

# Star and snowflake schema

DATABASE DESIGN



Lis Sulmont  
Curriculum Manager

# Star schema

## Dimensional modeling: star schema

### Fact tables

- Holds records of a metric
- Changes regularly
- Connects to dimensions via foreign keys

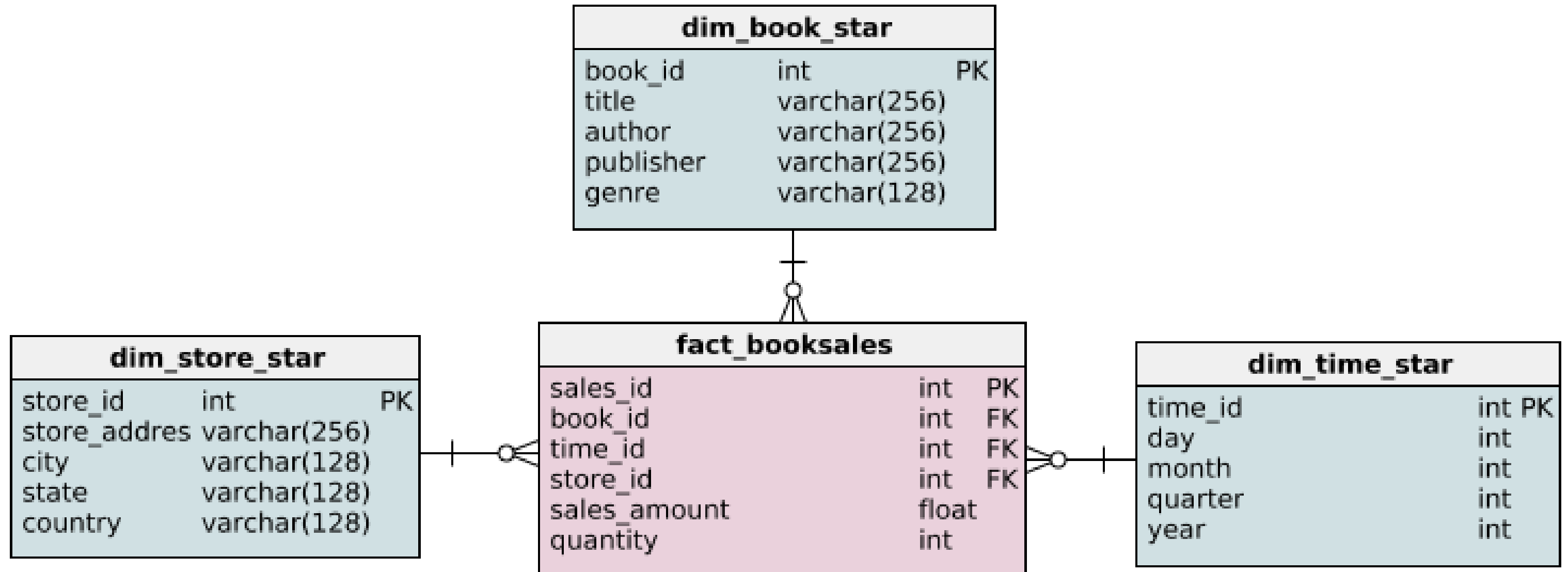
### Dimension tables

- Holds descriptions of attributes
- Does not change as often

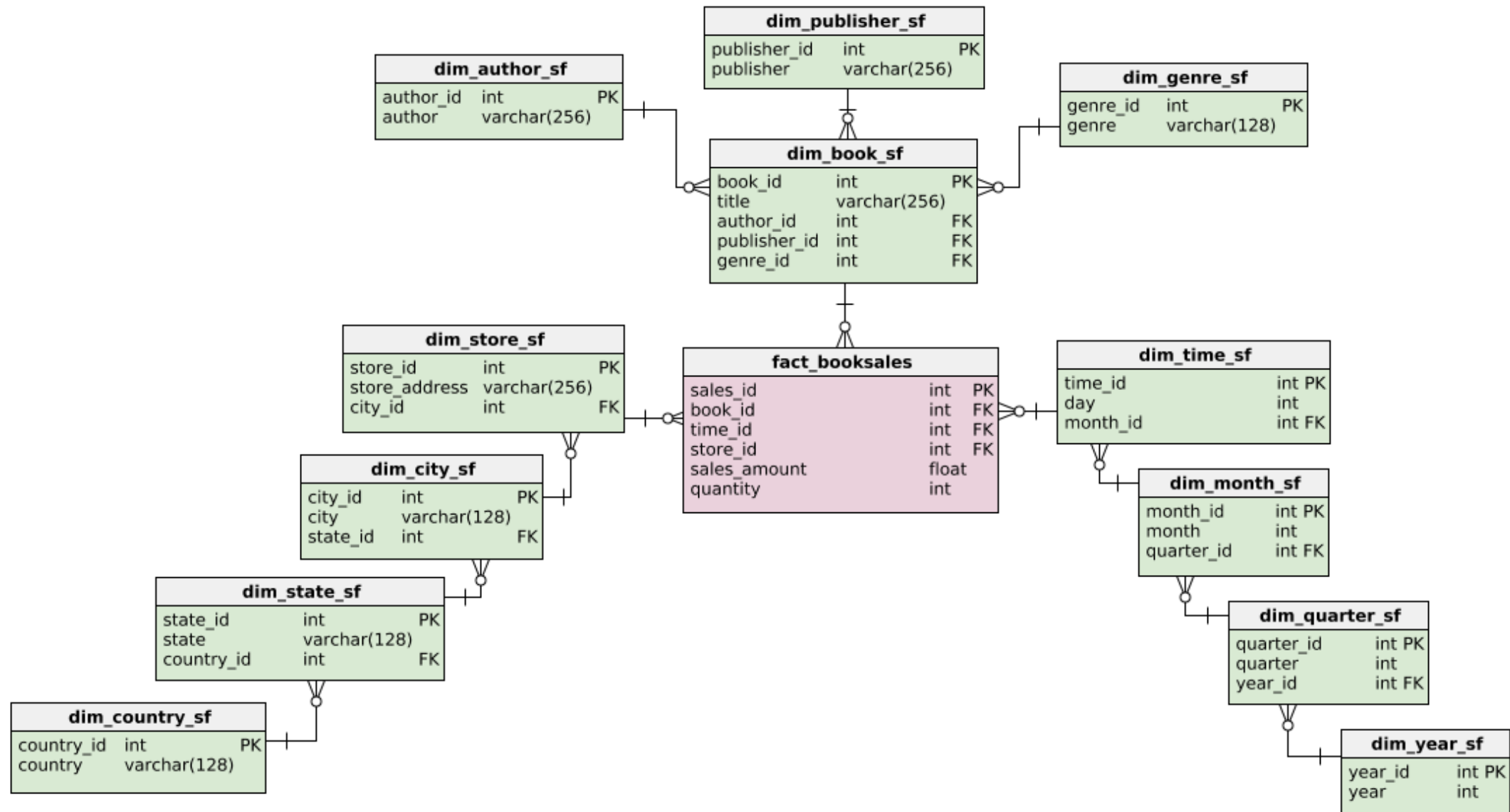
### Example:

