Database Management Systems

Module 30: Relational Database Design/10: Design Summary and Temporal Data

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Understood multi-valued dependencies to handle attributes that can have multiple values
- Learnt Fourth Normal Form and decomposition to 4NF

Module Objectives

- To summarize the database design process
- To explore the issues with temporal data

Module Outline

- Database-Design Process
- Modeling Temporal Data

Database Design Process

Design Goals

- Goal for a relational database design is:
  - BCNF / 4NF
  - Lossless join
  - Dependency preservation
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.
- Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key

Further Normal Forms

- Further NFs
  - Elementary Key Normal Form (EKNF)
  - Essential Tuple Normal Form (ETNF)
  - Join Dependencies And Fifth Normal Form (5 NF)
  - Sixth Normal Form (6NF)
  - Domain/Key Normal Form (DKNF)
- **Join dependencies** generalize multivalued dependencies
  - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
- A class of even more general constraints, leads to a normal form called **domain-key normal form**.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used

Overall Database Design Process

- We have assumed schema R is given
  - R could have been generated when converting E-R diagram to a set of tables
  - R could have been a single relation containing all attributes that are of interest (**universal relation**)
  - Normalization breaks R into smaller relations
  - R could have been the result of some ad hoc design of relations, which we then test/convert to normal form

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an employee entity with attributes
    $$department\_name \text{ and } building,$$
    and a functional dependency
    $$department\_name \rightarrow building$$
  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but rare — most relationships are binary

Denormalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying prereqs along with course_id, and title requires join of course with prereq
  - **Course(course_id, title, . . .)**
  - **Prerequisite(course_id, prereq)**
- Alternative 1: Use denormalized relation containing attributes of course as well as prereq with all above attributes: **Course(course_id, title, prereq, . . .)**
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: Use a materialized view defined as **Course** $\bowtie$ **Prerequisite**
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:
  Instead of earnings (*company_id, year, amount*), use
  - *earnings_2004, earnings_2005, earnings_2006*, etc., all on the schema (*company_id, earnings*).
    - Above are in BCNF, but make querying across years difficult and needs new table each year
  - *company_year (company_id, earnings_2004, earnings_2005, earnings_2006*)
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
    - Is an example of a **crosstab**, where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools

LIS Example for 4NF

- Consider a different version of relation **book_catalogue** having the following attributes:
  - *book_title*
  - *book_catalogue*, *author_lname*: A *book_title* may be associated with more than one author.
- **book_title** {*book_title*, *author_fname*, *author_lname*, *edition*}

book_title	author_fname	author_lname	edition
DBMS CONCEPTS	BRINDA	RAY	1
DBMS CONCEPTS	AJAY	SHARMA	1
DBMS CONCEPTS	BRINDA	RAY	2
DBMS CONCEPTS	AJAY	SHARMA	2
JAVA PROGRAMMING	ANITHA	RAJ	5
JAVA PROGRAMMING	RIYA	MISRA	5
JAVA PROGRAMMING	ADITI	PANDEY	5
JAVA PROGRAMMING	ANITHA	RAJ	6
JAVA PROGRAMMING	RIYA	MISRA	6
JAVA PROGRAMMING	ADITI	PANDEY	6

Figure: **book_catalogue**

LIS Example 4NF (2)

book_title	author_fname	author_lname	edition
DBMS CONCEPTS	BRINDA	RAY	1
DBMS CONCEPTS	AJAY	SHARMA	1
DBMS CONCEPTS	BRINDA	RAY	2
DBMS CONCEPTS	AJAY	SHARMA	2
JAVA PROGRAMMING	ANITHA	RAJ	5
JAVA PROGRAMMING	RIYA	MISRA	5
JAVA PROGRAMMING	ADITI	PANDEY	5
JAVA PROGRAMMING	ANITHA	RAJ	6
JAVA PROGRAMMING	RIYA	MISRA	6
JAVA PROGRAMMING	ADITI	PANDEY	6

Figure: **book_catalogue**

- Since the relation has no FDs, it is already in BCNF.
- However, the relation has two nontrivial MVDs $book\_title \twoheadrightarrow \{author\_fname, author\_lname\}$ and $book\_title \twoheadrightarrow edition$. Thus, it is not in 4NF.
- Nontrivial MVDs must be decomposed to convert it into a set of relations in 4NF.

