches. As a result, we obtained 34 new matches, totalizing 3590 matches (92%).
- *Matching of 10 first characters*: This strategy repeats the previous one, but truncating titles to 10 characters. We obtained 46 new matches, totalizing 3636 matches (94%).
- *Manual look-up*: This strategy consists in manually examining unmatched MovieLens titles looking for possible matches in the IMDb_Movies table. We used intuition for finding titles in the huge IMDb collection (e.g. transposing words for "La Bamba (1987)" and "Bamba, La (1987)") and knowledge of foreign languages (e.g. "Cité des enfants perdus, La (1995)" and "City of Lost Children, The (1995)"). We obtained 116 new matches, totalizing 3752 matches (97%).
- *Web look-up*: The last strategy uses the search engine of MovieLens web site to find unmatched movies. Then, we used the link provided by MovieLens site to access the movie page at IMDb and we copied its title. We matched the remaining 131 movie titles.

After matching titles, we found that 2 movies had the same corresponding movie at IMDb, i.e. we detected 2 additional duplicates. Tuples with original title were kept; those with foreign title were removed.

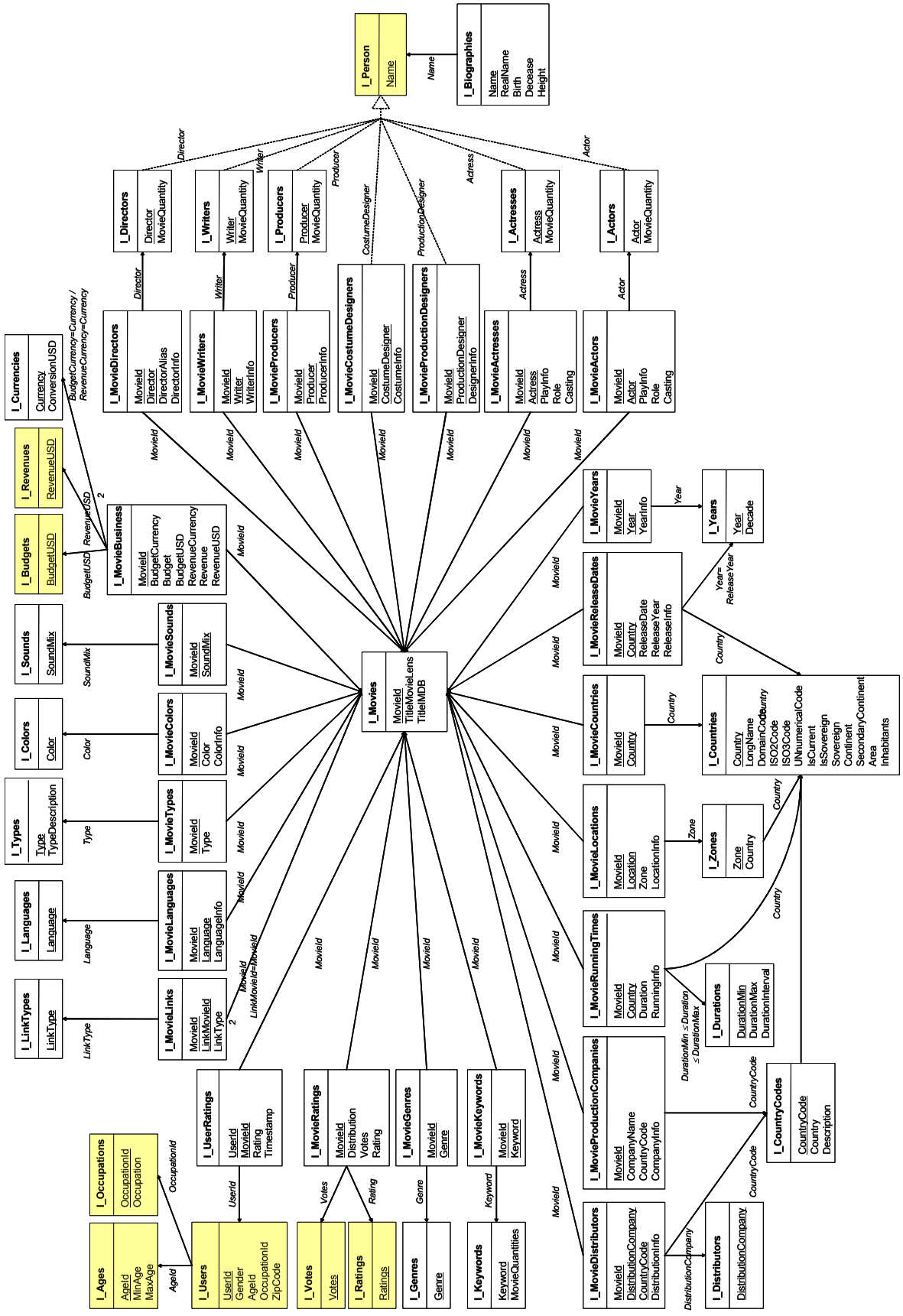

Figure 23 – Integrated schema

4.2. Integrated schema

The integrated schema consists in 52 tables describing movies, companies and persons related to movies and the users that evaluated movies. Figure 23 shows the tables of the integrated schema, their primary keys (underlined attributes) and foreign keys (arrows between tables). Additional dotted lines relate some tables to a fictitious (not implemented) relation, describing persons. Shadow tables were used for the construction of the referential but are not visible for making queries. Table 13 describes each table, its attributes and constraints.

Table	Attributes	Constraints