- Supply books to stores in USA and Canada
- Keep track of book sales

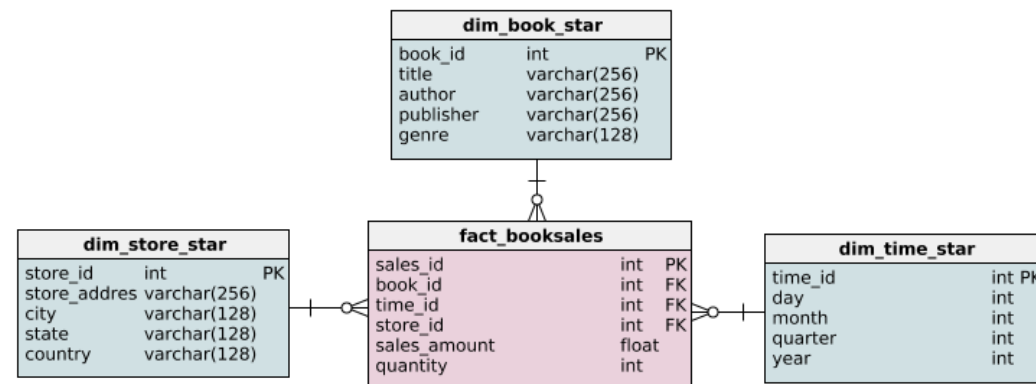
# Star schema example



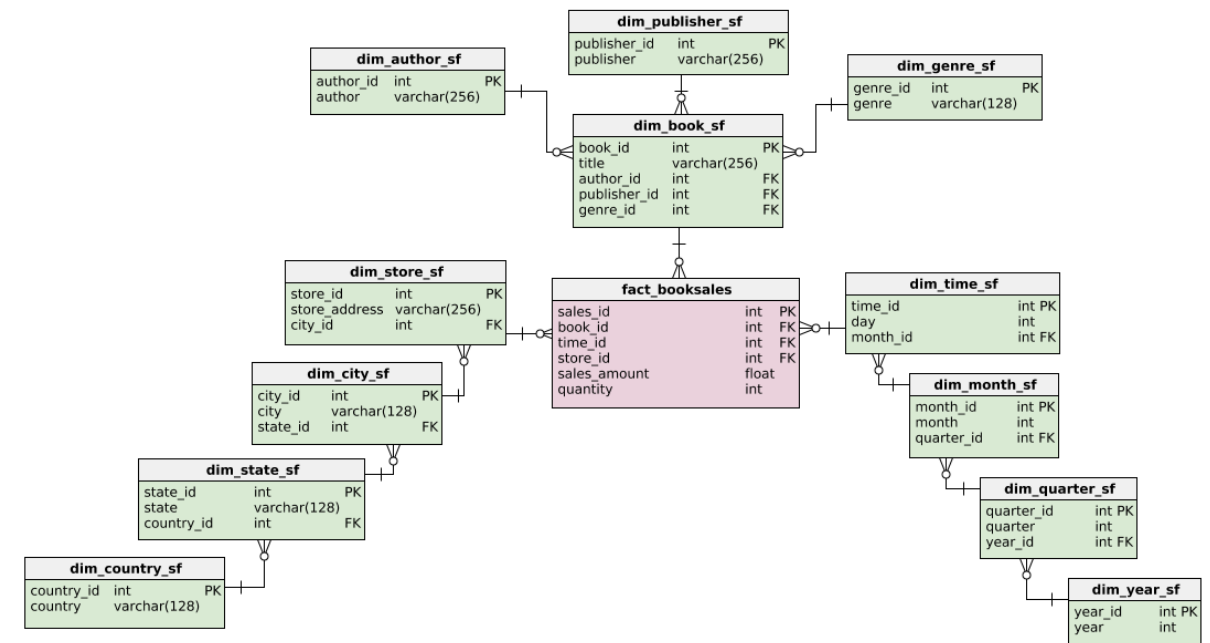
# Snowflake schema (an extension)