LIS Example 4NF (3)

book_title	author_fname	author_lname
DBMS CONCEPTS	BRINDA	RAY
DBMS CONCEPTS	AJAY	SHARMA
JAVA PROGRAMMING	ANITHA	RAJ
JAVA PROGRAMMING	RIYA	MISRA
JAVA PROGRAMMING	ADITI	PANDEY

Figure: **book_author**

book_title	edition
DBMS CONCEPTS	1
DBMS CONCEPTS	2
JAVA PROGRAMMING	5
JAVA PROGRAMMING	6

Figure: **book_edition**

- We decompose **book_catalogue** into **book_author** and **book_edition** because:
  - **book_author** has trivial MVD $book\_title \twoheadrightarrow \{author\_fname, author\_lname\}$
  - **book_edition** has trivial MVD $book\_title \twoheadrightarrow edition$.

Temporal Databases

Temporal Databases

- Some data may be inherently historical because they include time-dependent / time-varying data, such as:
  - Medical Records
  - Judicial records
  - Share prices
  - Exchange rates
  - Interest rates
  - Company profits
  - etc.
- The desire to model such data means that we need to store not only the respective value but also an associated date or a time period for which the value is valid. Typical queries expressed informally might include:
  - Give me last month's history of the Dollar-Pound Sterling exchange rate.
  - Give me the share prices of the NYSE on October 17, 1996.
- Temporal databases provide a uniform and systematic way of dealing with historical data

Source: https://www.cs.uct.ac.za/mit_notes/database/htmls/chp18.html

Temporal Data

- **Temporal data** have an association time interval during which the data are valid.
- A **snapshot** is the value of the data at a particular point in time
- In practice, database designers may add start and end time attributes to relations
- For example, *course(course_id, course_title)* is replaced by
  course(course_id, course_title, start, end)
  - Constraint: no two tuples can have overlapping valid times and are Hard to enforce efficiently
  - Foreign key references may be to current version of data, or to data at a point in time
    - For example, student transcript should refer to course information at the time the course was taken

Temporal Database Theory

- **Model of Temporal Domain**: Single-dimensional linearly ordered which may be
  - Discrete or dense
  - Bounded or unbounded
  - Single dimensional or multi-dimensional
  - Linear or non-linear
- **Timestamp Model**
- **Temporal ER model** by adding valid time to
  - Attributes: address of an instructor at different points in time
  - Entities: time duration when a student entity exists
  - Relationships: time during which a student attended a course
  - But no accepted standard
- **Temporal Functional Dependency Theory**
- **Temporal Logic**
- **Temporal Query Languge**: TQuel [1987], TSQL2 [1995], SQL/Temporal [1996], SQL/TP [1997]

Modeling Temporal Data: Uni / Bi Temporal

- There are **two different aspects** of time in temporal databases.
  - **Valid Time**: Time period during which a fact is true in real world, provided to the system.
  - **Transaction Time**: Time period during which a fact is stored in the database, based on transaction serialization order and is the timestamp generated automatically by the system.
- Temporal Relation is one where each tuple has associated time; either valid time or transaction time or both associated with it.
  - **Uni-Temporal Relations**: Has one axis of time, either *Valid Time* or *Transaction Time*.
  - **Bi-Temporal Relations**: Has both axis of time – *Valid time* and *Transaction time*. It includes Valid Start Time, Valid End Time, Transaction Start Time, Transaction End Time.

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Modeling Temporal Data: Example (1)

- **Example.**
  - Let's see an example of a person, John:
    - John was born on April 3, 1992 in Chennai.
    - His father registered his birth after three days on April 6, 1992.
    - John did his entire schooling and college in Chennai.
    - He got a job in Mumbai and shifted to Mumbai on June 21, 2015.
    - He registered his change of address only on Jan 10, 2016.