I_Movies *Join between IMDb and MovieLens*	– MovieId: Numeric(4); MovieLens' id – TitleMovieLens: String(100) – TitleImdb: String(250)	Primary key: MovieId Unique: TitleImdb
I_Actors *Master of actors*	– Actor: String(75) – MovieQuantity: Numeric(3); the number of played movies	Primary key: Actor
I_Actresses *Master of actresses*	– Actress: String(75) – MovieQuantity: Numeric(3); the number of played movies	Primary key: Actress
I_Ages *Age intervals*	– AgeId: Numeric(2) – MinAge: Numeric(2) – MaxAge: Numeric(2)	Primary key: AgeId
I_Biographies *Biographies of actors, actresses, directors, producers and other people involved in movies*	– Name: String(70) – RealName: String(220) – Birth: String(130); date and place of birth – Decease: String(160); date, place and cause of decease – Height: String(15)	Primary key: Name
I_Budgets *Master of budget intervals*	– BudgetUSD: Numeric(15,2); start of an interval (internal use)	Primary key: BudgetUSD
I_Colors *Master of colors*	– Color: String(20)	Primary key: Color
I_Countries *Master of countries, providing several aggregation criteria and international country codes*	– Country: String(40) – LongName: String(110) – DomainCode: String(2); internet domain – ISO2Code: String(2); ISO3166-1-alpha2 code – ISO3Code: String(3); ISO3166-1-alpha2 code – UNnumericalCode: Numeric(3); united nations country code – IsCurrent: Numeric(1); 1 for current countries, 0 for old ones – IsSovereign: Numeric(1); 1 for sovereign UN nations, 2 for sovereign non-UN nations, 3 for sovereign non-recognized nations, 4 for dependent territories and 5 for areas of special sovereignty – Sovereign: String(40); name of sovereign nation (current country in the case of old countries) – Continent: String(20) – SecondaryContinent: String(20); for countries having territories in two continents (the one with the highest are is taken as main continent) – Area: Numeric(8) – Inhabitants: Numeric(10)	Primary key: Country

