

# Module\_3\_duckdb-key

Elyse, Jeanne & Sheila

## Table of contents

Learning Objectives (Module 3) . . . . .	2
The Mindset Shift: From Files to Database . . . . .	2
Arrow vs. DuckDB: When to Use What? . . . . .	2
Key Concept: In-Process Analytics Database . . . . .	3
<b>Rule of thumb:</b> . . . . .	3
DuckDB Setup . . . . .	3
Load Your Enhanced Toolkit . . . . .	3
Create Dataset Reference . . . . .	5
Check Our Data Setup . . . . .	6
Load Our Parquet Data into DuckDB . . . . .	9
Understanding SQL vs. dplyr: Two Languages, Same Ideas (10 minutes) . . .	10
The Great News: You Already Know the Concepts! . . . . .	10
Do We Have to Use SQL? . . . . .	10
Basic SQL Operations (Just for Understanding) . . . . .	11
The Magic: <code>show_query()</code> - See What dplyr Creates . . . . .	14
Why This Matters for Big Data . . . . .	15
Strategic <code>mutate()</code> Placement in Database Workflows (5 minutes) . . . . .	16
The Golden Rules for <code>mutate()</code> with Databases . . . . .	16
Quick Decision Framework . . . . .	18
Superpower 1: Lightning-Fast Aggregations . . . . .	18
Superpower 2: Window Functions Made Easy . . . . .	18
Superpower 3: SQL and dplyr Interchangeability . . . . .	21
Real-World Analytics Pipeline (25 minutes) . . . . .	24
Exercise 1: Multi-Table Analysis (10 minutes) . . . . .	24
<b>What is a JOIN? (Think of it like dating apps!)</b> . . . . .	29
Exercise 2: Understanding Joins with our data and duckdb (10 minutes) . . .	29
Choose Your DuckDB Adventure (15 minutes) . . . . .	32
Beginner Challenge: Popular Authors Analysis . . . . .	32
Intermediate Challenge: Popular Authors Analysis . . . . .	35

Cleanup: Closing Your Database Connection . . . . .	38
Key Takeaways from Module 3 . . . . .	39
What You've Accomplished Today . . . . .	39
Essential DuckDB Patterns to Remember . . . . .	40
Coming Up: Advanced Applications . . . . .	41
Quick Self-Assessment . . . . .	41

## Learning Objectives (Module 3)

By the end of this module, you will be able to:

- **Understand** when to choose DuckDB over Arrow for analytical workloads
- **Connect** R to DuckDB databases for high-performance analytics
- **Execute** complex joins and window functions on large datasets
- **Leverage** SQL and dplyr interchangeably for data analysis
- **Optimize** analytical queries for multi-gigabyte datasets
- **Build** persistent analytical databases for reproducible research

## The Mindset Shift: From Files to Database

### Arrow vs. DuckDB: When to Use What?

**Arrow thinking** (Module 2):

```
# Great for: Reading, filtering, format conversion
open_dataset("file.parquet") |>
  filter(year == 2023) |>
  collect()
```

**DuckDB thinking** (Module 3):

```
# Great for: Complex analytics, joins, aggregations
#con <- dbConnect(duckdb())
#seattle_tbl <- tbl(con, "seattle_checkouts")

seattle_tbl |>
  window_rank(CheckoutYear, Checkouts) |>
  complex_join(other_table) |>
  collect()
```

## Key Concept: In-Process Analytics Database

DuckDB is like having a **miniature data warehouse** running inside R:

- **In-process:** Runs directly in your R session (no separate server needed)
- **Columnar:** Optimized for analytical queries (fast aggregations)
- **SQL + dplyr:** Speaks both languages fluently
- **Persistent:** Can save your work and reuse it later

Think of it like:

- **Arrow:** “A really smart file reader”
- **DuckDB:** “A personal analytics database”

## Rule of thumb:

- **Simple filtering/reading:** Use Arrow
- **Complex analytics/joins:** Use DuckDB
- **Multiple related tables:** Definitely DuckDB
- **Window functions/advanced SQL:** DuckDB shines

## DuckDB Setup

### Load Your Enhanced Toolkit

```
# Enhanced package loading for database analytics
required_packages <- c("tidyverse", "arrow", "duckdb", "DBI", "dbplyr", "glue")

# Install missing packages
for (pkg in required_packages) {
  if (!requireNamespace(pkg, quietly = TRUE)) {
    install.packages(pkg)
  }
}

# Load all packages
for (pkg in required_packages) {
```

```
library(pkg, character.only = TRUE)
}
```

Warning: package 'tidyverse' was built under R version 4.3.3

Warning: package 'tibble' was built under R version 4.3.3

Warning: package 'tidyr' was built under R version 4.3.3

Warning: package 'readr' was built under R version 4.3.3

Warning: package 'purrr' was built under R version 4.3.3

Warning: package 'dplyr' was built under R version 4.3.3

Warning: package 'stringr' was built under R version 4.3.3

Warning: package 'forcats' was built under R version 4.3.3

Warning: package 'lubridate' was built under R version 4.3.3

-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --

v dplyr 1.1.4 v readr 2.1.5

v forcats 1.0.0 v stringr 1.5.1

v ggplot2 3.5.2 v tibble 3.2.1

v lubridate 1.9.3 v tidyr 1.3.1

v purrr 1.0.2

-- Conflicts ----- tidyverse\_conflicts() --

x dplyr::filter() masks stats::filter()

x dplyr::lag() masks stats::lag()

i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become

Warning: package 'arrow' was built under R version 4.3.3

Attaching package: 'arrow'

The following object is masked from 'package:lubridate':

duration

The following object is masked from 'package:utils':

timestamp

Warning: package 'duckdb' was built under R version 4.3.3

Loading required package: DBI

Warning: package 'DBI' was built under R version 4.3.3

Warning: package 'dbplyr' was built under R version 4.3.3

Attaching package: 'dbplyr'

The following objects are masked from 'package:dplyr':

ident, sql

Warning: package 'glue' was built under R version 4.3.3

## Create Dataset Reference

Make sure to get your referenced .csv with `open_dataset()`. Similar to how you would use `read_csv()` but we are not reading in a file:

```
seattle_csv <- open_dataset(  
  sources = "data/seattle-library-checkouts.csv",  
  col_types = schema(ISBN = string()),  
  format = "csv"  
)
```

## Check Our Data Setup

Let's verify we have our data files from Module 2:

```
# Check if we have our CSV file
csv_exists <- file.exists("data/seattle-library-checkouts.csv")
parquet_exists <- dir.exists("data/parquet/seattle")

glue("CSV file exists: {csv_exists}")
```

CSV file exists: TRUE

```
glue("Parquet files exist: {parquet_exists}")
```

Parquet files exist: TRUE

```
if (csv_exists) {
  file_size_gb <- file.size("data/seattle-library-checkouts.csv") / (1024^3)
  glue("CSV file size: {round(file_size_gb, 2)} GB")
}
```

CSV file size: 8.58 GB

**Note:** We already created Parquet files in Module 2, but let's review the two approaches you can use:

**Method 1: Simple Parquet** - This is what we created in Module 2:

```
# Method 1: Simple Parquet (single folder, multiple files)
# Good for: Smaller datasets, simple analytics

# Save as regular Parquet files
#write_dataset(
#  seattle_csv,
#  path = "data/parquet/seattle_parquet",
#  format = "parquet"
#)
```

