Experiences with Building, Querying and Mining NoSQL Column Wide Databases

Yusriyah Rahman, Meet Solanki, and Christie I. Ezeife

School of Computer Science, University of Windsor
401 Sunset Avenue, Windsor, ON N9B3P4, Canada

1 Row Wise to Column Wide

Figure 1 and 2 display the database which records data about students enrolled in courses and their grades. The row wise schema in Figure 1 has 3 entities, while the column wide schema in Figure 2 expands to 11, making it almost 4 times larger, since it consists of 8 additional entities. Each attribute was separated into its own entity linked through primary and foreign keys. An additional entity, TakesID, introduced a unique primary key (tid), resulting in one more entity than the total number of original attributes. The implementation of the tid primary key resulted in the column wide schema to consist of 11 attributes instead of 10 as in the row wise schema.

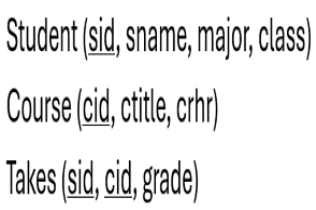

Fig. 1. Row wise schema

Fig. 2. Column wide schema

2 Query Performance Analysis

Figures 3 and 4 are row and column wide queries respectively. They display student's IDs and their corresponding cumulative averages using the row wise and column wide schemata, respectively according to the figures.

```
SELECT sid, AVG(grade)
FROM Takes
GROUP BY sid;
```

Fig. 3. Row wise query

```
SELECT TS.sid, AVG(TG.grade)
FROM TakesStudentID TS
JOIN TakesGrade TG ON TS.tid = TG.tid
GROUP BY TS.sid;
```

Fig. 4. Column wide query

The row wise query fetches entire tuple and attributes. Here for example, the query reads the Takes table and fetches all attributes sid, cid, and grade even though only sid and grade are required, resulting in Takes to be scanned tuple by tuple. This increases computation time (CPU decoding overhead) to 1.20 seconds for example with 1,000,000 rows of 1KB each. Column wide queries however fetch only query related attributes, which are sid and grade in this case. This makes the column wide database optimal as it reduces memory usage, disk input and output, and computation time, like to 0.43 seconds for our instance. Therefore, the column wide database executes this query about 0.77 seconds or 64% faster.

3 Applying the Apriori Algorithm

The Apriori Algorithm can be applied to the transactional dataset in Table 1 generated from the row wise or column wide schemata in Figure 1 or 2, respectively. Since Figure 1 and Figure 2 consist of the same data, when the Apriori algorithm is applied to them, they yield identical results.

sid	Courses Taken
1	{Comp1400, Math1720}
2	{Comp1400}

Table 1. Transactional Dataset

Item set	Support
{Comp1400}	2/2 = 100%
{Math1720}	1/2 = 100%
{Comp1400, Math1720}	1/2 = 50%

Table 2. Item set Supports

Rule	Confidence
Comp1400 → Math1720	1/2 = 50%
Math1720 → Comp1400	1/1 = 100%

Table 3. Association Rules and Confidence Levels

If the minimum support threshold is 50%, then all item sets are frequent, as shown in Table 2 with support levels. From the item set "{Comp1400, Math1720}" the rules "50% of students who took Comp1400 also took Math1720" and "all students who took Math1720 also took Comp1400" may be derived as shown in Table 3 along with their respective confidence levels determining the rules.

References

1. Elmasri, R., Navathe, S.B.: *Fundamentals of Database Systems*, 7th edn. Pearson Education, Boston (2016).
2. Shahnawaz, M., Kumar, M.: A comprehensive survey on big data analytics: characteristics, tools and techniques. *ACM Computing Surveys* 57(8), Article 196 (2025). https://doi.org/10.1145/3718364