TOWARDS COMPACT OPERATOR TREES FOR QUERY INDEXING

Hagen Höpfner
School of Information Technology, International University in Germany, Campus 3, 76646 Bruchsal, Germany

Erik Buchmann
Institute for Program Structures and Data Organization, Universität Karlsruhe (TH), Germany

Keywords: Databases, Query Indexes, Query Semantics.

Abstract: Application areas like semantic caches or update relevancy checks require query based indexing: They use an algebra representation of the query tree to identify reusable fragments of former query results. This requires compact query representations, where semantically equivalent (sub-)queries are expressed with identical terms. It is challenging to obtain such query representations: Attributes and relations can be renamed, there are numerous ways to formulate equivalent selection predicates, and query languages like SQL allow a wide range of alternatives for joins and nested queries. In this paper we present our first steps towards optimizing SQL-based query trees for indexing. In particular, we use both existing equivalence rules and new transformations to normalize the sub-tree structure of query trees. We optimize selection and join predicates, and we present an approach to obtain generic names for attributes and table aliases. Finally, we discuss the benefits and limitations of our intermediate results and give directions for future research.

1 INTRODUCTION

Nowadays ubiquitous, nomadic, and pervasive computing are not longer visions but realized in various scenarios. Devices become smaller and easier to carry around and wireless links connect to world wide available information anytime and everywhere. Due to slow, unreliable and/or energy-intensive wireless networks, efficient strategies to retrieve and cache data from central servers on mobile devices are key for almost all mobile applications. Related techniques include hoarding (Kuenning and Popek, 1997), replication (Gray et al., 1996) or semantic caching (Lee et al., 1999). For example, semantic caching materializes query results at the mobile devices and reuse them for future queries, i.e., the queries are used for indexing the cached data. Therefore, it is necessary to find out if (sub-)queries overlap. Figure 1 illustrates this idea. The results of three queries X_1, X_2, and X_3 were cached and can be reused for answering the new queries X_4, X_5, X_6, and X_7. Queries are represented in form of conjunctively linked operator trees. Hence, the re-usability of the cached data can be analyzed by traversing the tree (illustrated by dashed and dotted lines in Figure 1). More details can be seen in

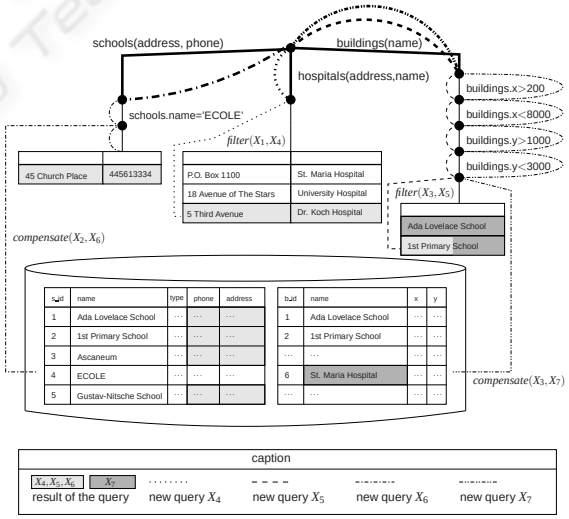

Figure 1: Query index of a semantic cache (Höpfner and Sattler, 2003).

(Höpfner and Sattler, 2003).

It is challenging to realize query indexing on the basis of current SQL-based DBMS. SQL allows to formulate semantically equivalent queries in syntactically different ways. Even the straightforward Ex-

Höpfner H. and Buchmann E. (2008).
TOWARDS COMPACT OPERATOR TREES FOR QUERY INDEXING.
In *Proceedings of the Third International Conference on Software and Data Technologies - ISDM/ABF*, pages 174-179
DOI: 10.5220/0001888001740179

ample 1 might result in two different representations for the equivalent queries Q_1 and Q_2. Furthermore, current DBMS transform queries solely according to the execution time. In contrast, semantic caches require

- minimal query representations that do not exhaust the resources of mobile devices,
- identical representations of semantically equivalent sub-trees in the query tree, and
- representations without contradictory renaming operations that complicate the detection of semantically overlapping (sub-)queries.