I_CountryCodes *Codes of countries*	– CountryCode: String(15) – Country: String(40) – Description: String(50); further info (regions, deprecations...)	Primary key: CountryCode Foreign keys: – Country: ref. I_Countries
I_Currencies *Master of currencies*	– Currency: String(3) – ConversionUSD: Numeric(12,10); at October 2006	Primary key: Currency
I_Directors *Master of directors*	– Director: String(40) – MovieQuantity: Numeric(3); the number of directed movies	Primary key: Director
I_Distributors *Master of distribution companies*	– DistributionCompany: String(70)	Primary key: DistributionCompany
I_Durations *Master of duration intervales*	– DurationMin: Numeric(5) – DurationMin: Numeric(5) – DurationInterval: Numeric(5)	Primary key: DurationMin
I_Genres *Master of genres*	– Genre: String(12)	Primary key: Genre
I_Keywords *Master of keywords*	– Keyword: String(60) – MovieQuantity: Numeric(5); the number of movies having the keyword	Primary key: Keyword
I_Languages *Master of languages*	– Language: String(30)	Primary key: Language
I_LinkTypes *Types of links between movies*	– LinkType: String(30); references, remakes, etc.	Primary key: LinkType
I_MovieActors *Actors of movies*	– MovieId: Numeric(4) – Actor: String(75) – PlayInfo: String(110); further info (uncredited, alias, special roles…) – Role: String(256); role name and/or description – Casting: Numeric(3); casting position	Primary key: MovieId, Actor Foreign keys: – MovieId: ref. I_Movies – Actor: ref. I_Actors
I_MovieActresses *Actresses of movies*	– MovieId: Numeric(4) – Actress: String(75) – PlayInfo: String(110); further info (uncredited, alias, special roles…) – Role: String(256); role name and/or description – Casting: Numeric(3); casting position	Primary key: MovieId, Actress Foreign keys: – MovieId: ref. I_Movies – Actress: ref. I_Actresses
I_MovieBusiness *Budget and revenue of movies*	– MovieId: Numeric(4) – BudgetCurrency: String(3) – Budget: Numeric(15,2) – BudgetUSD: Numeric(15,2) – RevenueCurrency: String(3) – Revenue: Numeric(15,2) – RevenueUSD: Numeric(15,2)	Primary key: MovieId Foreign keys: – MovieId: ref. I_Movies – BudgetCurrency: ref. I_Currencies – RevenueCurrency: ref. I_Currencies
I_MovieColors *Colors of movies*	– MovieId: Numeric(4) – Color: String(20) – ColorInfo: String(50); further info (versions)	Primary key: MovieId, Color Foreign keys: – MovieId: ref. I_Movies – Color: ref. I_Colors
I_MovieCostume-Designers *Costume designers of movies*	– MovieId: Numeric(4) – CostumeDesigner: String(50) – CostumeInfo: String(100); further info (roles, alias, …)	Primary key: MovieId, CostumeDesigner Foreign keys: – MovieId: ref. I_Movies