**Method 2: Partitioned Parquet** - An alternative approach for time-series data:

```
# Method 2: Partitioned Parquet (organized by year)
# Good for: Large datasets, year-based filtering
write_dataset(
  seattle_csv,
  path = "data/parquet/seattle_by_year",
  format = "parquet",
  partitioning = "CheckoutYear" # Creates a folder for each year
)
```

Just for fun lets see compare the file sizes...

```
# Compare file sizes across all three methods

# Original CSV
csv_size_bytes <- file.size("data/seattle-library-checkouts.csv")
csv_size_gb <- csv_size_bytes / (1024^3)
glue("Original CSV size: {round(csv_size_gb, 2)} GB")
```

Original CSV size: 8.58 GB

```
# Simple parquet (from Method 1)
simple_parquet_bytes <- sum(file.size(list.files("data/parquet/seattle_parquet/",
                                                full.names = TRUE, recursive = TRUE)))
simple_parquet_gb <- simple_parquet_bytes / (1024^3)
glue("Simple parquet size: {round(simple_parquet_gb, 2)} GB")
```

Simple parquet size: 4.11 GB

```
# Partitioned parquet (from Method 2)
partitioned_parquet_bytes <- sum(file.size(list.files("data/parquet/seattle_by_year/",
                                                      full.names = TRUE, recursive = TRUE)))
partitioned_parquet_gb <- partitioned_parquet_bytes / (1024^3)
glue("Partitioned parquet size: {round(partitioned_parquet_gb, 2)} GB")
```

Partitioned parquet size: 4.11 GB

Approach	Best For	Key Benefits	Trade-offs
<b>Simple Parquet</b>	General analytics, exploratory work, datasets < 5GB	→ <b>Simpler file structure</b> - easier to understand and manage → <b>Faster initial setup</b> - no need to think about partitioning → <b>Universal compatibility</b> - works with any analysis tool	→ Slower filtering by year/date → Must scan entire dataset for time-based queries
<b>Partitioned Parquet</b>	Time-series analysis, production pipelines, datasets > 5GB	→ <b>Lightning-fast date filtering</b> - only reads relevant year folders → <b>Organized structure</b> - easy to find specific time periods → <b>Scalable</b> - excellent performance even with massive datasets	→ More complex folder structure → Requires planning partitioning strategy → Can create many small files if over-partitioned

## Connect to DuckDB: Your Personal Data Warehouse

```
# Create a persistent DuckDB database
# (This is like opening a new Excel workbook, but for big data)
con <- dbConnect(duckdb::duckdb(), dbdir = "data/seattle.duckdb")

# Let's see what we're working with
glue(" Connected to DuckDB database at: data/seattle.duckdb")
```

Connected to DuckDB database at: data/seattle.duckdb

## What just happened?

- Created a **persistent** DuckDB database file
- Established a connection we can use throughout our session
- The database will save our work even after R shuts down



## Load Our Parquet Data into DuckDB

Now we'll create a table in our database using the Parquet files we created in Module 2:

```
# This reads from the Parquet files and creates a database table
# Instead of loading all the data, we'll create a "window" to look through
# This is called a VIEW - it lets us see the data without loading it all
dbExecute(con, "
  DROP TABLE IF EXISTS seattle_checkouts;
  CREATE TABLE seattle_checkouts AS
  SELECT *
  FROM read_parquet('data/parquet/seattle_parquet/*.parquet')
")
```

```
[1] 41389465
```

```
# Verify our data loaded correctly
record_count <- dbGetQuery(con, "SELECT COUNT(*) as total FROM seattle_checkouts")
glue(" Loaded {format(record_count$total, big.mark = ',')} records into DuckDB")
```

Loaded 41,389,465 records into DuckDB

```
# Create our dplyr reference to the table
seattle_tbl <- tbl(con, "seattle_checkouts")
```

## Quick Data Exploration

```
# Basic exploration using dplyr syntax
seattle_tbl |>
  summarise(
    total_records = n(),
    earliest_year = min(CheckoutYear, na.rm = TRUE),
    latest_year = max(CheckoutYear, na.rm = TRUE),
    total_checkouts = sum(Checkouts, na.rm = TRUE)
  ) |>
  collect()
```

```
# A tibble: 1 x 4
  total_records earliest_year latest_year total_checkouts
      <dbl>         <dbl>         <dbl>         <dbl>
1    41389465         2005         2022         144487078
```

```
# Peek at the structure
```

```
seattle_tbl |>
```

```
  head(5) |>
```

```
collect()
```

```
# A tibble: 5 x 12
```

	UsageClass	CheckoutType	MaterialType	CheckoutYear	CheckoutMonth	Checkouts
	<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	Physical	Horizon	BOOK	2016	6	1
2	Physical	Horizon	BOOK	2016	6	1
3	Digital	OverDrive	EBOOK	2016	6	1
4	Physical	Horizon	BOOK	2016	6	1
5	Physical	Horizon	SOUNDDISC	2016	6	1

```
# i 6 more variables: Title <chr>, ISBN <chr>, Creator <chr>, Subjects <chr>,  
#   Publisher <chr>, PublicationYear <chr>
```

### What's the difference from Arrow?

With Arrow, we were always “viewing” files. With DuckDB, we’ve **loaded** the data into a database structure optimized for fast analytics!

## Understanding SQL vs. dplyr: Two Languages, Same Ideas (10 minutes)

### The Great News: You Already Know the Concepts!

**This workshop is about using the dplyr that we know and love** - but understanding some SQL can help you see what’s happening behind the scenes. DuckDB speaks both languages fluently!

Your challenge: Create a new table called `seattle_by_year` using the partitioned files.

### Do We Have to Use SQL?

**Short answer: No!** This workshop focuses on **dplyr** because:

- You already know dplyr syntax
- It’s more intuitive and readable
- **dbplyr** automatically translates dplyr to SQL for you
- You can be productive immediately

But knowing some SQL helps because:

- You can see what dplyr is doing behind the scenes
- Some complex operations are cleaner in SQL
- You can mix both approaches when needed

## Basic SQL Operations (Just for Understanding)

### 1. Selecting Rows (WHERE = filter())

```
# dplyr approach (what we'll use most)
seattle_tbl |>
  filter(CheckoutYear == 2020) |>
  head() |>
  collect()
```

```
# A tibble: 6 x 12
```

	UsageClass	CheckoutType	MaterialType	CheckoutYear	CheckoutMonth	Checkouts
	<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	Physical	Horizon	BOOK	2020	1	2
2	Digital	OverDrive	EBOOK	2020	1	12
3	Digital	OverDrive	AUDIOBOOK	2020	5	1
4	Digital	OverDrive	EBOOK	2020	5	1
5	Digital	OverDrive	AUDIOBOOK	2020	2	4
6	Digital	OverDrive	EBOOK	2020	2	1

```
# i 6 more variables: Title <chr>, ISBN <chr>, Creator <chr>, Subjects <chr>,
#   Publisher <chr>, PublicationYear <chr>
```

```
# SQL equivalent (just for comparison)
dbGetQuery(con, "
  SELECT *
  FROM seattle_checkouts
  WHERE CheckoutYear = 2020
  LIMIT 6
")
```

	UsageClass	CheckoutType	MaterialType	CheckoutYear	CheckoutMonth	Checkouts
1	Physical	Horizon	BOOK	2020	1	2
2	Digital	OverDrive	EBOOK	2020	1	12
3	Digital	OverDrive	AUDIOBOOK	2020	5	1

