[639] Data Modeling and Normalization

Meenakshi Syamkumar

Learning Objectives

Review	data modeling
Understand	normalization and normalization forms

Data Modeling

- Well-constructed data architectures must reflect the goals and business logic of the organization
- Deliberately choosing a coherent structure for data
- Lack of rigorous data modeling could create data swamps:
  - lots of redundant, mismatched, or simply wrong data

Data Model

- A data model represents the way data relates to the real world
- How should the data be structured and standardized to best reflect your organization’s processes, definitions, workflows, and logic
- A good data model captures how communication and work naturally flow within your organization
- A good data model should correlate with impactful business decisions
- A good data model contains consistent definitions

Conceptual, Logical, and Physical Data Models

- Conceptual
  - Contains business logic and rules
  - describes the system's data, such as schemas, tables, and fields (names and types)
  - Visualized typically using entity-relationship (ER) diagram
  - Example: encode the connections among customer ID, customer name, customer address, and customer orders
- Logical
  - Details how the conceptual model will be implemented in practice by adding significantly more detail
  - Example: add information on the types of customer ID, customer names, and custom addresses
- Physical
  - Defines how the logical model will be implemented in a database system
  - Example: add specific databases, schemas, and tables to our logical model, including configuration details

Normalization

Normalization

- Enforces strict control over the relationships of tables and columns within a database
- Goal of normalization is to remove the redundancy of data within a database and ensure referential integrity
- Objectives of normalization:
  - To free the collection of relations from undesirable insertion, update, and deletion dependencies
  - To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the lifespan of application programs
  - To make the relational model more informative to users
  - To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by

Normalization forms

- Denormalized
  - No normalization. Nested and redundant data is allowed.
- First normal form (1NF)
  - Each column is unique and has a single value. The table has a unique primary key.
- Second normal form (2NF)
  - The requirements of 1NF, plus partial dependencies are removed.
- Third normal form (3NF)
  - The requirements of 2NF, plus each table contains only relevant fields related to its primary key and has no transitive dependencies.

Database terminology

- Unique primary key
  - a single field or set of multiple fields that uniquely determines rows in the table
  - each key value occurs at most once
- Partial dependency
  - a subset of fields in a composite key can be used to determine a non-key column of the table
- Transitive dependency
  - when a non-key field depends on another non-key field

Denormalized form

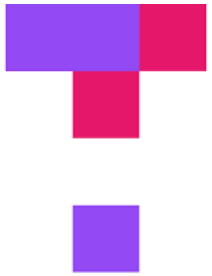
OrderID	OrderItems	CustomerID	CustomerName	OrderDate
100	[{ "sku": 1, "price": 50, "quantity": 1, "name:": "Thingamajig" }, { "sku": 2, "price": 25, "quantity": 2, "name:": "Whatchamacallit" }]	5	Joe Reis	2022-03-01

- *OrderDetail* table contains five fields
- The primary key is *OrderID*
- *OrderItems* field contains a nested object with two SKUs along with their price, quantity, and name.

1NF

OrderID	Sku	Price	Quantity	ProductName	CustomerID	CustomerName	OrderDate
100	1	50	1	Thingamajig	5	Joe Reis	2022-03-01
100	2	25	2	Whatchamacallit	5	Joe Reis	2022-03-01

- `OrderDetail` *without repeats or nested data*
- Do you see a problem with this version of the table? Let's answer this using



TOP HAT

1NF

OrderID	Sku	Price	Quantity	ProductName	CustomerID	CustomerName	OrderDate
100	1	50	1	Thingamajig	5	Joe Reis	2022-03-01
100	2	25	2	Whatchamacallit	5	Joe Reis	2022-03-01
101	3	75	1	Whozeewhatzit	7	Matt Housley	2022-03-01
102	1	50	1	Thingamajig	7	Matt Housley	2022-03-01

- Larger sample of *`OrderDetail`*

1NF: creating unique primary (composite) key

Order ID	LineItem Number	Sku	Price	Quantity	Product Name	Customer ID	Customer Name	OrderDate
100	1	1	50	1	Thingama jig	5	Joe Reis	2022-03-01
100	2	2	25	2	Whatchama callit	5	Joe Reis	2022-03-01
101	1	3	75	1	Whozee whatzit	7	Matt Housley	2022-03-01
102	1	1	50	1	Thingama jig	7	Matt Housley	2022-03-01

- Numbering the lines in each order by adding a column called *LineItemNumber*
- The composite key *(OrderID, LineItemNumber)* is now a unique primary key.

2NF

- A partial dependency is a non-key column that is fully determined by a subset of the columns in the unique primary (composite) key
- Partial dependencies can occur only when the primary key is composite
- In our example, the last three columns are determined by *OrderID*
  - *OrderID* → *CustomerID*
  - *OrderID* → *CustomerName*
  - *OrderID* → *OrderDate*
- Solution: split the *OrderDetail* table into two tables

2NF

OrderID	CustomerID	CustomerName	OrderDate
100	5	Joe Reis	2022-03-01
101	7	Matt Housley	2022-03-01
102	7	Matt Housley	2022-03-01

- *OrderID* is a primary key for *Orders*.

OrderID	LineItemNumber	Sku	Price	Quantity	ProductName
100	1	1	50	1	Thingamajig
100	2	2	25	2	Whatchamacallit
101	1	3	75	1	Whozeewhatzit
102	1	1	50	1	Thingamajig

- The composite key (*OrderID*, *LineItemNumber*) is a unique primary key for *OrderLineItem*
- Problem? `Sku` determines `ProductName` in `OrderLineItem`

3NF: OrderLineItem table

- Transitive dependency: *Sku* depends on the composite key, and *ProductName* depends on *Sku*
  - *OrderID* → *Sku*
  - *Sku* → *ProductName*
- Solution: break down *OrderLineItem* into *OrderLineItem* and *Skus*

OrderID	LineItemNumber	Sku	Price	Quantity
100	1	1	50	1
100	2	2	25	2
101	1	3	75	1
102	1	1	50	1

Sku	ProductName
1	Thingamajig
2	Whatchamacallit
3	Whozeewhatzit