I_MovieCountries *Countries of movies*	– MovieId: Numeric(4) – Country: String(40)	Primary key: MovieId, Country Foreign keys: – MovieId: ref. I_Movies – Country: ref. I_Countries
I_MovieDirectors *Countries of movies*	– MovieId: Numeric(4) – Director: String(40) – DirectorAlias: String(40); director's appearing name in the movie – DirectorInfo: String(70); further info e.g. uncredited movies, co-direction, direction of episodes, segments or videos…	Primary key: MovieId, Director Foreign keys: – MovieId: ref. I_Movies – Director: ref. I_Directors
I_MovieDistributors *Companies that distributed movies*	– MovieId: Numeric(4) – DistributionCompany: String(70) – CountryCode: String(15) – DistributorInfo: String(100); further info (years, countries, media, …)	Primary key: MovieId, DistributionCompany, CountryCode Foreign keys: – MovieId: ref. I_Movies – DistributionCompany: ref. I_Distributors – CountryCode: ref. I_CountryCodes
I_MovieGenres *Genres of movies*	– MovieId: Numeric(4) – Genre: String(12)	Primary key: MovieId, Genre Foreign keys: – MovieId: ref. I_Movies – Genre: ref. I_Genres
I_MovieKeywords *Keywords of movies*	– MovieId: Numeric(4) – Keyword: String(60)	Primary key: MovieId, Keyword Foreign keys: – MovieId: ref. I_Movies – Keyword: ref. I_Keywords
I_MovieLanguages *Languages of movies*	– MovieId: Numeric(4) – Language: String(30) – LanguageInfo: String(70); further info (versions)	Primary key: MovieId, Language Foreign keys: – MovieId: ref. I_Movies – Language: ref. I_Languages
I_MovieLinks *Links between movies*	– MovieId: Numeric(4) – LinkMovieId: Numeric(4) – LinkType: String(30); references, remakes, etc.	Primary key: MovieId, LinkMovieId, LinkType Foreign keys: – MovieId: ref. I_Movies – Link MovieId: ref. I_Movies – LinkType: ref. I_LinkTypes
I_MovieLocations *Running locations of movies*	– MovieId: Numeric(4) – Location: String(140) – Zone: String(30) – LocationInfo: String(256); further info (years, countries, media, …)	Primary key: MovieId, Location Foreign keys: – MovieId: ref. I_Movies – Zone: ref. I_Zones
I_MovieProducers *Producers of movies*	– MovieId: Numeric(4) – Producer: String(60) – ProducerInfo: String(90); further info (role, awards, …)	Primary key: MovieId, Producer Foreign keys: – MovieId: ref. I_Movies – Producer: ref. I_Producers
I_MovieProduction-Companies *Companies that produced movies*	– MovieId: Numeric(4) – CompanyName: String(120) – CountryCode: String(15) – CompanyInfo: String(140); further info (years, places, participations, …)	Primary key: MovieId, CompanyName, CountryCode Foreign keys: – MovieId: ref. I_Movies – CountryCode: ref. I_CountryCodes
I_MovieProduction-Designers *Production designers of movies*	– MovieId: Numeric(4) – ProductionDesginer: String(35) – DesignerInfo: String(140); further info (co-productions, roles, …)	Primary key: MovieId, ProductionDesginer Foreign keys: – MovieId: ref. I_Movies

I_MovieRatings *Global ratings on movies (not differentiated par user)*	– MovieId: Numeric(4) – Distribution: String(10); unknown meaning – Votes: Numeric(8,2); quantity of votes (decimals are always zero) – Rating: Number(4,2), value between 0 and 10	Primary key: MovieId Foreign keys: – MovieId: ref. I_Movies
I_MovieReleaseDates *Dates of different releases*	– MovieId: Numeric(4) – Country: String(40) – ReleaseDate: String(20) – ReleaseYear: Numeric(4) – ReleaseInfo: String(90); further info (release city/festival)	Primary key: MovieId, Country, ReleaseDate Foreign keys: – MovieId: ref. I_Movies – ReleaseYear: ref. I_Years – Country: ref. I_Countries
I_MovieRunningTimes *Running times of movies*	– MovieId: Numeric(4) – Country: String(40) – Duration: Numeric(5) – RunningInfo: String(75); further info (version, episodes, media, …)	Primary key: MovieId, Country Foreign keys: – MovieId: ref. I_Movies – Country: ref. I_Countries
I_MovieSounds *Sound mix of movies*	– MovieId: Numeric(4) – SoundMix: String(40)	Primary key: MovieId, SoundMix Foreign keys: – MovieId: ref. I_Movies – SoundMix: ref. I_Sounds
I_MovieTypes *Types of movies*	– MovieId: Numeric(4) – Type: String(2); {C=cinema, TV=television, V=video, VG=video game, S=serie, M=miniserie}	Primary key: MovieId Foreign keys: – MovieId: ref. I_Movies – Type: ref. I_Types
I_MovieWriters *Writers of movies*	– MovieId: Numeric(4) – Writer: String(45) – WriterInfo: String(110); further info (role, awards, …)	Primary key: MovieId, Writer Foreign keys: – MovieId: ref. I_Movies – Writer: ref.: I_Writers
I_MovieYears *Years of movies*	– MovieId: Numeric(4) – Year: Numeric(4) – YearInfo: String(30); further info (shot years)	Primary key: MovieId, Year Foreign keys: – MovieId: ref. I_Movies – Year: ref. I_Years
I_Occupations *User occupations*	– OccupationId: Numeric(2) – Occupation: String (25)	Primary key: OccupationId
I_Producers *Master of producers*	– Producer: String(60) – MovieQuantity: Numeric(3); the number of produced movies	Primary key: Producer
I_Ratings *Master of ratings*	– Rating: Number(4,2), value between 0 and 10 (internal use, for classifying ratings)	Primary key: Rating
I_Revenues *Master of revenue intervals*	– RevenueUSD: Numeric(15,2); start of an interval (internal use)	Primary key: RevenueUSD
I_Sounds *Master of sounds*	– SoundMix: String(40)	Primary key: SoundMix
I_Types *Types of movies*	– Type: String(2); {C=cinema, TV=television, V=video, VG=video game, S=serie, M=miniserie} – TypeDescription: String(10)	Primary key: Type
I_UserRatings *User ratings on movies*	– UserId: Numeric(4) – MovieId: Numeric(4) – Rating: Number(1), value in {1,2,3,4,5} – Timestamp: Numeric(9); represents milliseconds from an initial time	Primary key: UserId, MovieId Foreign keys: – UserId: ref. I_Users – MovieId: ref. I_Movies

