Database Systems

R4 Cheng

January 24, 2025

Best practice:

1. Always prepare your own key

Remark. *Say what you want instead of how to do*

Use IN/Not in to check if a value is in a set

Datalog

Each query is a rule.

E.g. For all values of Part, Subpart, and Qty,

if there is a tuple ⟨Part, Subpart, Qty⟩ in Assembly,

then there must be a tuple ⟨Part, Subpart⟩ in Components.

```
Components(Part, Subpart) :- Assembly(Part, Subpart, Qty).
```

E.g. For all values of Part, Part2, Subpart, and Qty,

if there is a tuple ⟨Part, Part2, Qty⟩ in Assembly, **and** a tuple ⟨Part2, Subpart⟩ in Components,

then there must be a tuple <Part, Subpart>in Components.

```
Components(Part, Subpart) :- Assembly(Part, Part2, Qty),
                             Components(Part2, Subpart).
```

Remark. *Each application of a Datalog rule can be understood in terms of relational algebra*

Unsafe rules

```
(Unsafe) V(x, y, z) :- Actor(x, y, 1998), z > 200
```

This is unsafe because **z** is not bound to any relation, meaning it can take on infinitely many values.

```
(Unsafe) W(x,y,z) :- Actor(x,y,z), not Plays(t,x)
```

This is unsafe because The variable **t** appears only in the negated literal not Plays(t, x) and does not appear in any positive literal in the body

Remark. *Every variable should appear in at least one positive body atom*

Relational Algebra

- Defines a set of basic operations on relations
- Each operation returns a relation
- **Result** of an operation can be the input of another operation

Basic operations:

- Selection (σ): Selects a subset of rows from relation.
- Projection (π): Deletes attributes that are not in the projection list and deletes duplicate rows.
- Union ($\cup$): Tuples in relation 1 and in relation 2.
- Set-difference ($-$): Tuples in relation 1 but not in relation 2.
- Cross product ($\times$): Allows us to combine two relations. it returns all possible pairs of tuples from the two relations.

Join is a combination of selection and cross product.

$$R \bowtie S = \sigma_{condition}(R \times S)$$

Relational Calculus

- First-order logic
- Tuple relational calculus (TRC)
- Domain relational calculus (DRC)

Each relational predicate P is:

- Atom (Actor(x, y, z))
- P $\wedge$ P (conjunction)
- P $\vee$ P (disjunction)
- P $\Rightarrow$ P (implication)
- $\neg$ P (negation)
- $\forall$ x P (for all x P holds)
- $\exists$ x P (for an x P holds)

Examples

Exists a schema:

$Movie(\underline{mid}, title, year, total-gross)$
$Actor(\underline{aid}, name, b-year)$
$Plays(\underline{mid}, \underline{aid})$

Q: Actor who played only in movies produced in 1990

$Result(x) = \forall y.Play(y, x) \Rightarrow \exists z \exists t.Movie(y, z, 1990, t)$

Tuple Relational Calculus

Form: $\{T \mid p(T)\}$

The result of this query is the set of all tuples t for which the formula $p(T)$ evaluates to true with $T = t$.

Domain Relational Calculus

Form: $\{< x_1, x_2, \ldots, x_n > \mid p(< x_1, x_2, \ldots, x_n >)\}$, where each x_i is either a domain variable or a constant and $p(< x_1, x_2, \ldots, x_n >)$ denotes a **DRC formula**.

Index

An **index** is a data structure that organizes data records on disk to optimize certain kinds of retrieval operations.

We use the term **data entry** to refer to the records stored in an index file.

There are three main alternatives for what to store as a data entry in an index:

1. A data entry $k*$ is an actual data record (with searh key value k).
2. A data entry $< k, rid >$ pair, where rid is the record id of a data record with search key value k.
3. A data entry $< k, red-list >$ pair, where $red-list$ is a list of record ids of data records with search key value k.

1. Hash-based Indexing
2. Tree-based Indexing

clustered index:

unclustered index:

sop:

JOIN Algorithms

TO TA

- why in 2 pass merge sort the number of M unit increases?
- book 9.7 why to point free space there?

TODO

- fully understand pages, frames, and buffer pool