Towards a Concurrent Implementation of Keyword Search Over Relational Databases

by

Richard J.I. Drake

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science (M.Sc.)

in

The Faculty of Science

Computer Science

University of Ontario Institute of Technology

Supervisor: Dr. Ken Q. Pu

December 2013

Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

RDBMS relational database management system. 8, 11

List of Symbols

N number of documents in collection. 12

M size of T. 12

T all terms in document collection. ix, 12

Chapter 1

Background (2 days)

Literature search on:

- DBExplore
- XRank
- BANKS
- ...

Chapter 2

A Tale of Two Data Models

The term "data model" refers to a notation for describing data and/or information. It consists of the data structure, operations that may be performed on the data, as well as constraints placed on the data [GUW09].

In this chapter we provide a formal definition of the relational data model, discuss its merits, its shortcomings, and contrast it to the document data model. Contrary to the relational model, the document model permits fast and flexible keyword search without requiring explicit domain knowledge of the data. In addition, we demonstrate the feasibility of encoding a relational model into a document model in a lossless manner.

2.1 Relational Model

In its most basic form, the relational data model is built upon sets and tuples. Each of these sets consist of a set of finite possible values. Tuples are constructed from these sets to form relations.

Definition 1 (Named Tuple). A named tuple t is an instance of a relation , consisting of values corresponding to the attributes of . For example,

$$t = \{\text{name} : \text{“Superman”}, \text{age} : 33, \text{birthplace} : \text{“Krypton”}\}$$

We denote the attributes of t as $\text{ATTR}[t] = \{\text{name}, \text{age}\}$ The values are $t[\text{name}] =$ "Superman", $t[\text{age}] = 33$, and $t[\text{birthplace}] =$ "Krypton",

Definition 2 (Relation). A relation r is a set of named tuples, $r = \left\{t_1, t_2, \ldots, t_n\right\}$, such that all the named tuples share the same attributes.

$$\forall t, t' \in r, \text{ATTR}[t] = \text{ATTR}[t']$$

For example,

$$r = \left\{ \begin{array}{l} \left\{\text{name} : \text{“Superman”}, \text{age} : 33, \text{birthplace} : \text{“Krypton”}\right\}, \\ \{\text{name} : \text{“Batman”}, \text{age} : 30, \text{birthplace} : \text{“Earth”}\}, \\ \{\text{name} : \text{“Flash”}, \text{age} : 53, \text{birthplace} : \text{“Earth”}\}, \\ \{\text{name} : \text{“Wonder Woman”}, \text{age} : 28, \text{birthplace} : \text{“Earth”}\}, \\ \left\{\text{name} : \text{“General Zod”}, \text{age} : 42, \text{birthplace} : \text{“Krypton”}\right\} \end{array} \right\}$$

Relations are typically represented as tables.

name	age	birthplace
Superman	33	Krypton
Batman	30	Earth
Flash	53	Earth
Wonder Woman	28	Earth
General Zod	42	Krypton

Table 2.1: Person table

Definition 3 (Keys). Keys are constraints imposed on relations. A key constraint K on a relation r is a subset of $\text{ATTR}[r]$ which may uniquely identify a tuple. Formally, we say r satisfies the key constraint K, denoted as $r \vDash K$, subject to

$$\forall t, t' \in r, t \neq t' \implies t[K] \neq t'[K]$$

For example, in Table 2.1, the relation satisfies the key constraint {name}, but not {age}.

Definition 4 (Foreign Keys). A foreign key constraint applies to two relations, r_1, r_2. It asserts that values of certain attributes of r_1 must appear as values of some corresponding attributes of r_2. A foreign key constraint is written as

$$\theta = r_1(a_1, a_2, \ldots, a_k) \to r_2(b_1, b_2, \ldots, b_k)$$

where $a_i \subseteq \text{ATTR}[r_1]$ and $b_i \subseteq \text{ATTR}[r_2]$. We say (r_1, r_2) satisfies θ, denoted as $(r_1, r_2) \models \theta$, if

$$\forall t \in r_1, \exists t' \in r_2 \mid t[a_1, a_2, \ldots, a_k] = t'[b_1, b_2, \ldots, b_k]$$

Example 1. Suppose we have a relation Superhero(name, superpower). We can impose a FK constraint of

$$\text{Superhero(name)} \to \text{Person(name)}$$

Definition 5 (Relational Database). A relational database, d, is a named collection of relations (as defined by Definition 2, keys (as defined by Definition 3), and foreign key constraints (as defined by Definition 4.