I_Users *Descriptive information about users*	– UserId: Numeric(4) – Gender: Char(1); value in {M,F} – Age: Numeric(2) – Occupation: Numeric(2) – ZipCode: String(10)	Primary key: UserId Foreign keys: – Age: ref. I_Ages – Occupation: ref. I_Occupation
I_Votes *Master of votes*	– Votes: Numeric(8,2); indicates the start of a voting interval (internal use, for classifying votes)	Primary key: Votes
I_Writers *Master of writers*	– Writer: String(45) – MovieQuantity: Numeric(3); the number of written movies	Primary key: Writer
I_Years *Master of years*	– Year: Numeric(4) – Decade: Numeric(4)	Primary key: Year
I_Zones *Master of geographical zones*	– Zone: String(30); a country, region, sea, … – Country: String(40)	Primary key: Zone Foreign keys: – Country: ref. I_Countries

Table 13 – Integrated schema

4.3. Construction of the integrated database

Matching MovieLens and IMDb movie titles was the most difficult task in the construction of the integrating database. The remaining tasks basically consist in joining the I_Movies table (the master of movies resulting from the matching process) in order to keep movie features of movies included in the master (i.e. assuring referential integrity). This allowed the creation of the appropriate foreign keys. In addition, we created master tables for most movie features. This section describes these tasks.

Figure 24 shows an overview of the process for building tables of the integrated database. In order to filter features of movies non-included in the master of movies, we use the *orphan elimination* task described in Sub-section 2.3, which simply joins the IMDb feature table (IMDb_P in Figure 24) with the I_Movies table. Then, some master tables were aggregated using the *aggregation* task, also described in Sub-section 2.3, which group tuples according to a feature attribute. The aggregation task was not executed for some feature tables (I_MovieCostumeDesigners, I_MovieProductionDesigners) because they have too few repetitive values (for example, each entry of the I_MovieCustomeDesigners table describes a different costume designer). The master tables generated for each table can be deduced from Figure 23, because they coincide with referential constraints (excepting those referencing the I_Movies table).

