Homework 04

Zhicheng Zhang

Relational Algebra 1

The video provides the introduction of relational algebra. At the very beginning, an example of simple college admissions database is given. Then, it shows us 4 types of operators as following.

- The SELECT operator R = σS(P) is used to return a subset of tuples of a relation based on a selection condition. The latter can consist of predicates combined using Boolean operators (AND, OR, NOT) (Lemahieu, Broucke & Baeses, 2018).
- The PROJECT operator R = πL(P) is used to project a relation on a list of attribute types (Lemahieu, Broucke & Baeses, 2018).
- The CROSS-PRODUCT operator R = P × Q is used to combine two relations. If P has a tuples and Q has b tuples, R will have ab tuples (Lemahieu, Broucke & Baeses, 2018).
- The JOIN operator R= P ⋈j Q allows to combine tuples of two relations based on specific conditions, also called join conditions. The result of a join is the combined effect of a Cartesian product and selection (Lemahieu, Broucke & Baeses, 2018).

Relational Algebra 2

The video shows us other 4 types of operator as following.

- The UNION operator applied to two union-compatible relations P and Q results into a new relation R = P ∪ Q so: R = {t | t ∈ P OR t ∈ Q} with max(|P|, |Q|) ≤ |R| ≤ |P| + |Q| (Lemahieu, Broucke & Baeses, 2018).
- The DIFFERENCE operator applied to two union-compatible relations P and Q results into a new relation R = P - Q such that: R = {t | t ∈ P AND t ∉ Q} with 0 ≤ |R| ≤ |P| (Lemahieu, Broucke & Baeses, 2018).
- The INTERSECTION operator applied to two union-compatible relations P and Q results into a new relation R = P ⋂ Q such that: R = {t | t ∈ P AND t ∈ Q} with 0 ≤ |R| ≤ |min(|P|, |Q|)| (Lemahieu, Broucke & Baeses, 2018).
- The RENAME operator is used to rename an attribute type of a relation (in case, e.g., naming conflicts can occur). The result of applying a RENAME operator on a relation

P to rename attribute type B to A can be represented as: R = ρA|B(P) with RENAME operator ρ (Lemahieu, Broucke & Baeses, 2018).

Functional Dependencies and Normalization 1

The video provides some tips of how to optimize database schema. It takes college application information as an example.

First, 3 type of anomalies is mentioned, they are:

- Redundancy
- Update anomaly - an issue encountered when a tuple in an unnormalized table is updated, causing the need to make multiple updates with the risk of inconsistency (Lemahieu, Broucke & Baeses, 2018).
- Deletion anomaly - an issue encountered when a tuple in an unnormalized table is deleted, resulting in the deletion of all corresponding data (Lemahieu, Broucke & Baeses, 2018).

To solve these problems, the initial "mega" relation should be decomposed to smaller and better relations.

Then, 5 type of normal from (NF) is mentioned, they are:

- The first normal form (1NF) - states that every attribute type of a relation must be atomic and single-valued. Hence, no composite or multi-valued attribute types are tolerated (Lemahieu, Broucke & Baeses, 2018).
- The second normal form (2NF) - when 1 NF is satisfied and every non-prime attribute type A in R is fully functionally dependent on any key of R (Lemahieu, Broucke & Baeses, 2018).
- The third normal form (3NF) - when 2 NF is satisfied and no non-prime attribute type of R is transitively dependent on the primary key (Lemahieu, Broucke & Baeses, 2018).
- The Boyce-Codd normal form (BCNF) - a normal form such that for each of the non-trivial functional dependencies X → Y, X is a superkey (Lemahieu, Broucke & Baeses, 2018).
- The fourth normal form (4NF) - when Boyce–Codd normal form is satisfied and for every non-trivial multivalued dependency X →→ Y, X is a superkey (Lemahieu, Broucke & Baeses, 2018).

By describing BCNF and 4NF, concepts of functional dependency and multivalued dependency is mentioned:

- Functional dependency (related to BCNF)- a constraint in which one value of an attribute type determines the value of another (Lemahieu, Broucke & Baeses, 2018).

- Multivalued dependency (related to 4NF) - multi-valued dependency a dependency X $\rightarrow\rightarrow$ Y, such that each X value exactly determines a set of Y values, independently of the other attribute types (Lemahieu, Broucke & Baeses, 2018).

Functional Dependencies and Normalization 2

The video shows us some details about functional dependency. It is a useful conception about data compression and query optimization.

As mentioned before, A functional dependency $X \rightarrow Y$ between two sets of attribute types X and Y implies that a value of X uniquely determines a value of Y. In other words, there is a functional dependency from X to Y or that Y is functionally dependent on X (Lemahieu, Broucke & Baeses, 2018).

If all attributes in a table is functional dependency for a set of attributes, such set of attributes are named as keys.

There are still some concepts about functional dependency:

- Trivial functional dependency - a functional dependency $X \rightarrow Y$ where Y is a subset of X. (Lemahieu, Broucke & Baeses, 2018).