We use $\textsc{Name}[d]$ to denote the name of d, $\textsc{Rel}[d]$ the list of relations in d, $\textsc{Key}[d]$ the list of key constraints of d, and $\text{FK}[d]$ the list of foreign key constraints of d.

2.1.1 Schema Group

Definition 6 (Schema Graph). If we view relations as vertices, and foreign key constraints as edges, a database d can be viewed as a *schema graph* G, formally defined as

$$\begin{aligned} \text{vertices} : V(G) &= \text{REL}[d] \\ \text{edges} : E(G) &= \text{FK}[d] \end{aligned}$$

Example 2. Given the following schema

Superhero(name, power)

Person(name, age, birthplace)

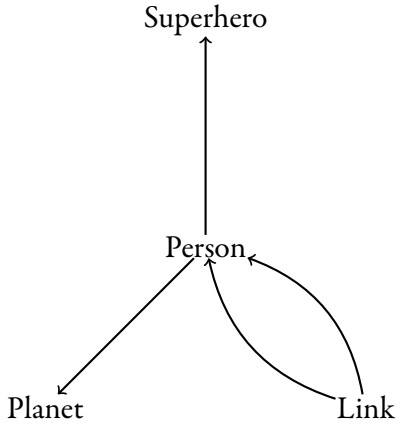
Planet(name, size, age, destroyed, galaxy)

Link(name, peer, relation type)

and the following foreign key constraints

$$\begin{aligned} \text{Superhero(name)} &\rightarrow \text{Person(name)} \\ \text{Person(birthplace)} &\rightarrow \text{Planet(name)} \\ \text{Link(name)} &\rightarrow \text{Person(name)} \\ \text{Link(peer)} &\rightarrow \text{Person(name)} \end{aligned}$$

we produce the following schema graph



The relational data model is particularly powerful for analytic queries. Given the schema below above, one can formulate the following analytic queries in a query language known as SQL.

```
SELECT Person.name
FROM   Person
       JOIN Planet
         ON Person.birthplace = Planet.name
WHERE  NOT Planet.destroyed;
```

Figure 2.1: Query to find superheroes with a home to go back to

Example 3. List all superheroes whose home planet has not been destroyed.

Results in the following output, given the data from Table 2.1,

name
Batman
Flash
Wonder Woman

2.1.2 Entity Group

Definition 7 (Entity Group)**.** An entity group is a forest, T, of tuples interconnected by join conditions defined by the foreign key constraints in the schema graph. Given two vertices $t_1, t_2' \in V(T)$, it must be that:

$\exists r_1, r_2 \in \text{REL}[d]$ such that $t_1 \in r_1, t_2 \in r_2$, and $(r_1, r_2) \in G$. This is to say that t_1 and t_2 belong to two relations that are connected by the schema graph.

Let $r_1(a_1, \ldots, a_k) \rightarrow r_2(b_1, \ldots, b_k)$ be the FK that connects r_1, r_2. We further assert that $t_1[a_1, \ldots, a_k] = t_2[b_1, \ldots, b_k]$.

The motivation of entity groups is to define complex structured objects that can include more information than individual tuples in the relations.

Example 4. The information in Table 2.2 all relates to Superman, however no single tuple in the database has all of this information as a result of database normalization.

Attribute	Value
name	Superman
age	33
birthplace	Krypton
superpower	flying, strength, speed
friends	Wonder Woman, Batman
enemies	General Zod

Table 2.2: Superman's properties

We require an entity group (Figure 2.2) to join together all pieces of information related to Superman.