4	Digital	OverDrive	EBOOK	2020	5	1
5	Digital	OverDrive	AUDIOBOOK	2020	2	4
6	Digital	OverDrive	EBOOK	2020	2	1
						Title
1	The dolphin : two versions, 1972-1973 / Robert Lowell ; edited by Saskia Hamilton.					
2	Dreaming of You: Gamblers Series, Book 2					
3	The New York Stories (Unabridged)					
4	Moving On					
5	The Talisman (Unabridged)					
6	Native Speaker					
ISBN		Creator		Subjects		
1	Lowell, Robert, 1917-1977,			Poetry		
2	Lisa Kleypas			Fiction, Romance		
3	John O'Hara			Fiction, Literature		
4	Anna Jacobs			Fiction, Literature		
5	Stephen King			Fiction, Horror		
6	Chang-Rae Lee			Fiction, Literature, Thriller		
		Publisher		PublicationYear		
1	Farrar, Straus and Giroux,			2019.		
2	HarperCollins Publishers Inc.			2005		
3	Books on Tape			2014		
4	Severn House Publishers Ltd			2012		
5	Simon & Schuster - Audiobooks			2014		
6	Penguin Group (USA), Inc.			2013		

## 2. Selecting Columns (SELECT = select())

```
# dplyr approach
seattle_tbl |>
  select(Title, Creator, CheckoutYear) |>
  head() |>
  collect()
```

```
# A tibble: 6 x 3
  Title                                Creator CheckoutYear
  <chr>                                <chr>         <dbl>
1 Super rich : a guide to having it all / Russell Simmons ~ Simmon~      2016
2 Shadowheart / James Barclay.         Barcla~      2016
3 Where I'm Reading From: The Changing World of Books      Tim Pa~      2016
4 Little spotted cat / by Alyssa Satin Capucilli ; illustr~ Capuci~      2016
5 Dog & butterfly [sound recording] / Heart.               Heart ~      2016
6 Precalculus the easy way / Larry S. Leff.                 Leff, ~      2016
```

```
# SQL equivalent
dbGetQuery(con, "
  SELECT Title, Creator, CheckoutYear
  FROM seattle_checkouts
  LIMIT 6
")
```

	Title
1	Super rich : a guide to having it all / Russell Simmons with Chris Morrow.
2	Shadowheart / James Barclay.
3	Where I'm Reading From: The Changing World of Books
4	Little spotted cat / by Alyssa Satin Capucilli ; illustrations by Dan Andreason.
5	Dog & butterfly [sound recording] / Heart.
6	Precalculus the easy way / Larry S. Leff.

	Creator	CheckoutYear
1	Simmons, Russell	2016
2	Barclay, James, 1965-	2016
3	Tim Parks	2016
4	Capucilli, Alyssa Satin, 1957-	2016
5	Heart (Musical group)	2016
6	Leff, Lawrence S.	2016

### 3. Grouping and Summarizing (GROUP BY = group\_by())

```
# dplyr approach (our preferred method)
seattle_tbl |>
  group_by(MaterialType) |>
  summarise(total_checkouts = sum(Checkouts)) |>
  arrange(desc(total_checkouts)) |>
  head() |>
  collect()
```

Warning: Missing values are always removed in SQL aggregation functions.  
 Use `na.rm = TRUE` to silence this warning  
 This warning is displayed once every 8 hours.

```
# A tibble: 6 x 2
  MaterialType total_checkouts
  <chr>         <dbl>
1 BOOK         64231952
2 VIDEODISC     31389390
```

3	EBOOK	18918056
4	SOUNDDISC	14579739
5	AUDIOBOOK	9936233
6	VIDEOCASS	1501066

```
# SQL equivalent
dbGetQuery(con, "
  SELECT MaterialType, SUM(Checkouts) as total_checkouts
  FROM seattle_checkouts
  GROUP BY MaterialType
  ORDER BY total_checkouts DESC
  LIMIT 6
")
```

	MaterialType	total_checkouts
1	BOOK	64231952
2	VIDEODISC	31389390
3	EBOOK	18918056
4	SOUNDDISC	14579739
5	AUDIOBOOK	9936233
6	VIDEOCASS	1501066

### The Magic: `show_query()` - See What dplyr Creates

The best part about **dbplyr** is you can see the SQL translation:

```
# Write dplyr code and see the SQL it generates
seattle_tbl |>
  filter(CheckoutYear >= 2020) |>
  group_by(MaterialType) |>
  summarise(
    total_checkouts = sum(Checkouts),
    avg_checkouts = mean(Checkouts)
  ) |>
  arrange(desc(total_checkouts)) |>
  show_query() # This shows you the SQL!
```

```
<SQL>
SELECT
  MaterialType,
  SUM(Checkouts) AS total_checkouts,
```

```

    AVG(Checkouts) AS avg_checkouts
FROM (
  SELECT seattle_checkouts.*
  FROM seattle_checkouts
  WHERE (CheckoutYear >= 2020.0)
) q01
GROUP BY MaterialType
ORDER BY total_checkouts DESC

```

### Key Takeaway:

- Use **dplyr** for your analysis (it's what this workshop teaches!)
- Use **show\_query()** when you're curious about the SQL
- Use **direct SQL** only when you need something dplyr can't do easily

### Why This Matters for Big Data

Both dplyr and SQL are **pushed down** to the database:

```

# This entire pipeline runs IN the database, not in R
result <- seattle_tbl |>                                # Connect to table
  filter(CheckoutYear >= 2020) |>                        # Database does the filtering
  group_by(MaterialType) |>                              # Database does the grouping
  summarise(total = sum(Checkouts)) |>                  # Database does the math
  arrange(desc(total))                                  # Database does the sorting
# Only when we add collect() does data come to R!

# See the result (still in database)
result

```

```

# Source:      SQL [?? x 2]
# Database:    DuckDB v1.1.3 [jmmccclu3@Windows 10 x64:R 4.3.2/C:\Users\jmmccclu3\Documents\usel
# Ordered by:  desc(total)
  MaterialType  total
  <chr>         <dbl>
1 EBOOK        7765062
2 BOOK          5939939
3 AUDIOBOOK    4806033
4 VIDEODISC     1327399
5 SOUNDDISC     446323

```

```

6 REGPRINT      51087
7 MIXED         44497
8 MUSIC         9043
9 LARGEPRINT    5267
10 VIDEO        4449
# i more rows

```

```

# Bring it to R memory
final_result <- result |> collect()

```

**The Big Idea:** Whether you write dplyr or SQL, the **database does the heavy lifting**, not R!

### Strategic mutate() Placement in Database Workflows (5 minutes)

When working with databases like DuckDB, **where** you place your mutate() operations can dramatically affect performance and functionality.