- Non-trivial functional dependency - Y is not a subset of X.

- Completed non-trivial functional dependency - X and Y have no interception at all.

There are some rules of functional dependency:

- Splitting rule: $A \rightarrow B_1, B_2, B_3, \ldots, B_n \Rightarrow A \rightarrow B_1, A \rightarrow B_2, A \rightarrow B_3, \ldots, A \rightarrow B_n$

- Combining rule: $A \rightarrow B_1, A \rightarrow B_2, A \rightarrow B_3, \ldots, A \rightarrow B_n \Rightarrow A \rightarrow B_1, B_2, B_3, \ldots, B_n$

- Trivial-dependency rule 1: $A \rightarrow B \Rightarrow A \rightarrow A \cup B$

- Trivial-dependency rule 1: $A \rightarrow B \Rightarrow A \rightarrow A \cap B$

- Transitive rule: $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$

In the end, the video shows what we want in relations - “minimal set of completely non-trivial functional dependencies such that all functional dependencies that hold on the relation follow from the dependencies in this set”.

Functional Dependencies and Normalization 3

The video shows us how to use functional dependency to create Boyce-Codd normal form (BCNF).

As mentioned before, A relation R is in the BCNF provided that for each of its non-trivial functional dependencies X → Y, X is a superkey - that is, X is either a candidate key or a superset thereof. It can be shown that the BCNF is stricter than the 3NF. (Lemahieu, Broucke & Baeses, 2018).

The BCNF decomposition algorithm:

```
Input: relation R + FDs for R
Output: decomposition of R into BCNF relationship with “lossless join”

Compute keys for R
Repeat until all relations are in BCNF:
        Pick any R’ with A → B that violates BCNF
        Decompose R’ into R1(A, B) and R2(A, rest)
        Compute FDs for R1 and R2
        Compute keys for R1 and R2
```

Functional Dependencies and Normalization 4

Topic of the videos are multivalued dependency and the fourth normal form (4NF).

The concept of multivalued dependency is: There is a multi-valued dependency from X to Y , X →→ Y , if and only if each X value exactly determines a set of Y values, independently of the other attribute types (Lemahieu, Broucke & Baeses, 2018).

The concept of the fourth normal form (4NF) is: A relation is in the 4 NF if it is in BCNF and for every one of its non-trivial multivalued dependencies X →→ Y , X is a superkey – that is, X is either a candidate key or a superset thereof (Lemahieu, Broucke & Baeses, 2018).

There are still some concepts about multivalued dependency:

- Trivial multivalued dependency - a multivalued dependency X →→ Y where Y is a subset of X **OR** X and Y are all attributes.

- Non-trivial multivalued dependency - Y is not a subset of X **AND** X and Y are not all attributes.

There are some rules of functional dependency:

- FD-is-an-MVD rule: A → B ➔ A →→ B

- Transitive rule: A →→ B, B →→ C ➔ A →→ C

The 4NF decomposition algorithm:

```
Input: relation R + FDs for R + MVDs for R
Output: decomposition of R into 4NF relationship with “lossless join”

Compute keys for R
Repeat until all relations are in 4NF:
        Pick any R’ with A →→ B that violates 4NF
        Decompose R’ into R1(A, B) and R2(A, rest)
        Compute FDs and MVDs for R1 and R2
        Compute keys for R1 and R2
```

Functional Dependencies and Normalization 5

The video provides an opinion that not all design in BDNF and 4NF are good design. Several examples are given to point out the shortcoming - after decomposition, no guarantee dependencies can be checked on decomposed relations. In other word, it will increase query workload and time complexity.

Lemahiew’s Book also mentions that “Denormalization can be achieved by combining two or more normalized tables into one. By doing so, join queries will execute faster since now only one table needs to be considered. However, this comes at a price of redundant data and possibly inconsistent data as well if not appropriately managed” (Lemahieu, Broucke & Baeses, 2018).

As a result, normalizing relations to a higher level is not always a good idea. Query efficiency should also be considered when designing a database.

Indexes

Indexes is a primary mechanism to improve database performance. Index is a stored, ordered list of key values that is part of the internal data model and that provides a fast access path to the physical data to speed up the execution time of a query (Lemahieu, Broucke & Baeses, 2018). Many DBMS's tend to build indexes for primary key automatically.

There are two basic data structure used by indexes:

- balance tree (B tree and B+ tree) - equality and range – logarithmic running time
- hash table - equality only - constant running time

Downsides of Indexes:

- Indexes takes extra space.
- Indexes creation takes extra time.
- Frequent indexes maintenance (caused by update operation) may offset its benefits.

The benefit of an index depends on:

- size of table
- data distribution
- query load vs. update load

To help make better indexes, physical design advisor is designed to provide some recommendations.

Reference

Lemahieu, W., Broucke, S. V., & Baeses B. (2018). Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data. New York, NY: Cambridge University Press.