Example 1: *Semantically but not syntactically equivalent*

Q_1 :
```
SELECT * FROM TABLE1 AS T1
  WHERE B=5 AND A=4
```
Q_2 :
```
SELECT * FROM (SELECT * FROM TABLE1
  WHERE A=4 and B=5) AS T1
```

In this paper, we describe our first steps towards transforming SQL into a compact query representation that is optimized for indexing purposes. Therefore, we use existing equivalence transformations, and we devise new transformations that adapt syntactical properties of queries. This comprises:

1. We introduce rules to reduce the complexity of query trees. Our rules utilize overwriting effects and interdependencies between operators to remove or merge certain operations.
2. We rewrite logical expressions to represent semantically equivalent terms as similar as possible while considering existing decision problems.
3. We normalize the aliases in the query tree to obtain identical representations for (sub-)queries that are semantically equivalent but contain different renaming operations.

Paper Structure. Section 2 defines query trees and specifies the set of operators that we support at the moment. Section 3 discusses our approach to unify the query representation, and Section 4 concludes.

2 QUERIES AND QUERY TREES

In this paper we focus on transforming SQL queries for indexing purposes. As a starting point we assume that the the SQL query has been translated into a canonical query tree of relational algebra operators. We focus on a relational complete set of algebra operators (Elmasri and Navathe, 2007) consisting of selection σ, projection π, set union $\cup$, set difference $-$, and Cartesian product $\times$. In addition, we consider renaming of attributes, renaming of relations ρ and set intersection $\cap$. Operators like aggregations or groupings will be part of our future work.

In order to meet the SQL semantics, we distinguish between set and multi-set operators. It is well-known that multi-set relational algebras have nearly the same properties as set based approaches. Thus, we briefly introduce our notation and refer to literature (Dayal et al., 1982; Grefen and de By, 1994) for further reading on relational algebra semantics. A multi-set projection (`SELECT`) is denoted with an enhanced relational projection operator π^a, while the projection operator π corresponds to the set semantics (`SELECT DISTINCT`). We ease our presentation by supposing that each query contains one projection operator π or π^a. We implicitly assume multi-set semantics for the Cartesian product, the set operators and the selection operator. If necessary, duplicate elimination can be done by the projection operator π. Assuming that the "*" means all attributes of relations used by a query, translating the two SQL queries from Example 1 could result in the relation algebra expression $Q_1 = \pi^a_*(\rho_{\mathsf{T1}}(\sigma_{B=5 \wedge A=4}(\mathsf{TABLE1})))$ and $Q_2 = \pi^a_*(\rho_{\mathsf{T1}}(\pi^a_*(\sigma_{A=4 \wedge B=5}(\mathsf{TABLE1}))))$. Formally, a query q can have the following recursive structure:

$$
\begin{aligned}
q &: \{\pi|\pi^a\}([\sigma]([\rho](R))) \\
q &: \{\pi|\pi^a\}([\sigma](\rho(q))) \\
q &: \{\pi|\pi^a\}([\sigma](cp)) \\
cp &: \{[\rho](R)|\rho(q)\} \times \{[\rho](S)|\rho(q)|cp\} \\
q &: \{\pi|\pi^a\}(q\{\cup|-|\cap\}q)
\end{aligned}
$$

Operators in square brackets encapsulate optional operators. R and S are relations, and braces mark alternatives. This structure spans a query tree where leafs represent the relations and inner nodes store query operators. The query tree is processed by starting on the leaves and following the edges of the tree. After performing the operation in the root, the query is completely answered. Sub-queries are boxed, i.e., they form self-contained queries that are sub-trees of the query tree.

Example 2:

Q_3 :
```
SELECT DISTINCT B FROM TABLE1, TABLE2
  WHERE TABLE2.B>TABLE1.A
```

Figure 2 shows the query trees for the queries Q_1, Q_2 and Q_3, which will be used as examples throughout the paper. Operators of the sub-query in Q_2 are highlighted by a gray box. The algebraic expression of Q_3 is $\pi^a_B(\sigma_{\mathsf{TABLE2}.B>\mathsf{TABLE1}.A}(\mathsf{TABLE1} \times \mathsf{TABLE2}))$.