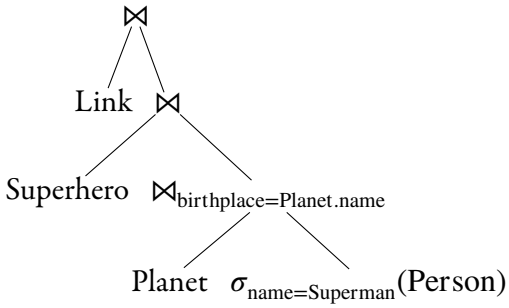


Figure 2.2: Superhero entity group

2.1.3 Pros and Cons of the Relational Model

In order to better understand the motivation behind this work, it is important to examine both the strong and weak points of the relational model.

Pros

The relational model was first proposed by Edgar F. Codd in 1969 while working at IBM [Cod69].

The enforcement of constraints is essential to the relational model. There are several types of constraints. The first constraint maintains uniqueness.

The Person relation has the attribute name as its primary key. In order for other relations to reference a specific named tuple, the name attribute must be unique.

Example 5 (Unique Constraint). Attempt to insert another person named "Superman."

```
INSERT INTO Person
VALUES      ('Superman',
             35,
             'Earth');
```

The relational database management system (RDBMS) enforces the primary key constraint on the `name` attribute, rejecting the insertion.

```
ERROR:  duplicate key value violates unique constraint "person_pkey"
DETAIL:  Key (name)=(Superman) already exists.
```

With the uniqueness of named tuples guaranteed (as demonstrated in Example 5), we must ensure that any named tuples that are referenced actually exist. If they do not, the database must not permit the operation to continue. Doing so would lead to dangling references.

Example 6 (Referential Integrity). Attempt to insert the superhero "Aquaman" with the superpower "telepathy."

```
INSERT INTO Superhero
VALUES      ('Aquaman',
             'telepathy');
```

Again we see the RDBMS protecting the integrity of the data.

```
ERROR:  insert or update on table "superhero" violates foreign
key constraint "superhero_name_fkey"
DETAIL:  Key (name)=(Aquaman) is not present in table "person".
```

In addition to enforcing consistency, the relational model is capable of providing higher-level views of the data through aggregation.

Example 7 (Aggregation (Simple))**.** Find the number of friends Superman has.

```
SELECT COUNT(*)
FROM   Link
WHERE  name = 'Superman'
       AND relationtype = 'friend';
```

Example 8 (Aggregation (Group By))**.** List the number of enemies of each Person.

```
SELECT name,
       Count(name)
FROM   Link
WHERE  relationtype = 'foe'
GROUP  BY name;
```

The above query produces the following output.

name	count
Superman	1
General Zod	1

Information stored within a properly designed database is normalized. That is, no information is repeated.

Example 9 (Normalization)**.** For example, suppose the planet Krypton is discovered to have been in the "Xeno" Galaxy rather than the "Andromeda" Galaxy. If this information were not normalized, each person whose birthplace was Krypton would need to be updated. Since this information is normalized, the following query will suffice.

```
UPDATE planet
SET    galaxy = 'Xeno'
WHERE  name = 'Krypton';
```

The above examples are some of the most important reasons for choosing the relation model over others. Unfortunately the relational model is not without its downsides.

Cons

While the relational model excels at ensuring data consistency, aggregation, and reporting, it is not suitable for every task. In order to issue queries, a user must be familiar with the schema. This requires specific domain knowledge of the data.

Example 10. Find all enemies of superheroes from Earth.

The above query requires the use of two joins.

```
SELECT peer
FROM   Link
       JOIN Person
         ON Person.name = Link.name
       JOIN Planet
         ON Planet.name = Person.birthplace
WHERE  relationtype = 'foe'
       AND Planet.name = 'Earth';
```

A casual user is unlikely to determine the correct join path, name of the tables, name of the attributes, etc. This is in contrast to the document model, where the data is semi-structure or unstructured, requiring minimal domain knowledge.

The relational model is also rigid in structure. If a relation is modified, every query referencing said relation may require a rewrite. Even a simple attribute being renamed (e.g. $\rho_{\text{name/alias}}(\text{Person})$) is capable of modifying the join paths. This rigidity places additional cognitive burden on users.