### The Golden Rules for mutate() with Databases

#### BEFORE collect() (Pushed to Database)

```

# These mutate operations happen IN the database (fast!)
seattle_tbl |>
  mutate(
    # Simple transformations - database can handle these
    checkout_decade = floor(CheckoutYear/10) * 10,
    is_recent = CheckoutYear >= 2020,
    title_length = LENGTH(Title) # SQL function
  ) |>
  filter(is_recent == TRUE) |> # Can filter on the new column!
  collect()

```

# A tibble: 6,076,919 x 15

	UsageClass	CheckoutType	MaterialType	CheckoutYear	CheckoutMonth	Checkouts
	<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	Digital	OverDrive	AUDIOBOOK	2022	9	1
2	Digital	OverDrive	EBOOK	2022	9	2
3	Digital	OverDrive	EBOOK	2022	9	3
4	Digital	OverDrive	EBOOK	2022	9	1
5	Digital	OverDrive	EBOOK	2022	9	1



6	Digital	OverDrive	AUDIOBOOK	2022	9	1
7	Physical	Horizon	BOOK	2022	9	2
8	Digital	OverDrive	AUDIOBOOK	2022	9	1
9	Physical	Horizon	VIDEODISC	2022	9	2
10	Digital	OverDrive	AUDIOBOOK	2022	9	2

```
# i 6,076,909 more rows
# i 9 more variables: Title <chr>, ISBN <chr>, Creator <chr>, Subjects <chr>,
#   Publisher <chr>, PublicationYear <chr>, checkout_decade <dbl>,
#   is_recent <lgl>, title_length <dbl>
```

AFTER collect() (In R Memory)

```
# These mutate operations happen in R (slower, but more flexible)
seattle_tbl |>
  collect() |> # Brings ALL data to R first!
  mutate(
    # Complex R-specific operations
    title_words = str_count(Title, "\\w+"),
    checkout_season = case_when(
      CheckoutMonth %in% c(12,1,2) ~ "Winter",
      CheckoutMonth %in% c(3,4,5) ~ "Spring",
      # ... more complex logic
    )
  )
```

```
# SLOW: Bring all data to R, then transform
system.time({
  slow_result <- seattle_tbl |>
    collect() |> # Loads millions of rows into R!
    mutate(checkout_decade = floor(CheckoutYear/10) * 10) |>
    filter(checkout_decade == 2020)
})

# FAST: Transform in database, then collect filtered results
system.time({
  fast_result <- seattle_tbl |>
    mutate(checkout_decade = floor(CheckoutYear/10) * 10) |>
    filter(checkout_decade == 2020) |>
    collect() # Only brings filtered results to R
})
```

When to Use Each Approach

Use <code>mutate()</code> BEFORE <code>collect()</code> when:	Use <code>mutate()</code> AFTER <code>collect()</code> when:
Simple math operations	Complex string manipulation with R functions
Date/time extraction	Custom R functions
Basic case_when logic	Advanced statistical calculations
You need to filter on the new column	Complex joins with R objects

### Quick Decision Framework

Ask yourself: “Can SQL do this operation?”

- **Yes** → `mutate()` before `collect()`
- **No** → `mutate()` after `collect()`
- **Not sure** → Try before first, move after if it fails!

This strategic placement is one of the key skills that separates good big data analysts from great ones!

### Superpower 1: Lightning-Fast Aggregations

Let’s compare the same analysis using DuckDB vs. what we might do with regular R

**Why this is impressive:**

- Processed millions of rows in seconds
- Calculated multiple statistics simultaneously
- All done in the database before bringing results to R

### Superpower 2: Window Functions Made Easy

Window functions let you do advanced analytics like rankings, running totals, and comparisons within groups:

```
# Find the top 3 most popular titles each year
top_titles_by_year <- seattle_tbl |>
  group_by(CheckoutYear, Title) |>
  summarise(total_checkouts = sum(Checkouts), .groups = "drop") |>
  group_by(CheckoutYear) |>
  window_order(desc(total_checkouts)) |>
```

```
mutate(rank = row_number()) |>
filter(rank <= 3) |>
arrange(CheckoutYear, rank) |>
collect()

# Display results
top_titles_by_year |>
head(15) # Show top 3 for first 5 years
```

# A tibble: 15 x 4

# Groups: CheckoutYear [5]

	CheckoutYear	Title	total_checkouts	rank
	<dbl>	<chr>	<dbl>	<dbl>
1	2005	<Unknown Title>	34189	1
2	2005	Greatest hits	1947	2
3	2005	Uncataloged Folder or Bag--DWN	1905	3
4	2006	<Unknown Title>	52396	1
5	2006	Greatest hits	3268	2
6	2006	Flightplan [videorecording] / Touchstone ~	2564	3
7	2007	<Unknown Title>	47460	1
8	2007	Greatest hits	3170	2
9	2007	Little Miss Sunshine [videorecording] / B~	2335	3
10	2008	<Unknown Title>	45007	1
11	2008	Michael Clayton [videorecording] / Warner~	4957	2
12	2008	No country for old men [videorecording] /~	4693	3
13	2009	<Unknown Title>	41854	1
14	2009	Burn after reading [videorecording] / Foc~	6152	2
15	2009	Juno [videorecording] / Fox Searchlight P~	5501	3

### Advanced Window Functions:

```
# Calculate running totals and percentage changes
monthly_trends <- seattle_tbl |>
group_by(CheckoutYear, CheckoutMonth) |>
summarise(monthly_checkouts = sum(Checkouts), .groups = "drop") |>
arrange(CheckoutYear, CheckoutMonth) |>
mutate(
  # Running total throughout the dataset
  cumulative_checkouts = cumsum(monthly_checkouts),
  # Previous month's value for comparison
  prev_month = lag(monthly_checkouts),
```

```

# Percentage change from previous month
pct_change = (monthly_checkouts - prev_month) / prev_month * 100
) |>
collect()

```

Warning: ORDER BY is ignored in subqueries without LIMIT

i Do you need to move arrange() later in the pipeline or use window\_order() instead?

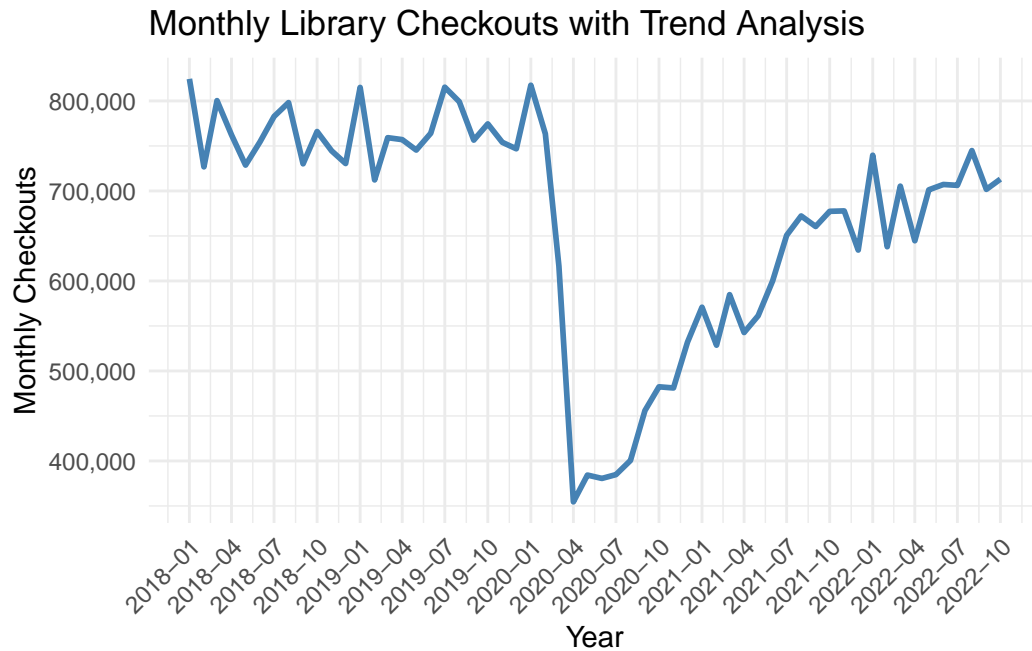
```

# Visualize the trends
monthly_trends |>
  filter(CheckoutYear >= 2018) |>
  mutate(date = as.Date(paste(CheckoutYear, CheckoutMonth, "01", sep = "-"))) |>
  ggplot(aes(x = date, y = monthly_checkouts)) +
  geom_line(color = "steelblue", size = 1) +
  scale_x_date(date_breaks = "3 months", date_labels = "%Y-%m") +
  scale_y_continuous(labels = scales::comma_format()) +
  theme_minimal() +
  labs(
    title = "Monthly Library Checkouts with Trend Analysis",
    x = "Year",
    y = "Monthly Checkouts"
  ) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

```

Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.

i Please use `linewidth` instead.