We start our transformations with the canonical

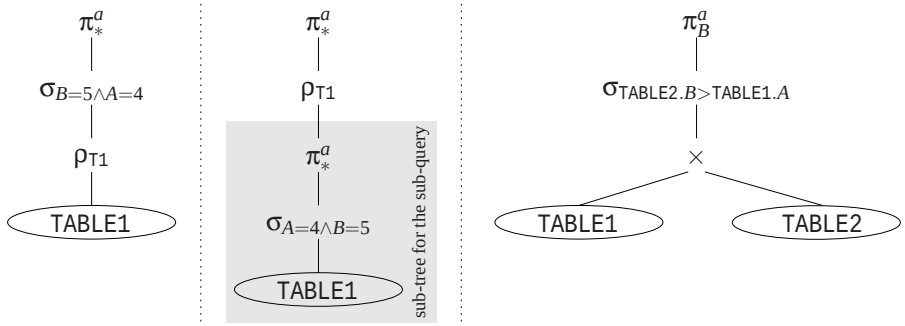

Figure 2: Query trees for the example queries (Q_1 left, Q_2 middle, Q_3 right).

query tree (Elmasri and Navathe, 2007). Thus, the query tree is normalized so that a projection constitutes the root of each (sub-)query, followed by an optional selection, an optional renaming operation, and either a Cartesian product, a relation name or another boxed sub-query. Note that sub-queries in SQL must have an unique name, i.e., if a query contains a sub-query then the renaming operator of the including (outer) query becomes mandatory.

3 COMPLEXITY REDUCING QUERY REWRITING

In this section we describe our approach towards less complex query representations for query indexing. We introduce three steps to rewrite the canonical query tree: (1) The removal of unnecessary sub-trees, (2) the equivalence transformation of predicate expressions and (3) a normalization of renaming operations. These steps can be implemented in the semantic cache on the mobile client, on a mobility supporting middle-ware (Höpfner, 2007), or in the back-end.

3.1 Sub-Tree Optimization

As Figure 2 shows, queries with a different sub-tree structure can have the same semantics. In order to assign equivalent queries with the same entry in a semantic cache, the sub-tree structure has to be normalized. Therefore, we reduce the number of unnecessary sub-trees in our first rewriting step. The resulting query is semantically equivalent to the initial one. Since this paper describes our ongoing work, there are still cases where two different but semantically equivalent queries result in different representation after normalization. Thus, there is high potential for future research. Different sub-trees can handle duplicates differently. For this reason, we have to consider duplicate elimination first.

π^a-π-Optimization. Observe that duplicate elimination at the (sub-)root node of a certain sub-tree implicitly holds for all of it's sub-trees, too. In other words, all sub-queries of a `SELECT DISTINCT`-query can be written as `SELECT DISTINCT`-sub-queries. The proof (Gupta et al., 1995) of this observation is that `SELECT DISTINCT` D `FROM` $R \equiv$ `SELECT` D `FROM` R `GROUP BY` D, while groupings can be pushed down and pulled up under certain conditions. Hence, we traverse the query tree and replace all duplicate preserving projections π^a below a duplicate eliminating π-operator by π. Note that this paper leaves aside aggregations. Since the correctness of some aggregates depend on duplicates, such operations constitute exceptions for this step.

Sub-Queries without Set Operations and Cartesian Products. Nested sub-queries of the form $Q: Q_{out}(Q_{in_1}(Q_{in_2}\ldots(Q_{inmost})))$ can be reduced if the inner queries contain *neither* set-operations *nor* Cartesian products. Following the formal query representation in Section 2, the inmost sub-query Q_{inmost} must have the following structure:

$$Q_{inmost}: \{\pi|\pi^a\}([\sigma]([\rho](R))).$$

The handling of duplicates in nested sub-queries without Cartesian products is comparable to our π^a-π-optimization: the projections in all sub-queries Q_* can be unified to π if at least one sub-query removes duplicates. Otherwise, all projections in Q_* are π^a. For simplification purposes we consider π only. The same transformations can be applied for π^a.

Remember that SQL requires the renaming of sub-queries. As shown in Section 2 the inmost operator of the outer query has to be a ρ. In our canonical query tree structure, ρ is the parent of the projection node of an inner sub-query. Our sub-tree optimization starts at the inmost nested sub-query and is used bottom-up until a set operation, a Cartesian product or the root-node is reached. We combine a boxed sub-query with its nesting query by merging the projections and se-

lection predicates, and by unifying all relation names and attribute names which have been renamed in the nesting query. Therefore, we distinguish two cases:

Case 1. Outer (sub-)query uses projection and renaming:

$$\underbrace{\pi_{pl_o}(\rho_{al_o}}_{\text{outer sub-query}} \ (Q_{inmost}))$$

π_{pl_o} overwrites the projection of the inner query *but* is based on the renaming ρ_{al_o}. Furthermore, ρ_{al_o} overwrites a potential renaming within Q_{inmost}. So, both queries are merged as follows:

1. Remove the projection of the inner query if it it covers more attributes than the projection of the outer query. Otherwise, replace the outer projection by the inner one.
2. If the inner query does neither contain a renaming nor a selection, both queries have been merged: Stop rewriting.
3. If the inner query contains a selection σ_{sc_i}, change the order of σ_{sc_i} and ρ_{al_o}.