# Same fact table, different dimensions



Star schemas: one dimension



Snowflake schemas: more than one dimension

Because dimension tables are *normalized*

# What is normalization?

- Database design technique
- Divides tables into smaller tables and connects them via relationships
- **Goal:** reduce redundancy and increase data integrity

# What is normalization?

- Database design technique
- Divides tables into smaller tables and connects them via relationships
- **Goal:** reduce redundancy and increase data integrity

**Identify repeating groups of data and create new tables for them**

# Book dimension of the star schema

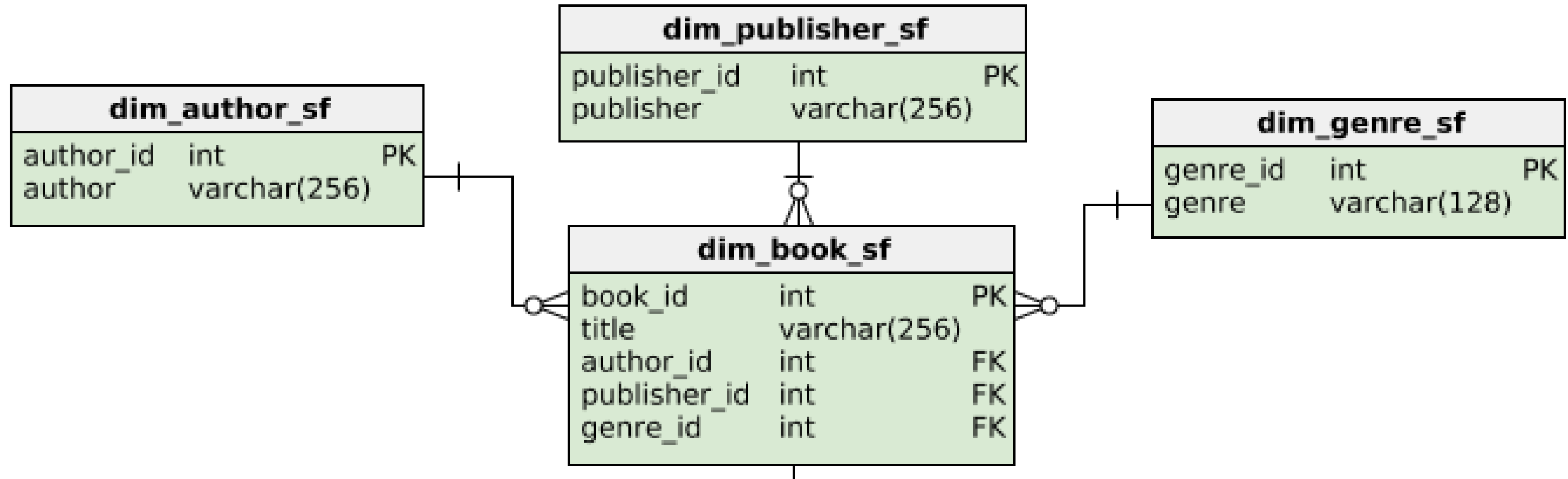
dim_book_star		
book_id	int	PK
title	varchar(256)	
author	varchar(256)	
publisher	varchar(256)	
genre	varchar(128)	

Most likely to have repeating values:

- Author
- Publisher
- Genre



# Book dimension of the snowflake schema

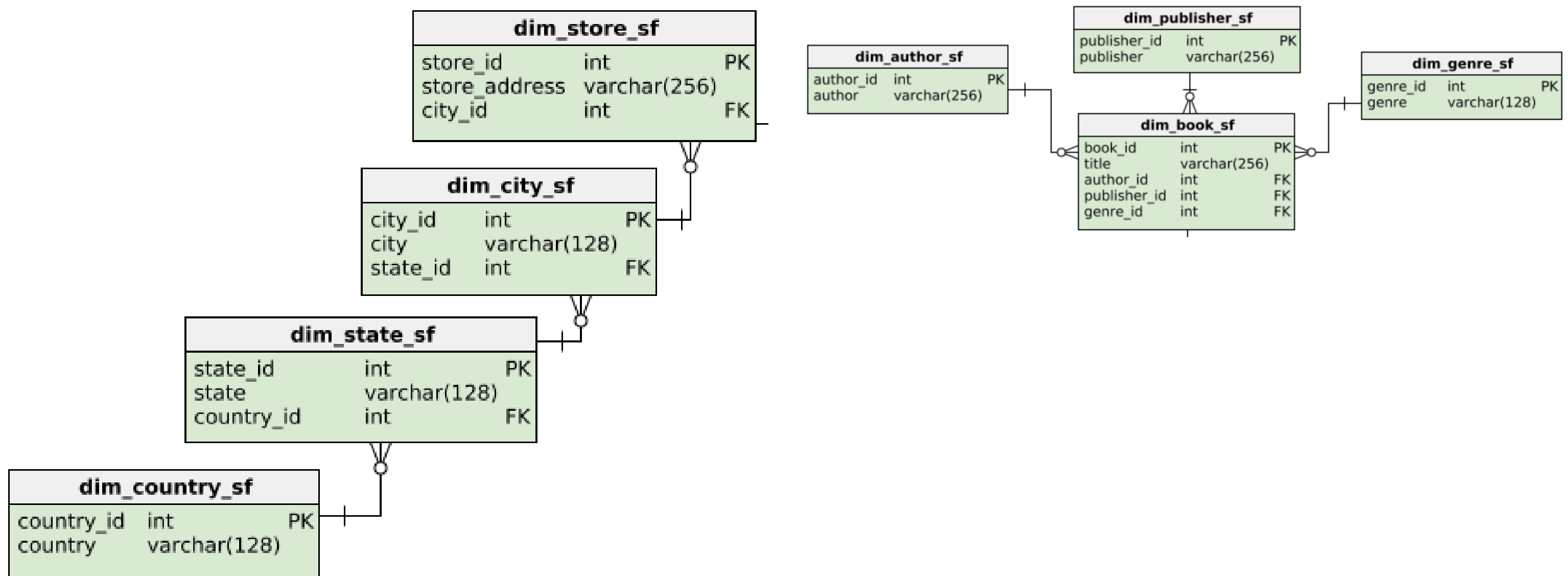


# Store dimension of the star schema

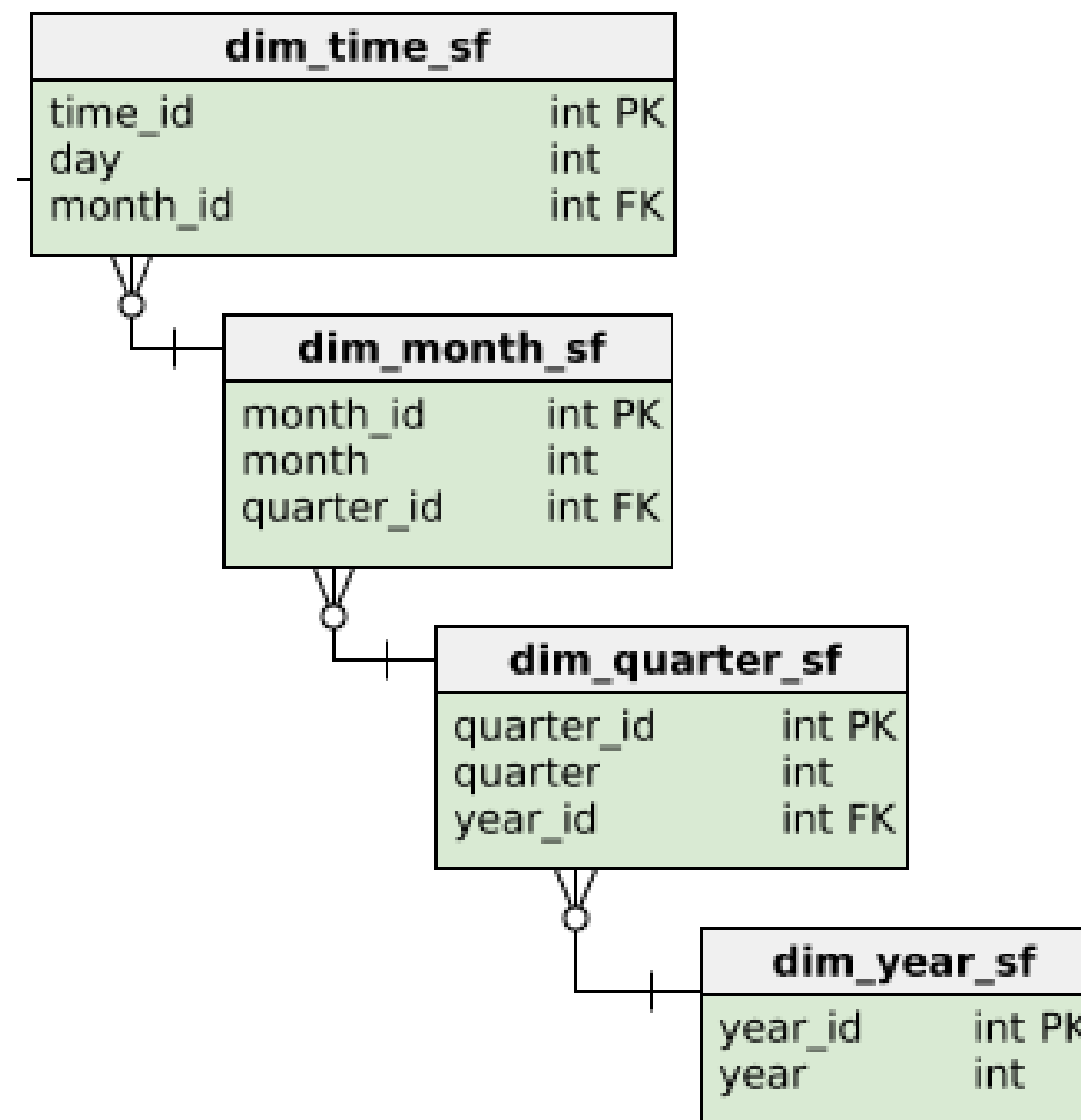
dim_store_star		
store_id	int	PK
store_addres	varchar(256)	
city	varchar(128)	
state	varchar(128)	
country	varchar(128)	