### Superpower 3: SQL and dplyr Interchangeability

One of DuckDB's greatest strengths is that you can use SQL and dplyr **interchangeably**:

```
# Method 1: Pure dplyr
dplyr_result <- seattle_tbl |>
  filter(CheckoutYear == 2020) |>
  group_by(MaterialType) |>
  summarise(total = sum(Checkouts)) |>
  arrange(desc(total)) |>
  collect()

#inspect
dplyr_result
```

```
# A tibble: 43 x 2
  MaterialType      total
  <chr>            <dbl>
1 EBOOK           2793961
2 AUDIOBOOK       1513625
3 BOOK            1241999
4 VIDEODISC        361587
```

```

5 SOUNDDISC          116221
6 MIXED              9118
7 REGPRINT           7573
8 VIDEO              2430
9 MUSIC              2404
10 SOUNDDISC, VIDEODISC 1049
# i 33 more rows

```

```

# Method 2: Pure SQL
sql_result <- dbGetQuery(con, "
  SELECT MaterialType, SUM(Checkouts) as total
  FROM seattle_checkouts
  WHERE CheckoutYear = 2020
  GROUP BY MaterialType
  ORDER BY total DESC
")

#inspect
sql_result

```

	MaterialType	total
1	EBOOK	2793961
2	AUDIOBOOK	1513625
3	BOOK	1241999
4	VIDEODISC	361587
5	SOUNDDISC	116221
6	MIXED	9118
7	REGPRINT	7573
8	VIDEO	2430
9	MUSIC	2404
10	SOUNDDISC, VIDEODISC	1049
11	CR	789
12	LARGEPRINT	697
13	ER	624
14	SOUNDREC	499
15	ER, VIDEODISC	244
16	MAP	202
17	ER, SOUNDDISC	197
18	ATLAS	148
19	VISUAL	111
20	UNSPECIFIED	62
21	REGPRINT, SOUNDDISC	30

22	VIDEOREC	19
23	VIDEOCASS	17
24	MUSICSDREC	17
25	ER, NONPROJGRAPH	14
26	VIDEOCART	12
27	REGPRINT, VIDEOREC	10
28	KIT	9
29	SOUNDCASS	9
30	NOTATEDMUSIC	8
31	SOUNDDISC, SOUNDREC	7
32	FLASHCARD, SOUNDDISC	6
33	MICROFORM	5
34	ER, PRINT	3
35	SOUNDDISC, VIDEOCASS	2
36	SOUNDCASS, SOUNDDISC	2
37	PICTURE	1
38	SOUNDCASS, SOUNDDISC, VIDEOCASS, VIDEODISC	1
39	FLASHCARD	1
40	SOUNDCASS, SOUNDDISC, SOUNDREC	1
41	ER, VIDEOREC	1
42	PHOTO	1
43	GLOBE	1

```
# Method 3: Mixed approach - see the SQL that dplyr generates
seattle_tbl |>
  filter(CheckoutYear == 2020) |>
  group_by(MaterialType) |>
  summarise(total = sum(Checkouts)) |>
  show_query()
```

```
<SQL>
SELECT MaterialType, SUM(Checkouts) AS total
FROM (
  SELECT seattle_checkouts.*
  FROM seattle_checkouts
  WHERE (CheckoutYear = 2020.0)
) q01
GROUP BY MaterialType
```

## Real-World Analytics Pipeline (25 minutes)

### Exercise 1: Multi-Table Analysis (10 minutes)

**Research Question:** “How have different categories of library materials (Books, Digital, Video, Audio) performed over time? Are digital formats really growing faster than physical ones?”

This is a perfect question for demonstrating **joins** - one of DuckDB’s superpowers! We want to:

- Create a lookup table to categorize our materials
- Join our main data with this categorization
- Analyze trends across categories and formats
- Compare digital vs. physical growth patterns

Step 1: Create a Material Categories Lookup Table

```
# Create a material categories lookup table using dplyr
material_categories <- seattle_tbl |>
  select(MaterialType) |>
  distinct() |>
  mutate(
    category = case_when(
      MaterialType %in% c('EBOOK', 'AUDIOBOOK') ~ 'Digital',
      MaterialType %in% c('BOOK', 'LARGEPRINT', 'REGPRINT') ~ 'Books',
      str_detect(MaterialType, 'VIDEO|DVD') ~ 'Video',
      str_detect(MaterialType, 'SOUND|CD') ~ 'Audio',
      TRUE ~ 'Other'
    ),
    format_type = case_when(
      MaterialType %in% c('EBOOK', 'AUDIOBOOK') ~ 'Digital',
      TRUE ~ 'Physical'
    )
  ) |>
  compute(name = "material_categories")

# Create our reference to the new table (it's already created above)
material_cat <- material_categories

# Look at our categories
```



```
material_cat |>
  collect()
```

```
# A tibble: 71 x 3
  MaterialType      category format_type
  <chr>            <chr>      <chr>
1 MOVIE            Other    Physical
2 ATLAS            Other    Physical
3 ER               Other    Physical
4 VISUAL           Other    Physical
5 COMPFILE         Other    Physical
6 SOUNDCASS, SOUNDDISC, SOUNDREC Audio    Physical
7 VIDEODISC        Video    Physical
8 BOOK             Books    Physical
9 SONG             Other    Physical
10 COMIC           Other    Physical
# i 61 more rows
```

```
# Create our reference to the new table (it's already created above)
material_cat <- material_categories

# Look at our categories
material_cat |>
  collect()
```

```
# A tibble: 71 x 3
  MaterialType      category format_type
  <chr>            <chr>      <chr>
1 MOVIE            Other    Physical
2 ATLAS            Other    Physical
3 ER               Other    Physical
4 VISUAL           Other    Physical
5 COMPFILE         Other    Physical
6 SOUNDCASS, SOUNDDISC, SOUNDREC Audio    Physical
7 VIDEODISC        Video    Physical
8 BOOK             Books    Physical
9 SONG             Other    Physical
10 COMIC           Other    Physical
# i 61 more rows
```

## Step 2: Try a Complex Join (Let's See What Happens!)

Now let's join our main data with our lookup table:

```
# Complex analysis combining our main data with categories
category_analysis <- seattle_tbl |>
  left_join(material_cat, by = "MaterialType") |>
  filter(CheckoutYear >= 2018) |>
  group_by(CheckoutYear, category, format_type) |>
  summarise(
    total_checkouts = sum(Checkouts),
    unique_titles = n_distinct(Title),
    avg_checkouts_per_title = total_checkouts / unique_titles,
    .groups = "drop"
  ) |>
  collect()

#Inspect
category_analysis

# Visualize the category trends
category_analysis |>
  ggplot(aes(x = CheckoutYear, y = total_checkouts,
             color = category, linetype = format_type)) +
  geom_line(size = 1.2) +
  geom_point(size = 2) +
  theme_minimal() +
  labs(
    title = "Library Checkouts by Category and Format",
    x = "Year",
    y = "Total Checkouts",
    color = "Category",
    linetype = "Format"
  ) +
  scale_y_continuous(labels = scales::comma)
```

OHHHH NOOO!!!! You got an error. DO you know why? Read on young pawan!

