QSQL: Incorporating Logic-based Retrieval Conditions into SQL

Sebastian Lehrack and Ingo Schmitt

Brandenburg University of Technology Cottbus
Institute of Computer Science
Chair of Database and Information Systems

DOAG 2010 - Nuremberg
17/11/2010

Outline

- Motivation
- CQQL retrieval model
- Incorporating CQQL concepts into SQL
- Experiments and outlook

Motivation example

TV sets				
Name	Handling	Image quality	OSP	Sound quality
...	...	...	...	...

- Scenario: assessment of TV sets
- Properties `handling`, `image quality` and `sound quality` are rated by marks between 1 (best) and 6 (worst)

- Query: I want to find a device with a handling as easy as possible and the best possible quality of image. If a device has not a link to a sound system, the internal sound quality has to be as high as possible.

Motivation example

$$(ha \approx 1 \ \wedge \ iq \approx 1) \ \wedge \ (osp = no \Rightarrow sq \approx 1)$$

- Introducing similarity predicates for modeling vague conditions
  - handling as easy as possible
  - best possible quality of image
  - ...
- Similarity predicates are indicated by $\approx$ and are evaluated to a score value $\in [0, 1]$
- Score value expresses the degree of fulfilling
- Boolean evaluation with a threshold mark given by 2:

Motivation example

- Evaluation? $\rightsquigarrow$ classical approach Boolean logic
- Boolean evaluation needs a threshold mark for vague conditions given by 2:

TV sets				
Name	Handling	Image quality	OSP	Sound quality
TV1	2 (True)	2 (True)	yes (True)	3 (False)
TV2	2 (True)	1 (True)	yes (True)	1 (True)
TV3	1 (True)	2 (True)	no (False)	1 (True)
TV4	3 (False)	1 (True)	no (False)	4 (False)
...	...	...	...	...

- Result set includes TV1, TV2 and TV3
- TV2 would be the best one

Consequences

- Integrating impreciseness and proximity into a logic-based query language by introducing similarity predicates

- Combination of
  - Boolean predicates:
    `osp=no` $\rightsquigarrow$ truth value $\in \{\texttt{true}, \texttt{false}\}$ and
  - Similarity predicates:
    `handling` $\approx$ `1` $\rightsquigarrow$ score value $\in [0, 1]$

  within a logical query language

- Based on score values ranking becomes possible

Related approaches

- Boolean logic with threshold values [Oracle08]
- Fuzzy logic [Zadeh65]
  - Semantics are defined by aggregation functions without a semantical model
- Probabilistic Many-World-Semantics [MayBMS09, Trio09]
  - Probabilities for different possible worlds/states must be modeled in advanced
- Aggregation of probabilities without an inherent semantical model [Roelleke08]
  - Correct aggregation semantics have to be preserved by users

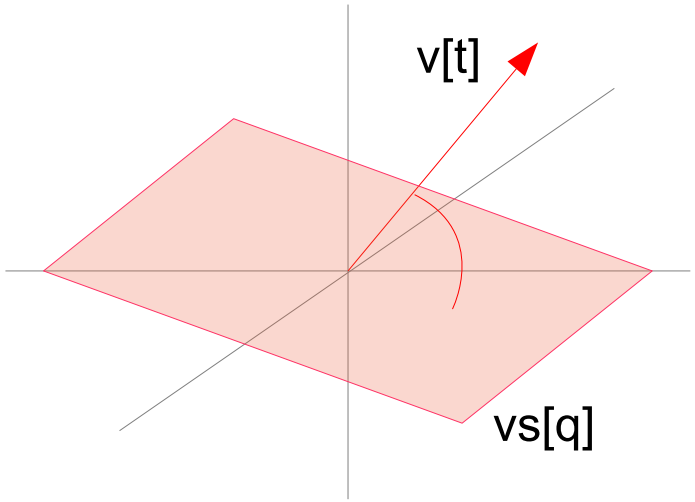
Outline

- Motivation
- CQQL retrieval model
- Incorporating CQQL concepts into SQL
- Experiments and outlook