In addition to having a rigid structure, most relational database management systems lack flexible string matching options. Assuming basic SQL-92 compliance, a RDBMS only supports the `LIKE` predicate [ISO11].

Example 11 (`LIKE` Predicate)**.** Find all people with a name ending in "man."

```
SELECT *
FROM   Person
WHERE  name LIKE '%man';
```

There are a couple of limitations to the `LIKE` predicate. First, it only supports basic substring matching. If a user accidentally searches for all people with a name ending in "men," nothing would be found.

Second, unless the column used in the predicate is indexed, performance may be poor ($\mathcal{O}(n)$).

2.2 Document Model

In this section we formally define the document model.

Documents are a unit of information. The definition of unit can vary. It may represent an email, a book chapter, a memo, etc. Contained within each document is a set of terms.

In contrast to the relational model, the document model represents semi-structured as well as unstructured data. Examples of information suitable to the document model includes emails, memos, book chapters, etc.

These pieces, or units, of information are broken into documents. Groups of related documents (for example, a library catalogue) are referred to as a document collection.

Definition 8 (Terms and Document)**.** A term, t, is an indivisible string (e.g. a proper noun, word, or a phrase). A document, d, is a bag of words. Let $\mathrm{freq}_{t,d}$ be the frequency of terms t in document d.

Let T denote all possible terms, and $\textsc{Bag}[T]$ be all possible bag of terms.

Remark 1. We use the bag-of-words model for documents. This means that position information of terms in a document is irrelevant, but the frequency of terms are kept in the document. Documents are non-distinct sets.

Definition 9 (Document Collection)**.** A document collection D is a set of documents, written $D = \{d_1, d_2, \ldots, d_k\}$. The size of D is denoted N. The number of unique terms, or size of T, in D, is denoted M.

Example 12. Consider the following short sentences.

1. Superman is strong on Earth and lives on Earth.
2. Batman was born on Earth.
3. Superwoman is fast on Earth.
4. Superman was born on Krypton.

Each sentence represents a document, giving us the following documents.

$$
\begin{aligned}
d_1 &= \{\text{“and”} : 1, \text{“on”} : 2, \text{“is”} : 1, \text{“lives”} : 1, \text{“earth”} : 2, \text{“strong”} : 1, \text{“superman”} : 1\} \\
d_2 &= \{\text{“batman”} : 1, \text{“on”} : 1, \text{“was”} : 1, \text{“earth”} : 1, \text{“born”} : 1\} \\
d_3 &= \{\text{“on”} : 1, \text{“is”} : 1, \text{“superwoman”} : 1, \text{“fast”} : 1, \text{“earth”} : 1\} \\
d_4 &= \{\text{“krypton”} : 1, \text{“born”} : 1, \text{“on”} : 1, \text{“was”} : 1, \text{“superman”} : 1\}
\end{aligned}
$$

2.2.1 Vectorization of Documents

One of the most fundamental approach for search documents is to treat documents as high dimensional vectors, and the document collection as a subset in a vector space. The search query becomes a nearest neighbour query in a vector space equipped with a distance measure.

The first step is to convert bag of terms into vectors. The standard technique [MRS08] uses a scoring function that measures the relative importance terms in documents.

Definition 10 (TF-IDF Score). The term frequency is the number of times a term t appears in a document d, as given by $\text{freq}_{t,d}$. The document frequency of a term t, denoted by df_t, is the number of documents in D that contains t. It is defined as

$$\text{df}_t = \mid \{d \in D : t \in d\} \mid$$

The combined TF-IDF score of t in a document d is given by

$$\text{tf-idf}_{D,t,d} = \frac{\text{freq}_{t,d}}{\mid d \mid} \cdot \log \frac{N}{\text{df}_t}$$

Remark 2. The first component, $\frac{\text{freq}_{t,d}}{|d|}$, measures the importance of a term within a document. It is normalized to account for document length. The second component, $\log \frac{N}{\text{df}_t}$, is a measure of the rarity of the term within the document collection D.