### Why This Happens:

- **dbplyr** translates your code to SQL
- SQL doesn't allow you to reference a column you're creating in the same SELECT statement

- The error message is actually quite helpful - it tells you exactly what to do!

**The Fix:** You need to split this into two steps or use a different approach because SQL can't reference `MaterialType` after you've already transformed it in the same query. You either need to:

1. Use the original column name throughout
2. Split into multiple steps
3. Use a subquery approach with `compute()` between steps

The SQL engine gets confused when you try to reference a column that's being created in the same statement - it's like trying to use a variable before you've finished declaring it!

### Step 3: Fix the Code (Your Turn!)

Update the code above so it will run. **Hint:** Move the calculation to a separate `mutate()` step.

```
# Complex analysis combining our main data with categories
category_analysis <- seattle_tbl |>
  left_join(material_cat, by = "MaterialType") |>
  filter(CheckoutYear >= 2018) |>
  group_by(CheckoutYear, category, format_type) |>
  summarise(
    total_checkouts = sum(Checkouts),
    unique_titles = n_distinct(Title),
    .groups = "drop"
  ) |>
  # Add the calculated column in a separate mutate step
  mutate(avg_checkouts_per_title = total_checkouts / unique_titles) |>
  collect()
```

The key differences are:

1. In the `summarise()` function:

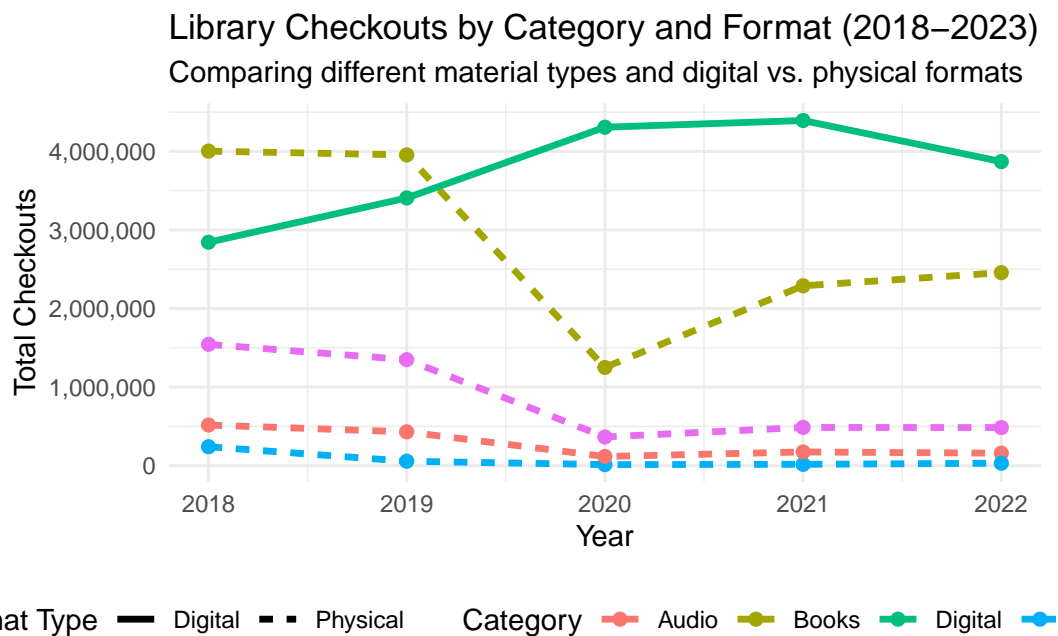
- **\*\*REMOVED:\*\*** ``avg_checkouts_per_title = total_checkouts / unique_titles,`` - This line

2. ADDED a separate `mutate()` step:

- ``mutate(avg_checkouts_per_title = total_checkouts / unique_titles) |>`` - This calcul

## Step 4: Visualize the Results

```
# Visualize the category trends
category_analysis |>
  ggplot(aes(x = CheckoutYear, y = total_checkouts,
             color = category, linetype = format_type)) +
  geom_line(size = 1.2) +
  geom_point(size = 2) +
  theme_minimal() +
  labs(
    title = "Library Checkouts by Category and Format (2018-2023)",
    subtitle = "Comparing different material types and digital vs. physical formats",
    x = "Year",
    y = "Total Checkouts",
    color = "Category",
    linetype = "Format Type"
  ) +
  scale_y_continuous(labels = scales::comma) +
  theme(legend.position = "bottom")
```



## Step 5: Answer Our Research Question

Turn and Talk (3 minutes):

- What trends do you see in digital vs. physical checkouts?
- Which categories seem most affected by recent changes?
- How might the library use this information for collection development?

#### **Key Learning Points:**

- **LEFT JOIN** kept all our checkout records and added category information
- **Two-step calculations** avoid SQL reference errors
- **Complex analysis** becomes manageable when broken into clear steps
- **Real insights** emerge from combining multiple data perspectives

#### **What is a JOIN? (Think of it like dating apps!)**

Imagine you have two lists:

- **List A:** People's names and their favorite books
- **List B:** Book titles and their authors

A **JOIN** is like playing matchmaker - you connect the two lists based on something they have in common (the book title)!

#### **Exercise 2: Understanding Joins with our data and duckdb (10 minutes)**

**Research Question:** “Do books with complete author information get checked out more often than books with missing author details? And how does this vary by publication year?”

This is a great question for learning joins! We want to:

- Compare checkout patterns for books with vs. without author information
- Look at how this changes over different publication years
- Combine information from multiple “tables” (views of our data)

#### **Step 1: Create Our “Tables” (Different Views of the Data)**

```

# Create books_info with cleaned data
books_info <- seattle_tbl |>
  select(Title, Creator, PublicationYear, MaterialType) |>
  filter(MaterialType == "BOOK") |>
  distinct() |>
  mutate(
    has_author = case_when(
      is.na(Creator) | Creator == "" ~ "Missing Author",
      TRUE ~ "Has Author"
    ),
    clean_year = as.numeric(sql("regexp_extract(PublicationYear, '\\d{4}'")),
    decade = case_when(
      clean_year >= 2010 ~ "2010s+",
      clean_year >= 2000 ~ "2000s",
      clean_year >= 1990 ~ "1990s",
      TRUE ~ "Pre-1990s"
    )
  )

# Create checkout_stats
checkout_stats <- seattle_tbl |>
  filter(MaterialType == "BOOK") |>
  group_by(Title) |>
  summarise(
    total_checkouts = sum(Checkouts),
    checkout_sessions = n(),
    .groups = "drop"
  ) |>
  mutate(avg_per_session = total_checkouts / checkout_sessions)

# Quick check both work
books_info |>
  head() |>
  collect()

```

# A tibble: 6 x 7

	Title	Creator	PublicationYear	MaterialType	has_author	clean_year	decade
	<chr>	<chr>	<chr>	<chr>	<chr>	<dbl>	<chr>
1	La tortuga ~	"Aliag~	2008.	BOOK	Has Author	2008	2000s
2	The Realist~	"	2016.	BOOK	Missing A~	2016	2010s+
3	Smooth talk~	"Kleyp~	2009.	BOOK	Has Author	2009	2000s
4	Playing Jul~	"Wetze~	[2015]	BOOK	Has Author	2015	2010s+