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Modeling Temporal Data: Example (2)

- John's Data In Non-Temporal Database

Date	Real world event	Address
April 3, 1992	John is born	
April 6, 1992	John's father registered his birth	Chennai
June 21, 2015	John gets a job	Chennai
Jan 10, 2016	John registers his new address	Mumbai

In a non-temporal database, John's address is entered as Chennai from 1992. When he registers his new address in 2016, the database gets updated and the address field now shows his Mumbai address. The previous Chennai address details will not be available. So, it will be difficult to find out exactly when he was living in Chennai and when he moved to Mumbai.

- John was born on April 3, 1992 in Chennai.
- His father registered his birth after three days on April 6, 1992.
- John did his entire schooling and college in Chennai.
- He got a job in Mumbai and shifted to Mumbai on June 21, 2015.
- He registered his change of address only on Jan 10, 2016.

Modeling Temporal Data: Example (3)

- Uni-Temporal Relation (Adding Valid Time To John's Data)

Name	City	Valid From	Valid Till
John	Chennai	April 3, 1992	June 20, 2015
John	Mumbai	June 21, 2015	∞

- John was born on April 3, 1992 in Chennai.
- His father registered his birth after three days on April 6, 1992.
- John did his entire schooling and college in Chennai.
- He got a job in Mumbai and shifted to Mumbai on June 21, 2015.
- He registered his change of address only on Jan 10, 2016.

- The valid time temporal database contents look like this:
  `Name, City, Valid From, Valid Till`
- Johns father registers his birth on 6th April 1992, a new database entry is made:
  `Person(John, Chennai, 3-Apr-1992,` ∞`).`
- On January 10, 2016 John reports his new address in Mumbai:
  `Person(John, Mumbai, 21-June-2015,` ∞`).`
  - The original entry is updated:
    `Person(John, Chennai, 3-Apr-1992, 20-June-2015).`

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Modeling Temporal Data: Example (4)

- Bi-Temporal Relation (John's Data Using Both Valid And Transaction Time)

Name	City	Valid From	Valid Till	Entered	Superseded
John	Chennai	April 3, 1992	June 20, 2015	April 6, 1992	Jan 10, 2016
John	Mumbai	June 21, 2015	∞	Jan 10, 2016	∞

- The database contents look like this:
  `Name, City, Valid From, Valid Till, Entered, Superseded`
- Johns father registers his birth on 6th April 1992:
  `Person(John, Chennai, 3-Apr-1992,` ∞`, 6-Apr-1992,` ∞`).`
- On January 10, 2016 John reports his new address in Mumbai:
  `Person(John, Mumbai, 21-June-2015,` ∞`, 10-Jan-2016,` ∞`).`
  - The original entry is updated as:
    `Person(John, Chennai, 3-Apr-1992, 20-June-2015, 6-Apr-1992 , 10-Jan-2016).`

- John was born on April 3, 1992 in Chennai.
- His father registered his birth after three days on April 6, 1992.
- John did his entire schooling and college in Chennai.
- He got a job in Mumbai and shifted to Mumbai on June 21, 2015.
- He registered his change of address only on Jan 10, 2016.

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Modeling Temporal Data: Summary

- **Advantages**
  - The main advantages of this bi-temporal relations is that it provides historical and roll back information.
    - Historical Information – Valid Time.
    - Rollback Information – Transaction Time.
  - *For example*, you can get the result for a query on John's history, like: Where did John live in the year 2001?. The result for this query can be got with the valid time entry. The transaction time entry is important to get the rollback information.
- **Disadvantages**
  - More storage
  - Complex query processing
  - Complex maintenance including backup and recovery

Source: https://www.mytecbits.com/oracle/oracle-database/what-is-temporal-database

Module Summary

- Discussed aspects of the database design process
- Studied the issues with temporal data

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".