Principles of Data Management

Alvin Lin

August 2018 - December 2018

Normalization

Normalization is a way to make bad data good by dealing with anomalies. Anomalies occur when data is inserted, deleted, or updated in an inconsistent way, causing rows to contain different data formats. This is usually solved by decomposition. If many rows of a table are going to the same data, we can decompose the table into two tables. For example, instead of:

```
Game(1, RDR2, M, \$59.99, RPG)
Game(18, RDR2, M \$59.99, RPG)
```

We can decompose this into two tables and store RDR2 in its own Game table as a template which we can update to reflect the change across all instances.

```
Game(RDR2, M \$59.99, RPG)
GameInstance(18, RDR2)
GameInstance(1, RDR2)
```

Decomposition can be either lossy or lossless, depending on whether or not the original structure of the data can be recovered. In this case, the decomposition is lossless since we can obtain the original records by joining the tables on the game title.

Function Dependencies

Certain fields can map to other fields. For example, ZIP codes can map to cities and states. We can make certain attributes in a table dependent on other attributes. Armstrong's Axioms:

- if $X \supseteq Y$ then $X \to Y$ is reflexivity

- if $X \to Y$ then $XZ \to YZ$ is augmentation
- if $X \to Y$ and $Y \to Z$, then $X \to Z$ is transitivity

We use Armstrong's Axioms to infer all functional dependencies and get the minimal set of functional dependencies that include all possible attributes.

- First normal form: all attributes are atomic.
- Second normal form: must be in first normal form, no partial key dependencies.
- Third normal form: must be in second normal form, no transitive dependencies.
- Boyce-Codd normal form (BCNF): stronger form of third normal form, no non-candidate key dependencies.

Denormalization

In order to denormalize, you must have normalized data to start with. This is usually performed as a method of combining tables. This reduces the number of tables and reduces the number of necessary join operations, which are often expensive. As a result, create, update, and delete operations are slowed down but read operations are sped up. Redundant data will occur and there will be less foreign keys.

Partitioning

Horizontal and vertical partitioning is a form of potentially lossy decomposition. Suppose we have a table with attributes:

ID	name	age	street	city

Vertical partitioning:

ID	name

ID	age	street	city

Horizontal partitioning involves putting the rows of a large database into multiple databases. For example, users with $age > 30$ will be put into one database and users with $age \leq 30$ will be put into another database with a view containing the union of both for access to all users. This is done for faster access, especially since loading large tables into memory is expensive and slow.