5	Light, colo~ "Birre~ [1969]	BOOK	Has Author	1969 Pre-1~
6	Seconds to ~ "Donla~ [2017]	BOOK	Has Author	2017 2010s+

```
checkout_stats |>
  head() |>
  collect()
```

```
# A tibble: 6 x 4
  Title                                total_checkouts checkout_sessions avg_per_session
  <chr>                                <dbl>             <dbl>          <dbl>
1 Dakota born / Debbie Macomb~         141               84            1.68
2 Batman : the long Halloween~        571              124            4.60
3 Cravings / Chrissy Teigen ;~       1138              75            15.2
4 Rude cakes / cooked up by R~         869              83            10.5
5 The man who folded himself ~         133              72             1.85
6 Honda Civic & CR-V automoti~         33               32             1.03
```

## Step 2: JOIN the Tables Together!

```
# Step 1: Create two simple "tables" - no complex cleaning needed!
books_summary <- seattle_tbl |>
  filter(MaterialType == "BOOK") |>
  group_by(Title) |>
  summarise(book_checkouts = sum(Checkouts), .groups = "drop")

digital_summary <- seattle_tbl |>
  filter(MaterialType %in% c("EBOOK", "AUDIOBOOK")) |>
  group_by(Title) |>
  summarise(digital_checkouts = sum(Checkouts), .groups = "drop")

# Step 2: Do different types of joins
# INNER JOIN - titles that exist in BOTH formats
both_formats <- books_summary |>
  inner_join(digital_summary, by = "Title") |>
  collect()

# LEFT JOIN - all books, with digital info if it exists
books_with_digital <- books_summary |>
  left_join(digital_summary, by = "Title") |>
  collect()
```

```
# Check results
glue("Books only: {nrow(books_summary |> collect())}")
```

Books only: 790313

```
glue("Digital only: {nrow(digital_summary |> collect())}")
```

Digital only: 317474

```
glue("Titles in BOTH formats: {nrow(both_formats)}")
```

Titles in BOTH formats: 3052

```
glue("Books with optional digital: {nrow(books_with_digital)}")
```

Books with optional digital: 790313

## Choose Your DuckDB Adventure (15 minutes)

Work in pairs to tackle one of these real-world analytical challenges:

### Beginner Challenge: Popular Authors Analysis

**Goal:** Analyze the trend from physical to digital library materials over time.

**Your mission:**

1. Create a list of digital material types
2. Use `case_when()` to classify materials as “Digital” or “Physical”
3. Group by CheckoutYear and format\_type
4. Calculate total checkouts per year for each format
5. Create two visualizations: trend lines and percentage breakdown

**Helpful hints for beginners:**

- Use `%in%` to check if MaterialType is in your digital\_materials list
- Use `case_when()` with `TRUE ~ "Physical"` as the default case



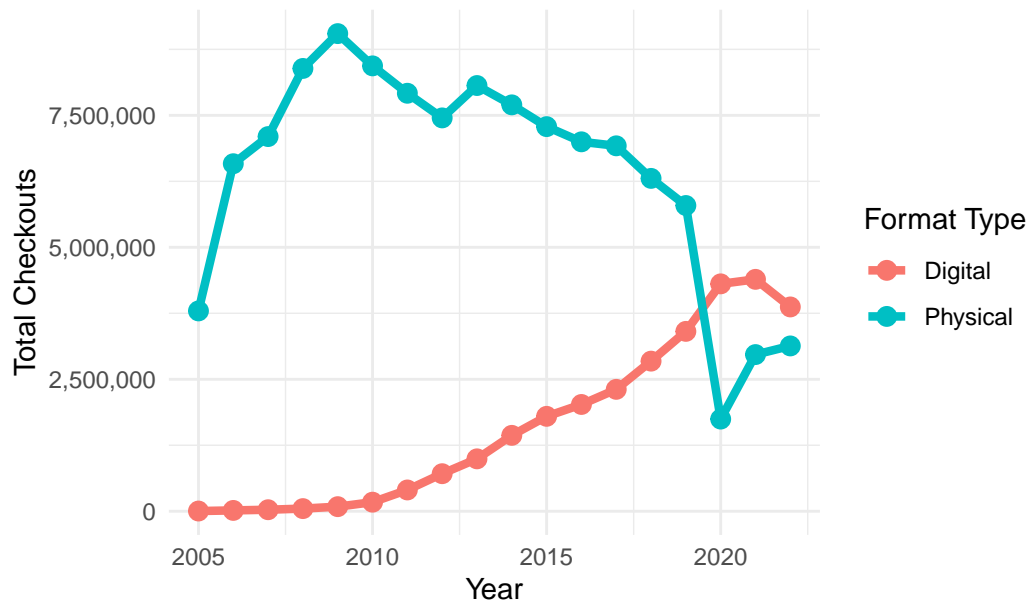
- Remember to use `.groups = "drop"` in your summarise
- Don't forget to `collect()` before creating visualizations!
- Use `geom_line()` for trends and `geom_col()` for percentages

```
# Classify materials as digital or physical
digital_materials <- c("EBOOK", "AUDIOBOOK", "DOWNLOAD", "ELECTRONIC RESOURCE")

material_trends <- seattle_tbl |>
  mutate(
    format_type = case_when(
      MaterialType %in% digital_materials ~ "Digital",
      TRUE ~ "Physical"
    )
  ) |>
  group_by(CheckoutYear, format_type) |>
  summarise(total_checkouts = sum(Checkouts, na.rm = TRUE), .groups = "drop") |>
  collect()

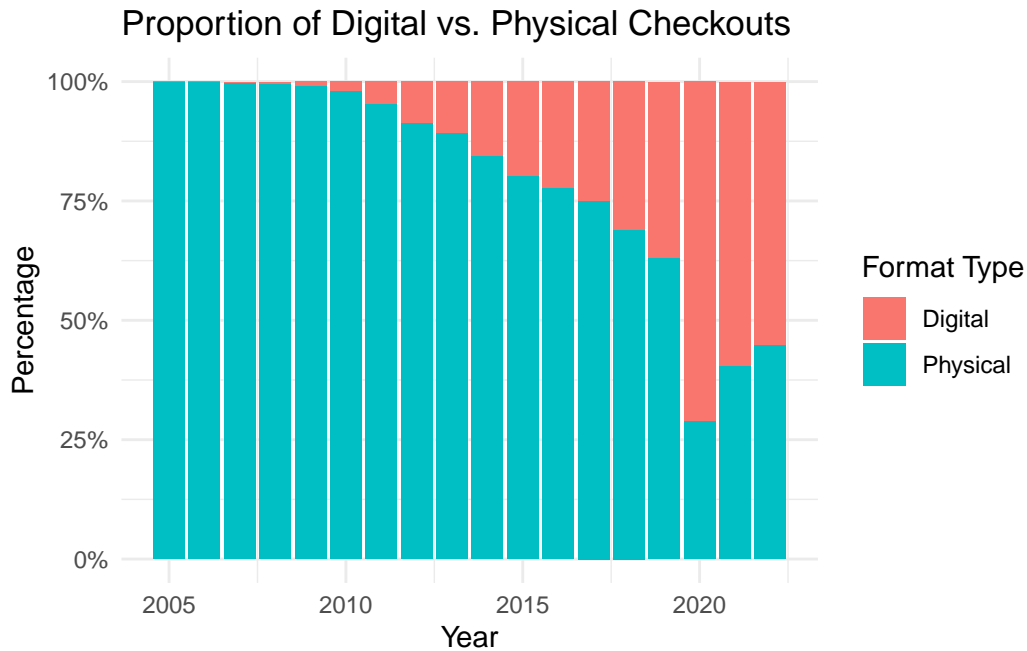
# Visualize the trends
ggplot(material_trends, aes(x = CheckoutYear, y = total_checkouts,
                           color = format_type, group = format_type)) +
  geom_line(linewidth = 1.5) +
  geom_point(size = 3) +
  theme_minimal() +
  labs(title = "Digital vs. Physical Checkouts Over Time",
       x = "Year", y = "Total Checkouts", color = "Format Type") +
  scale_y_continuous(labels = scales::comma)
```