Calculus query language: CQQL

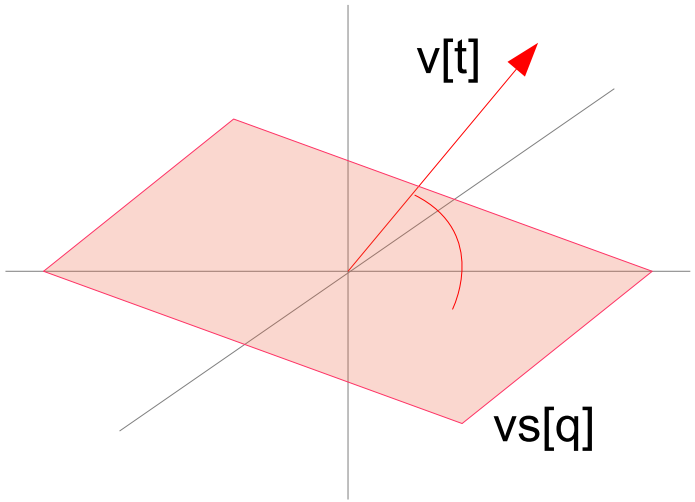
- CQQL: Commuting Quantum Query Language
- Extends the relational domain calculus $(\wedge, \vee, \neg, \exists, \forall, R_i(X, Y), \ldots)$ by integrating a similarity operator $(\approx)$
- Based on a retrieval model using the vector space model of quantum mechanic and logic
- Detailed mathematical description of this model is omitted in this talk

Basic idea



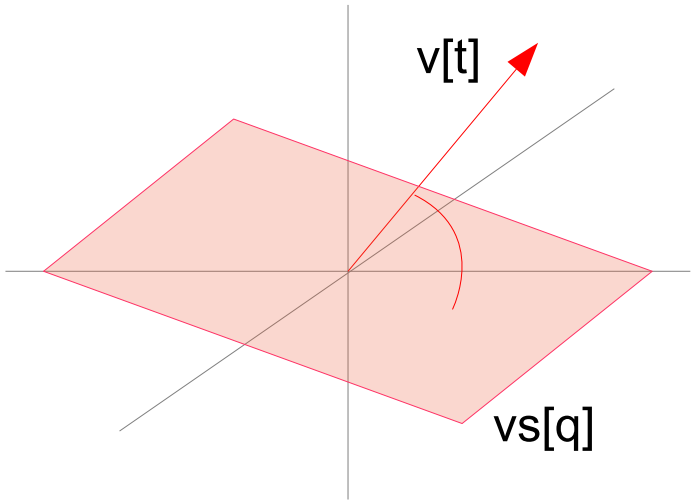
Query processing		**CQQL model**
value domain	$\leftrightarrow$	vector space
tuple	$\leftrightarrow$	vector
query	$\leftrightarrow$	condition space
evaluation result	$\leftrightarrow$	squared cosine of the angle between vector and condition space

Basic idea



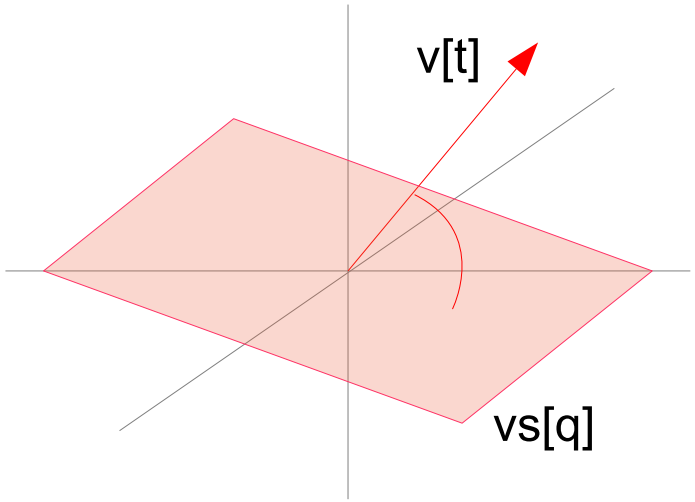
Query processing		**CQQL model**
value domain	$\leftrightarrow$	vector space
tuple	$\leftrightarrow$	vector
query	$\leftrightarrow$	condition space
evaluation result	$\leftrightarrow$	squared cosine of the angle between vector and condition space