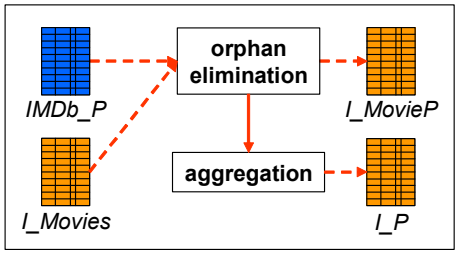

Figure 24 – Construction of tables of the integrated database

Three special data reconciliation tasks were performed for years, currencies and countries. For years, we performed the union of years appearing in I_MovieYears and I_MovieReleaseDates tables and then, we calculated *decade* as a derived attribute, using typical date manipulation functions (similar to the *year calculation* task described in Sub-section 2.3.2). For currencies, we obtained budget and revenue currencies from the I_MovieBusiness table. We obtained dollar exchange (at October 2006) for such currencies from different Internet sites, e.g. [1] (a better exchange should be obtained by considering currencies at movie running time; we refer this to future work). For countries, we had country names in the I_MovieCountries and I_MovieReleaseDates tables, in addition to some zones of the I_MovieLocations table and we had country codes in the I_MovieDistributors and I_MovieProductionCompanies tables. It was a big mess! Country names were written in different manners (e.g. 'UK' and 'United Kingdom') and sometimes referenced non-sovereign territories (e.g. French Polynesia) or non-current countries (e.g. 'URSS'), country codes were also ambiguous.

In order to build the master of countries (I_Countries) we downloaded United Nations official country list [10], which includes sovereignty status of countries (UN member sovereign states, non-UN member sovereign states, no general international recognition sovereign states, dependent territories and areas of special sovereignty). Sovereign countries of non-sovereign territories were consulted at Wikipedia [11]. We obtained different country codes (ISO3166-1 alpha 2 code, ISO3166-1 alpha 3 code, UN numerical code and internet domain code) and further descriptive data (continent, area and inhabitants) from the Schmidt lists of countries [8] and continents [9]. As a country can have territory in two continents, we defined as primary continent, the one in which the country has the biggest area. There were inconsistencies between UN official country names and those obtained from Wikipedia and Schmidt lists, but they were manually solved without major problems.

We created mapping tables in order to join the master with I_MovieCountries and I_MovieReleaseDates tables. Country names included in such tables but not in the master were manually disambiguated and related to one of the existing countries (when it was an alternative name of such country) or added as non-current country (when it was an old country name). I_MovieCountries and I_MovieReleaseDates tables were updated setting country names according to the mapping tables. Two special country names were created: (i) '_unknown' for Null values, and (ii) and _multiple for special cases, such as 'Korea', 'Serbia and Montenegro' or 'Soviet Union' that correspond to multiple current countries.

The master of country codes was created from the master of countries, selecting attributes country name and domain code. Mapping tables were created analogously in order to join the master with I_MovieDistributors and I_MovieProductionCompanies. We manually disambiguated country codes in an analogous way. We found several cases of far territories (e.g. 'French Guiana') which were associated to the corresponding sovereign country, multiple countries (e.g. 'us/ca' referencing United States and Canada) which were associated to the '_multiple' special country name and unknown countries codes (e.g. 'cshh' and 'ddde') which were associated to the '_unknown' special country name.

The master of zones was also created from the I_MovieLocations table, adding an attribute for storing a country name when the zone coincides with a country. The table was joined with the master of countries, and non-referenced zones were manually examined in order to determine if they correspond to a country (setting the country attribute with the appropriate value) or other types of zones, e.g. 'Persian Gulf' (setting the country attribute with the '_multiple' value).

As summary, the construction of the integrated database basically consisted in intersection integrated movie titles with IMDb tables describing movie features. In addition, we built master tables (sometimes integrating several IMDb tables) and integrating external data sources (country lists and currencies). For recapitulating, Table 14 shows the tables of the integrated database and their number of tuples.

5. Conclusion

In this report we described the procedure followed for extracting and integrating MovieLens and IMDb data. Specifically, we described source and target schemas and the algorithms that perform data extraction, data cleaning and data transformation and data integration.

Major difficulties were found at:

- IMDb data extraction. We needed to develop ad-hoc pre-processing routines in order to normalize hierarchical and tagged structures and substitute multiple tabulations and blanks.
- IMDb data transformation and cleaning. We found a lot of anomalies in extracted data, ranging from duplicate and orphan tuples to multiple attributes embedded in single columns. We needed to develop several cleaning and transformation tasks.
- Matching of MovieLens and IMDb titles. We implemented 8 different matching strategies, some of them including human interaction. This was the most time-consuming task.
- Aggregation of some master tables, especially, the master of countries. We extracted external data in order to solve conflictive country names and codes and lead with non-sovereign and non-current countries.