Example 13. Using the documents from Example 12, the TF-IDF scores are as follows.

	d_1	d_2	d_3	d_4
"and"	0.2857	0.0000	0.0000	0.0000
"on"	0.0000	0.0000	0.0000	0.0000
"superwoman"	0.0000	0.0000	0.4000	0.0000
"batman"	0.0000	0.4000	0.0000	0.0000
"is"	0.1429	0.0000	0.2000	0.0000
"fast"	0.0000	0.0000	0.4000	0.0000
"born"	0.0000	0.2000	0.0000	0.2000
"krypton"	0.0000	0.0000	0.0000	0.4000
"earth"	0.1186	0.0830	0.0830	0.0000
"lives"	0.2857	0.0000	0.0000	0.0000
"strong"	0.2857	0.0000	0.0000	0.0000
"was"	0.0000	0.2000	0.0000	0.2000
"superman"	0.1429	0.0000	0.0000	0.2000

Definition 11 (Document Vector)**.** Given a document collection $\boldsymbol{D}$ with $\boldsymbol{M}$ unique terms $\boldsymbol{T} = [t_1, t_2, \ldots, t_n]$, each document d can be represented by an $\boldsymbol{M}$-dimensional vector.

$$\vec{d} = \begin{bmatrix} \text{tf-idf}_{t_1,d} \\ \text{tf-idf}_{t_2,d} \\ \vdots \\ \text{tf-idf}_{t_n,d} \end{bmatrix}$$

Example 14. The documents in Example 12 would produce the following vectors.

$$\vec{d}_1 = \begin{bmatrix} 0.2857 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.1429 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.1186 \\ 0.2857 \\ 0.2857 \\ 0.0000 \\ 0.1429 \end{bmatrix}, \vec{d}_2 = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.4000 \\ 0.0000 \\ 0.0000 \\ 0.2000 \\ 0.0000 \\ 0.0830 \\ 0.0000 \\ 0.0000 \\ 0.2000 \\ 0.0000 \end{bmatrix}, \vec{d}_3 = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.4000 \\ 0.0000 \\ 0.2000 \\ 0.4000 \\ 0.0000 \\ 0.0000 \\ 0.0830 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}, \vec{d}_4 = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.2000 \\ 0.4000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.2000 \\ 0.2000 \end{bmatrix}$$

Definition 12 (Search Query)**.** A search query q is simply a document, namely a bag of terms. The top-k answers to q with respect to a collection $\boldsymbol{D}$ is defined as the k documents, $\{d_1, d_2, \ldots, d_k\}$, in $\boldsymbol{D}$, such that $\left\{\vec{d}_i\right\}$ are the closest vectors to $\vec{q}$ using Euclidean distance measure in $\mathbb{R}^N$.

Example 15. Given the search query $q = \{\text{superwoman}, \text{was}, \text{born}, \text{on}, \text{krypton}\}$, compute the vector $\vec{q}$ within the document collection $\boldsymbol{D}$ (as defined in Example 12).

$$\vec{q} = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.1474 \\ 0.2644 \\ 0.0000 \\ 0.2644 \\ 0.1474 \\ 0.0000 \end{bmatrix}$$

In order to determine the top-k documents for search query q, we need a way of measuring the similarity between documents.

Definition 13 (Cosine Similarity). Given two document vectors, $\vec{d}_1$ and $\vec{d}_2$, the cosine similarity is the dot product $\vec{d}_1 \cdot \vec{d}_2$, normalized by the product of the Euclidean distance of $\vec{d}_1$ and $\vec{d}_2$ in $\mathbb{R}^N$. It is denoted as $\text{similarity}_{\vec{d}_1,\vec{d}_2}$.

$$\text{similarity}_{\vec{d}_1,\vec{d}_2} = \frac{\vec{d}_1 \cdot \vec{d}_2}{|| \vec{d}_1 || \cdot || \vec{d}_2 ||} \tag{2.1}$$

$$= \frac{\sum_{i=1}^{N} \vec{d}_{1,i} \times \vec{d}_{2,i}}{\sqrt{\sum_{i=1}^{N} \left(\vec{d}_{1,i}\right)^2} \times \sqrt{\sum_{i=1}^{N} \left(\vec{d}_{2,i}\right)^2}} \tag{2.2}$$