Basic idea



Query processing		**CQQL model**
value domain	$\leftrightarrow$	vector space
tuple	$\leftrightarrow$	vector
query	$\leftrightarrow$	condition space
evaluation result	$\leftrightarrow$	squared cosine of the angle between vector and condition space

Basic idea



Query processing		**CQQL model**
value domain	$\leftrightarrow$	vector space
tuple	$\leftrightarrow$	vector
query	$\leftrightarrow$	condition space
evaluation result	$\leftrightarrow$	squared cosine of the angle between vector and condition space

Evaluation of a CQQL query

- After a syntactically normalisation a condition can be recursively evaluated by

$$
\begin{array}{lcl}
sv(t_i, q) & := & \varphi(t_i^{attr}, q) \text{ if q is an atom} \\
eval(t_i, q_1 \wedge q_2) & := & eval(t_i, q_1) * eval(t_i, q_2) \\
eval(t_i, q_1 \vee q_2) & := & eval(t_i, q_1) + eval(t_i, q_2) - \\
 & & eval(t_i, q_1 \wedge q_2) \\
eval(t_i, \neg q) & := & 1 - eval(t_i, q)
\end{array}
$$

- Can also be interpreted as probability aggregation of independent events

Evaluation of a CQQL query

- After a syntactically normalisation a condition can be recursively evaluated by

$$
\begin{array}{lcl}
eval(t_i, q) & := & \varphi(t_i^{attr}, q) \text{ if q is an atom} \\
eval(t_i, q_1 \wedge q_2) & := & eval(t_i, q_1) * eval(t_i, q_2) \\
eval(t_i, q_1 \vee q_2) & := & eval(t_i, q_1) + eval(t_i, q_2) - \\
 & & eval(t_i, q_1 \wedge q_2) \\
eval(t_i, \neg q) & := & 1 - eval(t_i, q)
\end{array}
$$

- Can also be interpreted as probability aggregation of independent events

Evaluation of a CQQL query

- After a syntactically normalisation a condition can be recursively evaluated by

$$
\begin{array}{lcl}
eval(t_i, q) & := & \varphi(t_i^{attr}, q) \text{ if q is an atom} \\
eval(t_i, q_1 \wedge q_2) & := & eval(t_i, q_1) * eval(t_i, q_2) \\
eval(t_i, q_1 \vee q_2) & := & eval(t_i, q_1) + eval(t_i, q_2) - \\
 & & eval(t_i, q_1 \wedge q_2) \\
eval(t_i, \neg q) & := & 1 - eval(t_i, q)
\end{array}
$$

- Can also be interpreted as probability aggregation of independent events

Evaluation of a CQQL query

- After a syntactically normalisation a condition can be recursively evaluated by

$$
\begin{array}{lcl}
eval(t_i, q) & := & \varphi(t_i^{attr}, q) \text{ if q is an atom} \\
eval(t_i, q_1 \wedge q_2) & := & eval(t_i, q_1) * eval(t_i, q_2) \\
eval(t_i, q_1 \vee q_2) & := & eval(t_i, q_1) + eval(t_i, q_2) - \\
 & & eval(t_i, q_1 \wedge q_2) \\
eval(t_i, \neg q) & := & 1 - eval(t_i, q)
\end{array}
$$

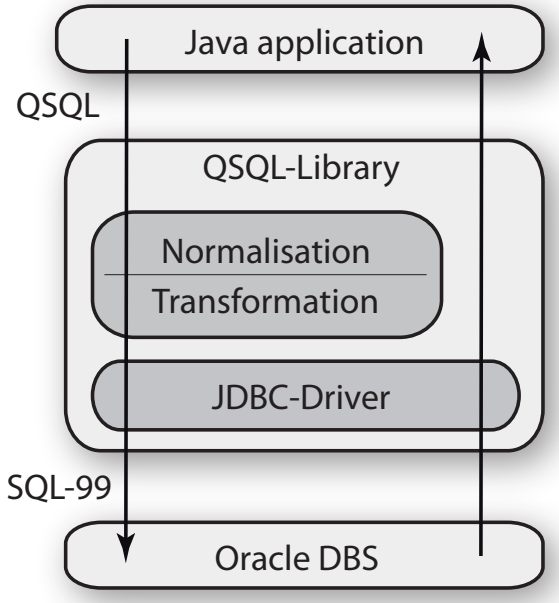
- Can also be interpreted as probability aggregation of independent events