Data extracted from MovieLens and IMDb sites, as well as the integrated database are available at the APMD project web site [7].

In this report we do not dealt with maintenance issues but we would like to treat some issues as future work. Specifically, as IMDb lists are continually updated, it may be interesting to adapt extraction, cleaning and

integration processes to incorporate new data. The problem seams to be more difficult for updates of MovieLens data (which are not announced by the moment) because manual reconciliation tasks will probably be necessary.

Master tables	Tuples
I_Movies	3881
I_Actors	63207
I_Actresses	31720
I_Biographies	258818
I_Colors	2
I_Countries	256
I_CountryCodes	265
I_Currencies	95
I_Directors	2170
I_Distributors	2391
I_Durations	6
I_Genres	25
I_Keywords	14974
I_Languages	85
I_LinkTypes	15
I_Producers	8142
I_Sounds	28
I_Types	6
I_Writers	5458
I_Years	90
I_Zones	133

Relationship tables	Tuples
I_MovieActors	110548
I_MovieActresses	48423
I_MovieBusiness	1740
I_MovieColors	3898
I_MovieCostumeDesigners	3373
I_MovieCountries	4993
I_MovieDirectors	4141
I_MovieDistributors	23060
I_MovieGenres	9288
I_MovieKeywords	105128
I_MovieLanguages	4796
I_MovieLinks	18160
I_MovieLocations	14345
I_MovieProducers	16749
I_MovieProductionCompanies	9517
I_MovieProductionDesigners	3152
I_MovieRatings	3861
I_MovieReleaseDates	46554
I_MovieRunningTimes	4839
I_MovieSounds	4928
I_MovieTypes	3881
I_MovieWriters	8766
I_MovieYears	3881

Auxiliary tables	Tuples
I_Ages	7
I_Budgets	6
I_Occupations	21
I_Ratings	10
I_Revenues	6
I_UserRatings	1000194
I_Users	6040
I_Votes	8

Table 14 – Statistics of data integration

6. References

[1] European Commision: "List of countries, territories and currencies", Web Report, March 2007. URL: http://publications.europa.eu/code/en/en-5000500.htm, last accessed on April 5th, 2007.

[2] GroupLens Research: "movielens: helping you to find the right movies". Web site, ULR: http://movielens.umn.edu, last accessed on July 9th, 2006.

[3] GroupLens Research: "MovieLens Data Sets". Web site, ULR: http://grouplens.org/node/12#attachments, last accessed on October 5th, 2006.

[4] Intenet Movie Database, Inc.: "The Intenet Movie Database", Web site, URL: http://www.imdb.com/, last accessed on July 9th, 2007.

[5] Intenet Movie Database, Inc.: "Alternate interfaces", Web Page, URL: http://www.imdb.com/interfaces, last accessed on October 5th, 2006.

[6] Peralta, V.: "Matching of MovieLens and IMDb Movie Titles". Technical Report, Laboratoire PRiSM, Université de Versailles, Versailles, France, March 2007.

[7] Peralta, V.: "APMD Test Platform". URL: http://apmd.prism.uvsq.fr/TestPlatform/, last accessed on May 30th, 2007.

[8] Schmidt M.: "List of countries and territories". Web page, URL: http://schmidt.devlib.org/data/countries.html, last accessed on April, 10th, 2007.

[9] Schmidt M.: “Continents”. Web page, URL: http://schmidt.devlib.org/data/continents.html, last accessed on April, 10th, 2007.

[10] United Nations Cartographic Section: “List of Territories”. PDF file, 2004. URL: http://www.un.org/Depts/Cartographic/english/geoname.pdf, last accessed on April 10th, 2007.

[11] Wikipedia: “List of countries”. Web page, URL: http://en.wikipedia.org/wiki/List_of_countries, last accessed on April 10th, 2007.