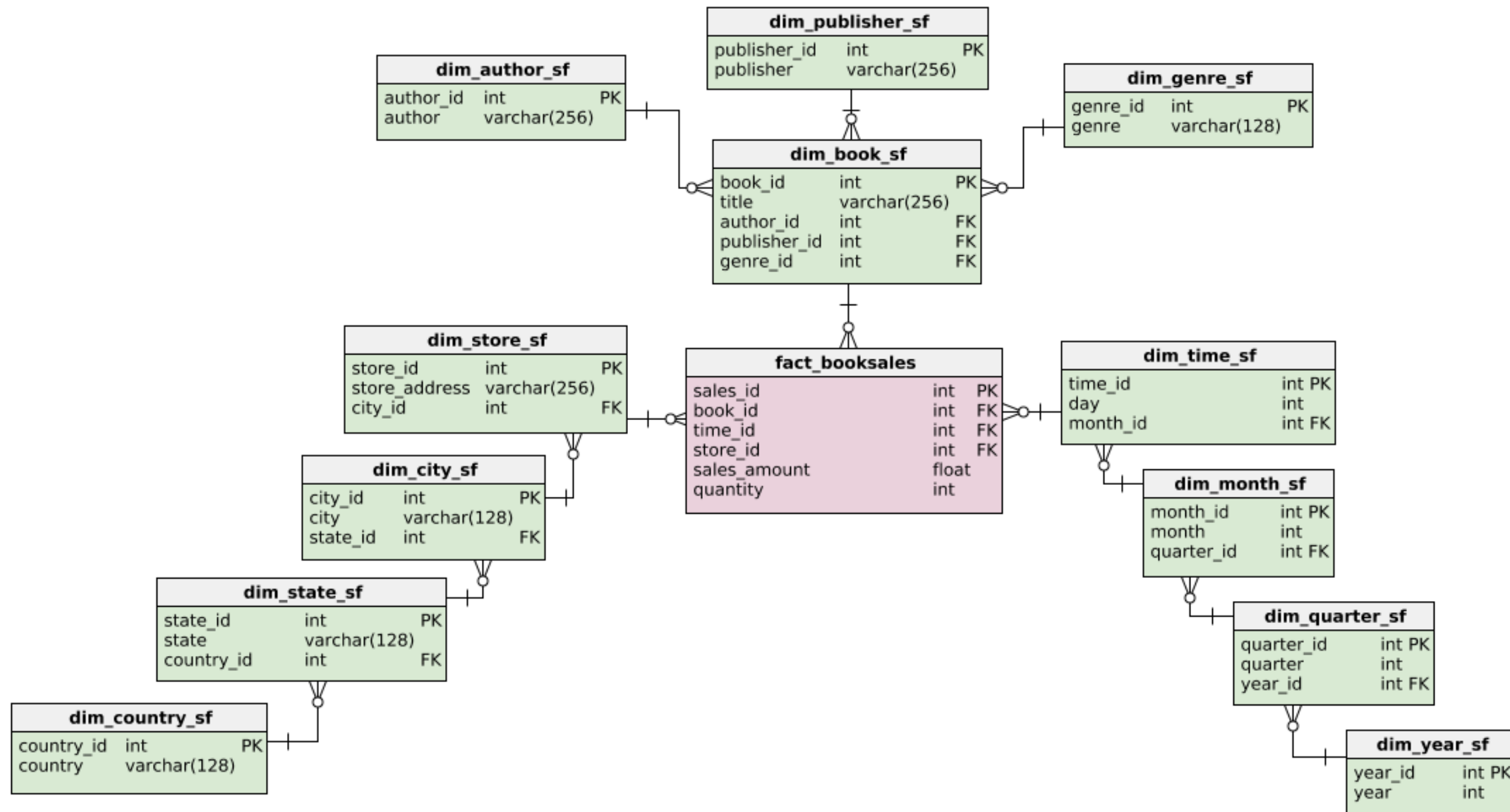
- City
- State
- Country

# Store dimension of the snowflake schema



dim_time_star	
time_id	int PK
day	int
month	int
quarter	int
year	int