   If the inner query contains a renaming $\rho_{al_i}(R)$, merge the set of aliases in the inner ($al_i = RA_i(AA_i)$) and outer ($al_o = RA_o(AA_o)$) query, i.e.,

   (a) Merge and reduce the attribute-name-aliases set: $AA_m = reduce(AA_i \cup AA_o)$.

   (b) For all relations in nested renaming sequences $R \rightarrow RA_i \rightarrow RA_o$: Replace $\rho_{al_i}(R)$ by $\rho_{al_m}(R)$ where $al_m = RA_o(AA_m)$.

   (c) For all inner attribute aliases $al^i \leftarrow a$ with $a \in R$ and $al^i \in AA_i$ that are overwritten by the outer renaming $al^o \leftarrow al^i$, $al^o \in AA_o$: Replace $al^i \leftarrow a$ by $al^o \leftarrow a$.

   (d) Remove $al^o \leftarrow al^i$ from $AA_i \cup AA_o$.

   (e) Replace the outer renaming ρ_{al_o} by the merged renaming ρ_{al_m}, and remove ρ_{al_i}.
4. If the inner query contains a selection σ_{sc_i}, adapt the selection conditions sc_i to the new aliases, i.e.,

   (a) Replace all relation names of R and RA_i in sc_i by RA_o.

   (b) Replace an attribute name a in sc_i by al with $al \leftarrow a \in AA_m$, if it has been renamed by $\rho_{RA_m(AA_m)}$.

Note that Step 3 is not an equivalence transformation, because ρ_{al_o} might rename attributes or relations needed for σ_{sc_i}. Step 4 corrects this. Method *reduce* replaces transitive aliases similar to the table aliases.

Case 2. Outer (sub-)query uses projection, selection and renaming:

$$\underbrace{\pi_{pl_o}(\sigma_{sc_o}(\rho_{al_o}}_{\text{outer sub-query}}(Q_{inmost})))$$

The difference to Case 1 is that the outer selection σ_{sc_o} and the inner selection σ_{sc_i} have to be merged after the other steps. Therefore, σ_{sc_i} is removed from from the sub-tree and and σ_{sc_o} is replaced by a new selection σ_{sc_m} where $sc_m = sc_o \wedge sc_i$ holds.

Example 3: *Sub-query optimization*
The initial tree representation of Q_2 is

$$\pi^a_* - \rho_{T1} - \boxed{\pi^a_* - \sigma_{A=4 \wedge B=5} - \text{TABLE1}}$$

In the first step the outer projection is replaced by the inner one. The resulting tree is:

$$\pi^a_* - \rho_{T1} - \boxed{\sigma_{A=4 \wedge B=5} - \text{TABLE1}}$$

The remaining inner sub-query contains a selection. Therefore, we have to continue with step 3. The result of step 3 is:

$$\pi^a_* - \boxed{\sigma_{A=4 \wedge B=5} - \rho_{T1} - \text{TABLE1}}$$

Since the inner sub-query does not contain a renaming, ρ_{T1} becomes the final renaming operator. $\sigma_{A=4 \wedge B=5}$ does not use the renamed relation name and the attributes were not renamed. Therefore, the select condition does not have to be changed.

3.2 Where-Condition Optimization

Our second optimization targets at selection and join predicates. As semantically equivalent predicates can be expressed in numerous ways, the usage of query representations for indexing calls for normalization of predicates. The equivalency problem of logical expressions is undecidable in the general case. However, when considering SQL, where-conditions are first-order predicate logic expressions which do not contain quantifiers and correspond to propositional logic expressions. Thus, we can conduct a large number of equivalence transformations on such expressions. We optimize the predicates in two phases:

Phase 1: Lexicographical Sorting. One difference between two predicates can result from the order of conjuncts. For example, the where-conditions of $Q_1 : B = 5 \wedge A = 4$ and $Q_2 : A = 4 \wedge B = 5$ are equivalent. Thus, the first phase orders the conjuncts lexicographically. The following rules hold:

$$B < A \equiv A > B \qquad B > A \equiv A < B$$
$$B \leq A \equiv A \geq B \qquad B \geq A \equiv A \leq B$$
$$B = A \equiv A = B \qquad B \neq A \equiv A \neq B$$

In particular, we rewrite all attribute-attribute-comparisons (AAC) so that the first attribute is lexicographically smaller than the second one.

