Australian National University

Normalisation – Part 3

Summary and Discussion

Summary of Normal Forms

- 1NF, 3NF and BCNF [illegible]r in practice. Other normal forms are rarely used.

1NF: only atomic values for attributes (*part of the definition for the relational data model*);

2NF: an intermediate result in the history of database design theory;

3NF: lossless and dependencies can be preserved;

BCNF: lossless but dependencies may not be preserved.

- 3NF can only **minimise (not necessarily eliminate) redundancy**. So a relation schema in 3NF may still have update anomalies.
- A relation schema in BCNF **eliminates redundancy**.

Why Denormalisation?

- **Do we need to norr ion schemas in all cases** when designing a relational database

- The normalisation process **may degrade performance** when data are frequently queried.

- Since relation schemas are decomposed into many smaller ones after normalisation, queries need to **join many relations together** in order to return the results.

- Unfortunately, **join operation is very expensive**.

- When data is **more frequently queried rather than being updated** (e.g., data warehousing system), a weaker normal form is desired (i.e., **denormalisation**).

Denormalisation

- **Denormalisation** is [illegible]**rocess** that
  - happens after t[illegible]sation process,
  - is often performed during the physical design stage, and
  - reduces the number of relations that need to be joined for certain queries.

- We need to distinguish:
  - **Unnormalised** – there is no systematic design.
  - **Normalised** – redundancy is reduced after a systematic design (to minimise data inconsistencies).
  - **Denormalised** – redundancy is introduced after analysing the normalised design (to improve efficiency of queries)

Trade-offs

- A good database design is to **find a balance** between desired properties, then normalise/denormalise relations to a desired degree.

Trade-offs – Data Redundancy vs. Query Efficiency

- Normalisation: **No D[illegible]dancy but No Efficient Query Processing**
- Data redundancies a[illegible]ed in the following relations.

STUD[illegible]

Name	StudentID	DoB
Tom	123456	25/01/1988
Michael	123458	21/04/1985

COURSE

CourseNo	Unit
COMP2400	6
COMP8740	12

ENROL

StudentID	CourseNo	Semester
123456	COMP2400	2010 S2
123456	COMP8740	2011 S2
123458	COMP2400	2009 S2

- However, the query for “list the names of students who enrolled in a course with 6 units” requires 2 join operations.

```
SELECT Name, CourseNo FROM ENROL e, COURSE c, STUDENT s WHERE
e.StudentID=s.StudentID and e.CourseNo=c.CourseNo and c.Unit=6;
```

Trade-offs – Data Redundancy vs. Query Efficiency

- Denormalisation: **Da[illegible]ancy but Efficient Query Processing**
- If a student enrolled [illegible], then the name and DoB of this student need to be stored repeatedly 15 times in ENROLMENT.

ENROLMENT

Name	StudentID	DoB	CourseNo	Semester	Unit
Tom	123456	25/01/1988	COMP2400	2010 S2	6
Tom	123456	25/01/1988	COMP8740	2011 S2	12
Michael	123458	21/04/1985	COMP2400	2009 S2	6

- However, the query for “list the names of students who enrolled a course with 6 units” can be processed efficiently (no join needed).

```
SELECT Name, CourseNo FROM ENROLMENT WHERE Unit=6;
```

Discussion

- Both normalisation and denormalisation are useful in database design.
  - **Normalisation**: obtain database schema avoiding redundancies and data inconsistencies
  - **Denormalisation**: join normalized relation schemata for the sake of better query processing
- Some problems of (de-)normalisation:
  - FDs **cannot handle null values**.
  - To apply normalisation, FDs must be **fully specified**.
  - The algorithms for normalisation **are not deterministic**, leading to different decompositions.