# Let's practice!

DATABASE DESIGN

# Normalized and denormalized databases

DATABASE DESIGN

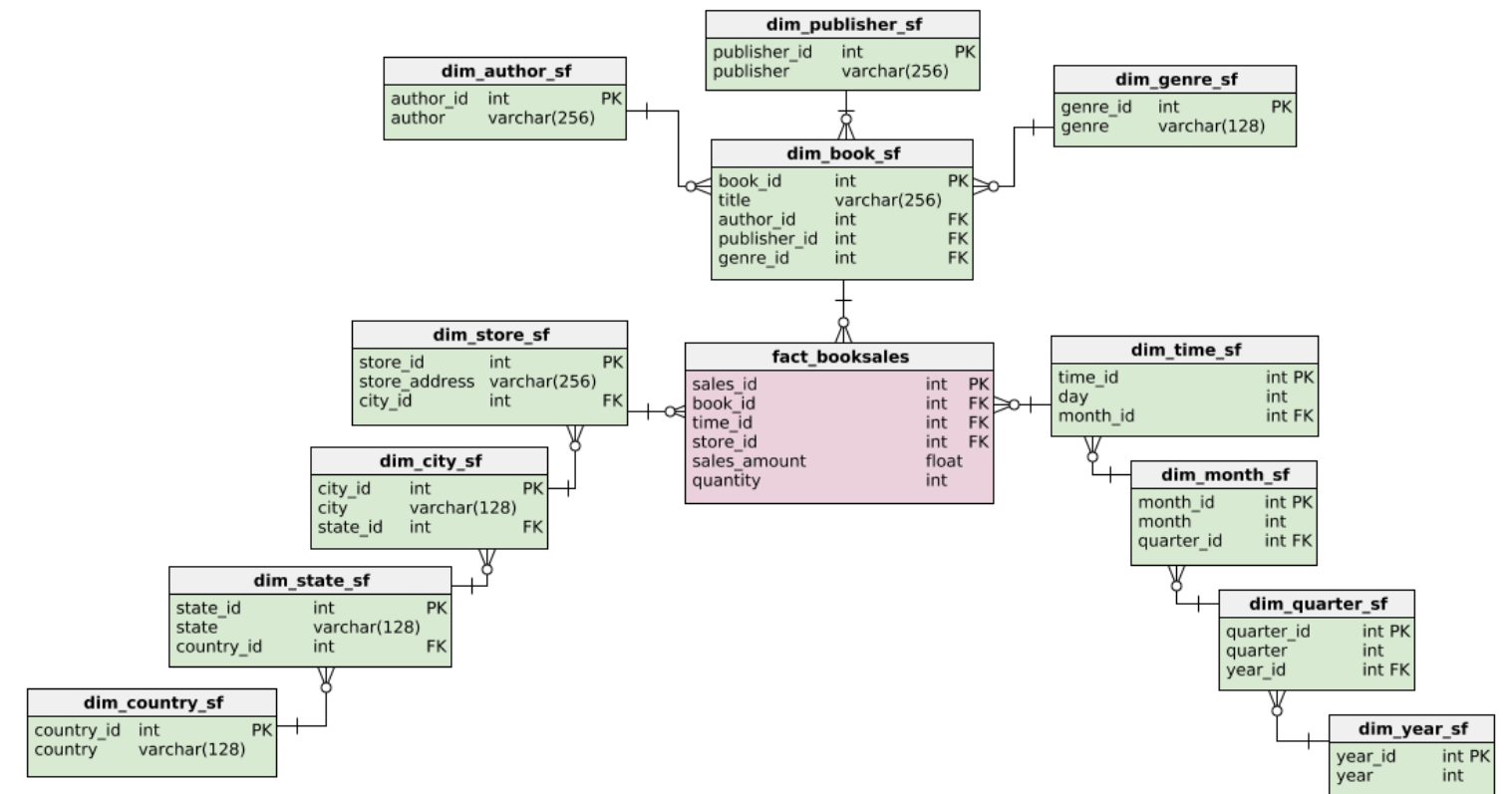
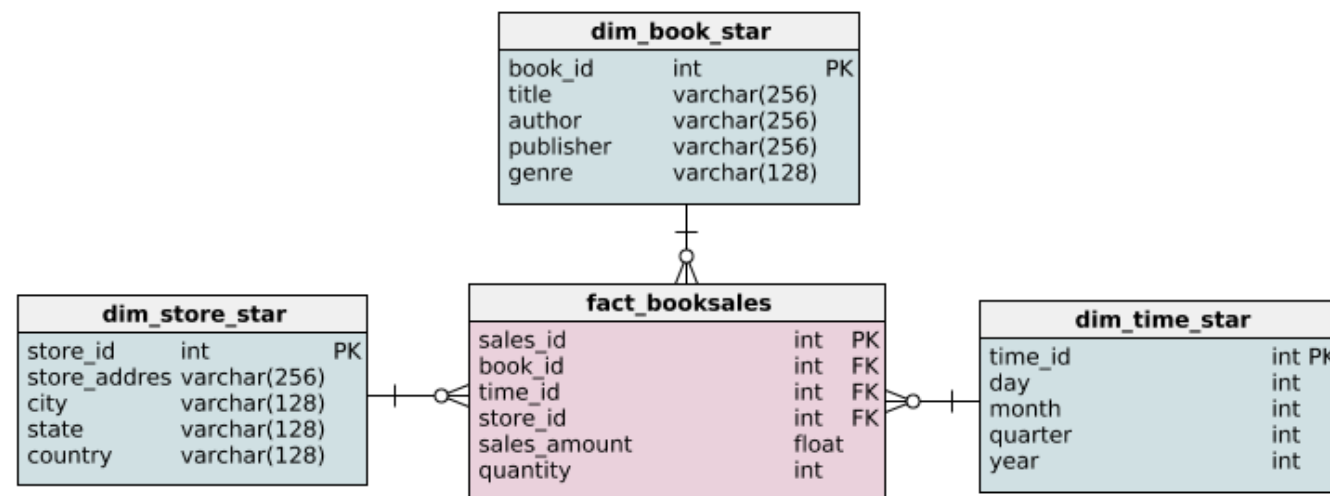