Example 4: *Ordering of conjuncts*
Given the AAC `TABLE2.B>TABLE1.A` of Q_3 we force a lexicographic order by exchanging the attributes and substituting the comparison operator. The result is `TABLE1.A<TABLE2.B`.

Phase 2: Term Minimization. In the second phase we analyze the attribute-value-comparisons (AVC). AVCs might form an unnecessarily complex propositional logic expression. There exist algorithms for transforming an arbitrary logical expression to a conjunctive normal form (Russell and Norvig, 2002) and to minimize this expression (Quine, 1952; Karnaugh, 1953; McCluskey, 1956; Biswas, 1984). However, all those algorithms find *one* minimal expression, but cannot exclude that there are other minimal expressions as well. Hence, it cannot be guaranteed that equivalent expressions are reduced to the same minimal expression. However, for the time being we take the algorithmically found term and order it lexicographically. More specifically, we represent each where-condition in a minimal conjunctive normal form where the atomic terms of each conjunct are ordered lexicographically. After ordering the conjuncts, we order the entire expression. The resulting example query trees are shown in Figure 3.

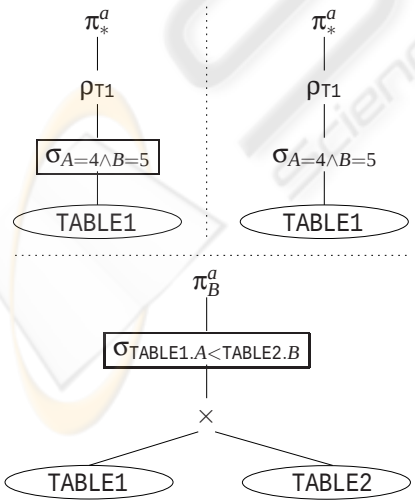

Figure 3: Query trees for the example queries (Q_1 top left, Q_2 top right, Q_3 bottom) after the where-condition optimization.

3.3 Harmonizing Aliases

SQL queries can specify aliases for tables, views, attributes etc. Since SQL allows many kinds of nested queries, the SQL parser automatically inserts renaming operations into query trees, too. However, renamed attributes or relations can result in different representations of semantically equivalent (sub-)queries. Thus, it is important to normalize names in query trees. Therefore, we replace aliases by generic names. This requires:

P1 The alias substitution must be deterministic.

P2 The aliases must not be identifiers in used in the base relations.

P3 The mapping must be isomorphic and complete.

When leaving aside SQL view definitions, the deepest alias in a query tree always renames the original relation; the second alias renames the first alias, etc. Thus, we can guarantee a deterministic substitution (**P1**) by using the md5-Algorithm (Rivest, 1992) on the renamed table/alias to compute generic names. A sequence of renaming operations *table*$\rightarrow$*alias*$_1\rightarrow\cdots\rightarrow$*alias*$_n$ is substituted by *table*$\rightarrow$*md5(table)*$\rightarrow\cdots\rightarrow$*md5(...md5(table))*. In order to avoid valid SQL table names (**P2**), we propose to use special prefix characters in aliases that are not allowed in SQL names, such as "@". An isomorphic and complete mapping (**P3**) can be ensured by traversing the whole query tree in a defined order. To distinguish base relations used more than once in a query (e.g. in case of self-joins), we maintain a counter table *CT* that stores whether and how often a base relation is used. The following algorithm does this for table aliases:

1. Traverse the query tree in a post-order manner.
2. For each renaming operation at each node,
   (a) if its a renaming of a base relation
      i. if base relations name is in the *CT* then increase and fetch its counter *c*
      ii. else insert the base relations name into *CT* and set its counter *c* to 1
   (b) substitute *source*$\rightarrow$*alias* by *source*$\rightarrow$*@_md5(source)_c*, and
   (c) replace all appearances of *alias* as table alias in all parent nodes by *@_md5(target)_c*.

The same algorithm can be applied for attribute aliases. However, attribute aliases and table aliases must not be mixed, e.g., by using different prefix characters. Table aliases can appear in other renaming operation or as prefix `TABLE_ALIAS.ATTRIBUTE` of attributes in selections and projections. Attribute aliases might also appear in further renaming operations or

as suffix of attribute names. Example 5 illustrates the harmonizing of aliases.