Recall we may represent search queries as documents and thus document vectors. Therefore we may compute the score of a document d for a search query q as

$$\text{similarity}_{\vec{d},\vec{q}}$$

Example 16. Given the document collection D (from Example 12) and search query q, compute the similarity between q and every document $d \in D$.

$$\text{similarity}_{\vec{d}_1,\vec{q}} = 0.000000 \tag{2.3}$$

$$\text{similarity}_{\vec{d}_2,\vec{q}} = 0.191533 \tag{2.4}$$

$$\text{similarity}_{\vec{d}_3,\vec{q}} = 0.265877 \tag{2.5}$$

$$\text{similarity}_{\vec{d}_4,\vec{q}} = 0.618553 \tag{2.6}$$

2.2.2 Extending the Document Model

In the extended document model, documents have attributes: ATTR[d], and each attribute have values (e.g. date, string, integer), or bag of terms. Thus:

$$d : \text{ATTR}[d] \rightarrow \text{BAG}[\text{Terms}]$$

Example 17 (Semi-Structured Document). We see that d_2 is about Batman. The document contents are semi-structured, containing both a name and the name of a planet. By adding attributes to the document, we are left with Table 2.3.

j

Attribute	Value
name	Batman
birthplace	Earth
body	Batman was born on Earth.

Table 2.3: Person document for Batman

which is similar in structure to the `Person` table.

2.2.3 Pros and Cons of the Document Model

- Good: exploratory queries using keywords (google)
- Good: easy (or no) syntax
- Good: fuzzy matching (using n-gram)
- Bad: No analytics

2.3 Best of both worlds (4 days, week 3)

- Hybrid database defined by both the relational model and the document model
- Translation between relational objects (entities and entity) groups to documents.
- Translation of documents back to relational objects.
- Proof of lossless translation between relational space and document space

Chapter 3

Along came Clojure

3.1 Basic principles of functional programming (2 days)

- immutable data structures
- persistent data structures using multi-versioning
- functions (and higher order functions) as values

3.2 Features of Clojure (2 days)

- Data structures supporting the universal design pattern
- Concurrency + STM
- Interoperability with JVM (including Lucene)

Chapter 4

Search w/ Clojure

4.1 Thirdparty libraries (1 day, week 4)

- Lucene

4.2 Indexing of relational objects (5 days, week 5)

- Schema definition
- Crawling using SQL
- Indexing using relational objects
- Fuzzy indexing of values (typed by classes)

4.3 Keyword Search in document space (5 days, week 6)

- Disambiguate keywords using fuzzy search (suggestion, overloaded terms)
- Flexibility keyword search for documents
- Translate search result back to relational space

4.4 Graph Search in document space (5 days, week 7)

- Why we need graph search
- Search in document graph using graph search algorithms with functional implementations: (Ford Fulkerson, BFS)
- Speed up using concurrency
- Clojure specific optimization: ref + atom

Chapter 5

Experimental evaluation (5 days, week 8)

5.1 Implementation

- Choice of language
- Statistics about the code base: LOC, classes, ?
- Github hosted

5.2 The data set

- Description of the data set
- Statistics of the data set

5.3 Runtime Evaluation

- Index speed
- Keyword search speed
- Graph search speed:

  - Ford Fulkerson
  - BFS
  - Concurrent BFS using refs
  - Concurrent BFS using atoms

5.4 Lessons learned

- Simple algorithms are easier to parallelize
- STM is effective: transactions do not rollback (that much), so we observe impressive speed-up in concurrent versions.
- Fine tuning is beneficial: atom is better than ref.
- The clojure way: correctness first, runtime optimization latter (ref to atom is natural).

Chapter 6

Conclusion (0 days)

Survived Clojure.

Bibliography

[Cod69] E. F. Codd. Derivability, redundancy and consistency of relations stored in large data banks. *Ibm research report, san jose, california*, RJ599, 1969.

[GUW09] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.

[ISO11] ISO. Information technology – database languages – sql – part 2: foundation (sql/foundation). ISO (ISO/IEC 9075-2:2011). Geneva, Switzerland: International Organization for Standardization, 2011.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN: 0521865719, 9780521865715.