Outline

- Motivation
- CQQL retrieval model
- Incorporating CQQL concepts into SQL
- Experiments and outlook

QSQL characteristics

- Designed as a SQL dialect
- Covers the core functionality of SQL-92: selection, projection, union, . . .
- Query processing is performed by a mapping between QSQL and SQL-99
- Implemented as JDBC driver library for the Oracle DBS



Selection

- Similarity conditions are placed in the where-clause
- Attribute `scoreval` is automatically generated

- Example query: Determine all *available* TV sets with a handling as easy as possible.
- Syntax in QSQL:

```
select name
from tv_set
where ( status='available' and ha ~ 1 ) and scoreval > 0
order by scoreval desc
```

Selection

- Mapping to SQL-99 by using auxiliary functions implemented in PL/SQL:
  - Score functions $\rightsquigarrow$ HA_TO_SCORE, ...
  - Logical connector functions $\rightsquigarrow$ LAND, ...
  - ...
- Mapping to SQL-99:

```
select name, LAND( EQUAL( status, 'available' ),
  HA_TO_SCORE( ha, 1 ) ) as scoreval
from tv_set
where scoreval > 0
order by scoreval desc
```

Union

- Example query: Determine all producers which are capable to offer a TV set or a DVD player with a handling as easy as possible.
- Syntax in QSQL:

```
select producer from tv_set where ha ~ 1
  union
select producer from dvd_player where ha ~ 1
```

Union

- Evaluation semantics for $E = E_1 \cup E_2$
- Score value computation is split in three cases

$$sv_E(t) = \begin{cases} sv_{E_1}(t) + sv_{E_2}(t) - sv_{E_1}(t) * sv_{E_2}(t) & \text{if } t \in E_1 \wedge t \in E_2 \\ sv_{E_1}(t) & \text{if } t \in E_1 \wedge t \notin E_2 \\ sv_{E_2}(t) & \text{if } t \notin E_1 \wedge t \in E_2 \end{cases}$$

Union

- Mapping to SQL-99:

```
select producer, max( scoreval )
  as scoreval
from (
    select E1.producer, LOR( E1.scoreval, E2.scoreval )
    as scoreval
    from
      ( select producer, HA_TO_SCORE( ha, 1 ) as scoreval
        from tv_set ) E1,
      ( select producer, HA_TO_SCORE( ha, 1 ) as scoreval
        from dvd_player ) E2
    where E1.producer = E2.producer
  union all
    select producer, HA_TO_SCORE( ha, 1 ) as scoreval
    from tv_set
    where producer not in ( select producer from dvd_player )
  union all
    select producer, HA_TO_SCORE( ha, 1 ) as scoreval
    from dvd_player
    where producer not in ( select producer from tv_set ) )
group by producer
```

Outline

- Motivation
- CQQL retrieval model
- Incorporating CQQL concepts into SQL
- Experiments and outlook

Experiments

- Test parameters:
  - Table of 100,000 artificial TV sets
  - Query: $(ha \approx 1 \ \wedge \ iq \approx 1) \ \wedge \ (osp = no \Rightarrow sq \approx 1)$
  - Implemented on an Oracle 11g DBS
- Boolean-based evaluation is clearly faster
- Final sorting step is crucial

Approach	Seconds
SQL	0.04
QSQL/unsorted	0.14
Skyline	1.01
Fuzzy/sorted	19.01
QSQL/sorted	19.25

Outlook

- Extending QSQL by further operations
- Implementation of a k-top operator to avoid a full sorting
- Applying more efficient mapping techniques

End

- Thank you for your attention.