Lis Sulmont  
Curriculum Manager

# Back to our book store example

## Denormalized: star schema

## Normalized: snowflake schema





# Denormalized Query

Goal: get quantity of all Octavia E. Butler books sold in Vancouver in Q4 of 2018

```
SELECT SUM(quantity) FROM fact_booksales
-- Join to get city
INNER JOIN dim_store_star on fact_booksales.store_id = dim_store_star.store_id
-- Join to get author
INNER JOIN dim_book_star on fact_booksales.book_id = dim_book_star.book_id
-- Join to get year and quarter
INNER JOIN dim_time_star on fact_booksales.time_id = dim_time_star.time_id
WHERE
dim_store_star.city = 'Vancouver' AND dim_book_star.author = 'Octavia E. Butler' AND
dim_time_star.year = 2018 AND dim_time_star.quarter = 4;
```

7600

Total of 3 joins

# Normalized query

```
SELECT
  SUM(fact_booksales.quantity)
FROM
  fact_booksales
  -- Join to get city
  INNER JOIN dim_store_sf ON fact_booksales.store_id = dim_store_sf.store_id
  INNER JOIN dim_city ON dim_store_sf.city_id = dim_city_sf.city_id
  -- Join to get author
  INNER JOIN dim_book_sf ON fact_booksales.book_id = dim_book_sf.book_id
  INNER JOIN dim_author_sf ON dim_book_sf.author_id = dim_author_sf.author_id
  -- Join to get year and quarter
  INNER JOIN dim_time_sf ON fact_booksales.time_id = dim_time_sf.time_id
  INNER JOIN dim_month_sf ON dim_time_sf.month_id = dim_month_sf.month_id
  INNER JOIN dim_quarter_sf ON dim_month_sf.quarter_id = dim_quarter_sf.quarter_id
  INNER JOIN dim_year_sf ON dim_quarter_sf.year_id = dim_year_sf.year_id
```

## Normalized query (continued)

```
WHERE
  dim_city_sf.city = `Vancouver`
AND
  dim_author_sf.author = `Octavia E. Butler`
AND
  dim_year_sf.year = 2018 AND dim_quarter_sf.quarter = 4;
```

```
sum
7600
```

Total of 8 joins

*So, why would we want to normalize a databases?*

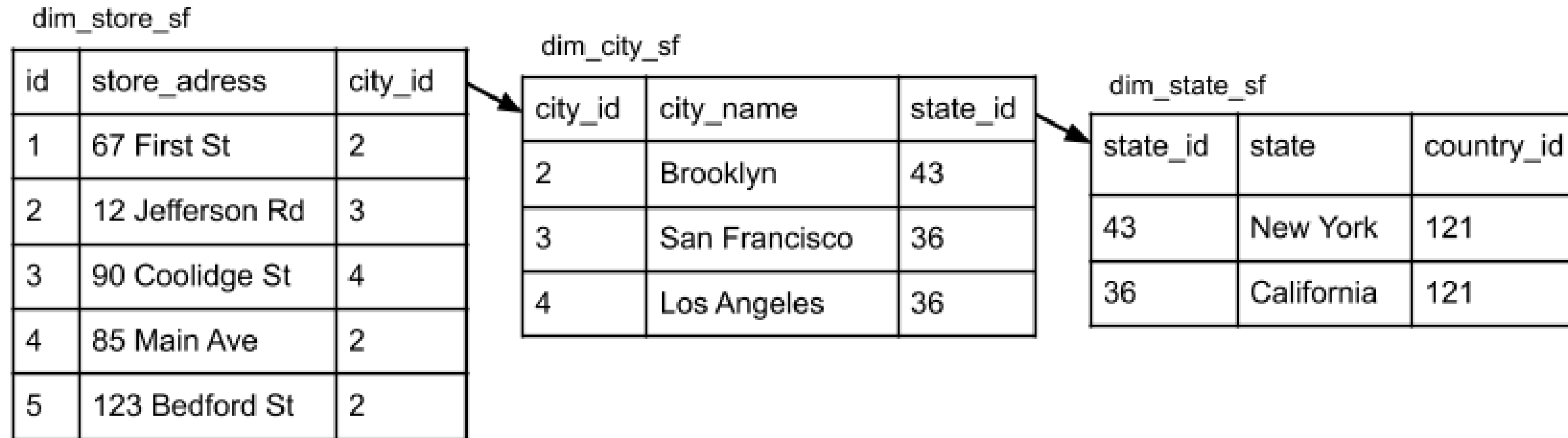
# Normalization saves space

dim\_store\_star

id	store_adress	city	state	country
1	67 First St	Brooklyn	New York	USA
2	12 Jefferson Rd	San Francisco	California	USA
3	90 Coolidge St	Los Angeles	California	USA
4	85 Main Ave	Brooklyn	New York	USA
5	123 Bedford St	Brooklyn	New York	USA

Denormalized databases enable data redundancy

# Normalization saves space



Normalization eliminates data redundancy

# Normalization ensures better data integrity

## 1. Enforces data consistency

Must respect naming conventions because of referential integrity, e.g., 'California', not 'CA' or 'california'

## 2. Safer updating, removing, and inserting

Less data redundancy = less records to alter

## 3. Easier to redesign by extending

Smaller tables are easier to extend than larger tables

# Database normalization

## Advantages

- Normalization eliminates data redundancy: save on storage
- Better data integrity: accurate and consistent data

## Disadvantages

- Complex queries require more CPU

# Remember OLTP and OLAP?

## OLTP

e.g., Operational databases

Typically highly normalized

- Write-intensive
- Prioritize quicker and safer insertion of data

## OLAP

e.g., Data warehouses

Typically less normalized

- Read-intensive
- Prioritize quicker queries for analytics



# Let's practice!

DATABASE DESIGN

# Normal forms

DATABASE DESIGN



Lis Sulmont  
Curriculum Manager

# Normalization

Identify repeating groups of data and create new tables for them

A more formal definition:

The goals of normalization are to:

- Be able to characterize the level of redundancy in a relational schema
- Provide mechanisms for transforming schemas in order to remove redundancy

<sup>1</sup> [https://opentextbc.ca/dbdesign01/chapter/chapter <sup>2</sup> 12 <sup>3</sup> normalization/](https://opentextbc.ca/dbdesign01/chapter/chapter%2012%20normalization/)

# Normal forms (NF)

*Ordered from least to most normalized:*

- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)
- Elementary key normal form (EKNF)
- Boyce-Codd normal form (BCNF)
- Fourth normal form (4NF)
- Essential tuple normal form (ETNF)
- Fifth normal form (5NF)
- Domain-key Normal Form (DKNF)
- Sixth normal form (6NF)

<sup>1</sup> [https://en.wikipedia.org/wiki/Database\\_normalization#Normal\\_forms](https://en.wikipedia.org/wiki/Database_normalization#Normal_forms)

# 1NF rules

- Each record must be unique - no duplicate rows
- Each cell must hold one value

## Initial data

```
| Student_id | Student_Email | Courses_Completed |
|-----|-----|-----|
| 235      | jim@gmail.com | Introduction to Python, Intermediate Python |
| 455      | kelly@yahoo.com | Cleaning Data in R |
| 767      | amy@hotmail.com | Machine Learning Toolbox, Deep Learning in Python |
```

# In 1NF form

Student_id	Student_Email
-----	-----
235	jim@gmail.com
455	kelly@yahoo.com
767	amy@hotmail.com

Student_id	Completed
-----	-----
235	Introduction to Python
235	Intermediate Python
455	Cleaning Data in R
767	Machine Learning Toolbox
767	Deep Learning in Python

# 2NF

- Must satisfy 1NF **AND**
  - If primary key is one column
    - then automatically satisfies 2NF
  - If there is a composite primary key
    - then each non-key column must be dependent on all the keys

## Initial data

Student_id (PK)	Course_id (PK)	Instructor_id	Instructor	Progress
235	2001	560	Nick Carchedi	.55
455	2345	658	Ginger Grant	.10
767	6584	999	Chester Ismay	1.00

# In 2NF form

Student_id (PK)	Course_id (PK)	Percent_Completed
-----	-----	-----
235	2001	.55
455	2345	.10
767	6584	1.00

Course_id (PK)	Instructor_id	Instructor
-----	-----	-----
2001	560	Nick Carchedi
2345	658	Ginger Grant
6584	999	Chester Ismay



# 3NF

- Satisfies 2NF
- No **transitive dependencies**: non-key columns can't depend on other non-key columns

## Initial Data

Course_id (PK)	Instructor_id	Instructor	Tech
-----	-----	-----	-----
2001	560	Nick Carchedi	Python
2345	658	Ginger Grant	SQL
6584	999	Chester Ismay	R

# In 3NF

Course_id (PK)	Instructor	Tech
-----	-----	-----
2001	Nick Carchedi	Python
2345	Ginger Grant	SQL
6584	Chester Ismay	R

Instructor_id	Instructor
-----	-----
560	Nick Carchedi
658	Ginger Grant
999	Chester Ismay

# Data anomalies

*What is risked if we don't normalize enough?*

1. Update anomaly
2. Insertion anomaly
3. Deletion anomaly

# Update anomaly

Data inconsistency caused by data redundancy when updating

Student_ID	Student_Email	Enrolled_in	Taught_by
230	lisa@gmail.com	Cleaning Data in R	Nick Carchedi
367	bob@hotmail.com	Data Visualization in R	Ronald Pearson
520	ken@yahoo.com	Introduction to Python	Hugo Bowne-Anderson
520	ken@yahoo.com	Arima Modeling in R	David Stoffer

To update student 520 's email:

- Need to update more than one record, otherwise, there will be inconsistency
- User updating needs to know about redundancy

# Insertion anomaly

Unable to add a record due to missing attributes

Student_ID	Student_Email	Enrolled_in	Taught_by
230	lisa@gmail.com	Cleaning Data in R	Nick Carchedi
367	bob@hotmail.com	Data Visualization in R	Ronald Pearson
520	ken@yahoo.com	Introduction to Python	Hugo Bowne-Anderson
520	ken@yahoo.com	Arima Modeling in R	David Stoffer

Unable to insert a student who has signed up but not enrolled in any courses

# Deletion anomaly

Deletion of record(s) causes unintentional loss of data

Student_ID	Student_Email	Enrolled_in	Taught_by
230	lisa@gmail.com	Cleaning Data in R	Nick Carchedi
367	bob@hotmail.com	Data Visualization in R	Ronald Pearson
520	ken@yahoo.com	Introduction to Python	Hugo Bowne-Anderson
520	ken@yahoo.com	Arima Modeling in R	David Stoffer

If we delete Student `230` , what happens to the data on `Cleaning Data in R` ?

# Data anomalies

*What is risked if we don't normalize enough?*

1. Update anomaly
2. Insertion anomaly
3. Deletion anomaly

The more normalized the database, the less prone it will be to data anomalies

*Don't forget the downsides of normalization from the last video*

# Let's practice

DATABASE DESIGN