Example 5: *Alias harmonization*
Before performing the alias harmonization, the query trees of Q_2 and Q_4 are:

$$Q_2 : \pi^a_* - \sigma_{A=4 \wedge B=5} - \rho_{\texttt{T1}} - \texttt{TABLE1}$$
$$Q_4 : \pi^a_* - \sigma_{A=4 \wedge B=5} - \rho_{\texttt{T2}} - \texttt{TABLE1}$$

Both queries contain only one renaming operation each ($Q_2 : \texttt{TABLE1} \rightarrow \texttt{T1}$, $Q_2 : \texttt{TABLE1} \rightarrow \texttt{T2}$). They rename TABLE1 but do not use the alias in any other operation. The md5-hash of TABLE1 is d20a1138c815109c831e910488ebf146. Hence, the modified trees are:

$$Q_2 : \pi^a_* - \sigma_{A=4 \wedge B=5} - \rho_{\texttt{@\_d20a...bf146\_1}} - \texttt{TABLE1}$$
$$Q_4 : \pi^a_* - \sigma_{A=4 \wedge B=5} - \rho_{\texttt{@\_d20a...bf146\_1}} - \texttt{TABLE1}$$

Q_2 and Q_4 are now syntactically equivalent.

4 SUMMARY AND OUTLOOK

Due to the increasing complexity of mobile applications, query indexing is an emerging topic in research and practice. In this paper we presented first ideas towards finding a compact representation of semantically equivalent database queries. Our approach reduces the syntactical complexity of database queries (1) by applying well known and proofed transformation rules, (2) by forcing an order within logical expressions, and (3) by normalizing the names of attributes and relations. All algorithms are of a deterministic nature, except the optimization of the where-condition. For this reason, we can not guarantee to transform all semantically equivalent queries to the same index entry. This might result in some duplicates in the query index, which might be acceptable for semantic caching and many other application areas. Thus, we can already declare success if we find rules that are applicable to the majority of real-world queries. As part of our ongoing research we plan to address further cases of semantic equivalence. Examples include self-joins like `SELECT * FROM T1 AS A, T2 AS B WHERE A.a = B.b` and `SELECT * FROM T1 AS B, T2 AS A WHERE A.a = B.b`. Both are equivalent but represented in a different way. A solution could be to order the From-list lexicographically. Furthermore, we intend to evaluate our approach by implementing a prototype that can be tested with well-known query mixes, e.g., the TCP-H benchmark.

REFERENCES

Biswas, N. N. (1984). Computer aided minimization procedure for boolean functions. In *Proceedings of the 21st conference on Design automation*.

Dayal, U., Goodman, N., and Katz, R. H. (1982). An extended relational algebra with control over duplicate elimination. In *Proceedings of the SIGACT-SIGMOD Symposium on Principles of Database Systems*.

Elmasri, R. and Navathe, S. B. (2007). *Fundamentals of Database Systems*. Addison Wesley.

Gray, J., Helland, P., O'Neil, P., and Shasha, D. (1996). The Dangers of Replication and a Solution. *SIGMOD Record*, 25(2):173–182.

Grefen, P. W. and de By, R. A. (1994). A multi-set extended relational algebra: a formal approach to apractical issue. In *Proceedings of the 10th ICDE*.

Gupta, A., Harinarayan, V., and Quass, D. (1995). Aggregate-Query Processing in Data Warehousing Environments. In *Proceedings of the 21st VLDB*.

Höpfner, H. (2007). Query Based Client Indexing in Client/Server Information Systems. *Journal of Computer Science*, 3(10):773–779.

Höpfner, H. and Sattler, K.-U. (2003). Towards Trie-Based Query Caching in Mobile DBS. In *Proceedings of the Workshop Scalability, Persistence, Transactions - Database Mechanisms for Mobile Applications*.

Karnaugh, M. (1953). The Map Method for Synthesis of Combinational Logic Circuits. *Transactions of American Institute of Electrical Engineers*, 72(7):593–599.

Kuenning, G. H. and Popek, G. J. (1997). Automated Hoarding for Mobile Computers. *ACM SIGOPS Operating Systems Review*, 31(5):264–275.

Lee, K. C. K., Leong, H. V., and Si, A. (1999). Semantic query caching in a mobile environment. *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(2):28–36.

McCluskey, E. J. (1956). Minimization of Boolean Functions. *Bell System Techical Journal*, 35(5):1417–1444.

Quine, W. V. O. (1952). The problem of Simplifying Truth functions. *American Mathematics Monthly*, 59(8):521–531.

Rivest, R. L. (1992). The MD5 Message-Digest Algorithm. Informational Errata. `http://tools.ietf.org/html/rfc1321`.

Russell, S. J. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*.