### Digital vs. Physical Checkouts Over Time



```
# percentage of digital vs physical for each year
format_percentages <- material_trends |>
  group_by(CheckoutYear) |>
  mutate(percentage = total_checkouts / sum(total_checkouts) * 100) |>
  arrange(CheckoutYear, format_type)

# Visualize the changing percentages
ggplot(format_percentages, aes(x = CheckoutYear, y = percentage,
                              fill = format_type)) +
  geom_col() +
  theme_minimal() +
  labs(title = "Proportion of Digital vs. Physical Checkouts",
       x = "Year", y = "Percentage", fill = "Format Type") +
  scale_y_continuous(labels = function(x) paste0(x, "%"))
```



### Intermediate Challenge: Popular Authors Analysis

**Goal:** Find the most popular authors by checkout volume and diversity.

**Your mission:**

1. Filter to books only and remove missing author information
2. Group by Creator (author)
3. Calculate total checkouts, unique titles, and average checkouts per title
4. Filter to authors with at least 5 different titles
5. Rank by total checkouts and create a horizontal bar chart

**Helpful hints for intermediate:**

- Use `filter(!is.na(Creator) & Creator != "")` to remove missing authors
- Remember the mutate-after-summarise pattern for calculated columns
- Use `n_distinct(Title)` to count unique titles per author
- Use `coord_flip()` to make horizontal bars
- Use `reorder(Creator, total_checkouts)` to sort bars by value

```

# Step 1: Filter to books only
books_only <- seattle_tbl |>
  filter(MaterialType == "BOOK")

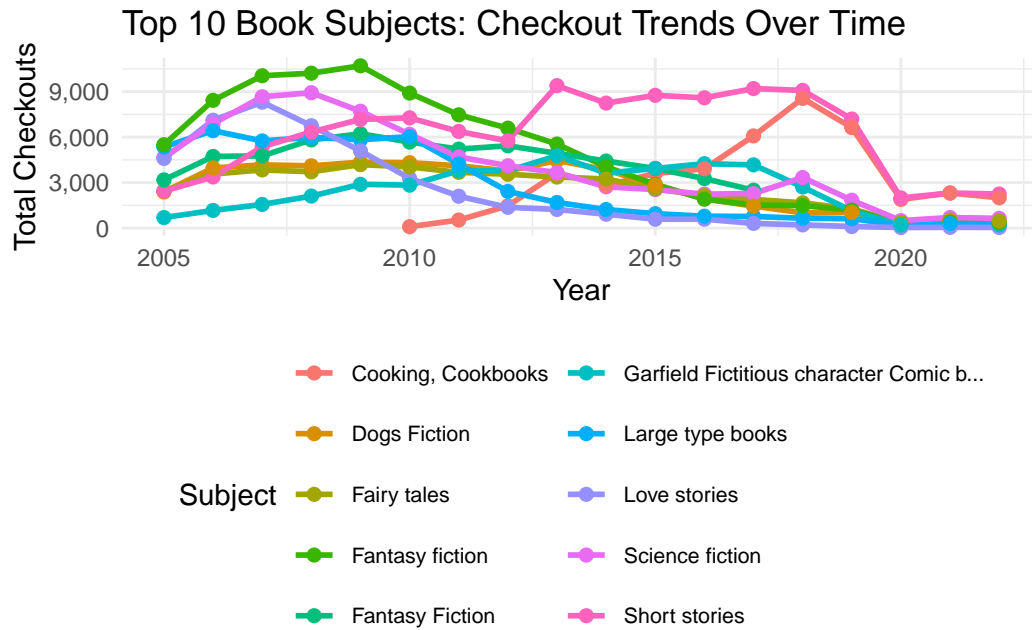
# Step 2: Group by Subjects column and CheckoutYear
# Step 3: Calculate total checkouts per subject per year
subject_analysis <- books_only |>
  filter(!is.na(Subjects) & Subjects != "") |> # Remove missing subjects
  group_by(Subjects, CheckoutYear) |>
  summarise(total_checkouts = sum(Checkouts), .groups = "drop") |>
  collect()

# Step 4: Find the top 10 subjects by total checkouts
top_subjects <- subject_analysis |>
  group_by(Subjects) |>
  summarise(total_all_years = sum(total_checkouts)) |>
  arrange(desc(total_all_years)) |>
  head(10) |>
  pull(Subjects)

# Filter to just the top 10 subjects for cleaner visualization
top_subject_trends <- subject_analysis |>
  filter(Subjects %in% top_subjects) |>
  mutate(Subjects = str_trunc(Subjects, 40)) # Shorten long subject names

# Step 5: Create a visualization showing trends over time
top_subject_trends |>
  ggplot(aes(x = CheckoutYear, y = total_checkouts, color = Subjects)) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  theme_minimal() +
  labs(
    title = "Top 10 Book Subjects: Checkout Trends Over Time",
    x = "Year",
    y = "Total Checkouts",
    color = "Subject"
  ) +
  scale_y_continuous(labels = scales::comma) +
  theme(legend.position = "bottom", legend.text = element_text(size = 8)) +
  guides(color = guide_legend(ncol = 2))

```



```
# Bonus: Summary table of top subjects
top_subjects_summary <- subject_analysis |>
  filter(Subjects %in% top_subjects) |>
  group_by(Subjects) |>
  summarise(
    total_checkouts = sum(total_checkouts),
    years_active = n_distinct(CheckoutYear),
    avg_yearly = round(mean(total_checkouts)),
    peak_year = CheckoutYear[which.max(total_checkouts)],
    peak_checkouts = max(total_checkouts)
  ) |>
  arrange(desc(total_checkouts))
```

top\_subjects\_summary

# A tibble: 10 x 6

	Subjects	total_checkouts	years_active	avg_yearly	peak_year	peak_checkouts
	<chr>	<dbl>	<int>	<dbl>	<dbl>	<dbl>
1	Short stori~	111096	18	111096	2021	111096
2	Fantasy fic~	87516	18	87516	2017	87516
3	Science fic~	72277	18	72277	2015	72277
4	Fantasy Fic~	60078	13	60078	2008	60078

5 Large type ~	49486	18	49486	2015	49486
6 Dogs Fiction	48917	18	48917	2007	48917
7 Fairy tales	46545	18	46545	2008	46545
8 Garfield Fi~	44044	18	44044	2017	44044
9 Cooking, Co~	43511	13	43511	2017	43511
10 Love stories	42690	18	42690	2010	42690

## Cleanup: Closing Your Database Connection

Always remember to close your database connections when you're done:

```
# Close the DuckDB connection
dbDisconnect(con, shutdown = TRUE)

# Verify it's closed
glue(" DuckDB connection closed successfully")
```

## Connection Best Practices:

```
# Method 1: Explicit cleanup (recommended for scripts)
con <- dbConnect(duckdb(), "data/seattle.duckdb")

# ... do your analysis ...
dbDisconnect(con, shutdown = TRUE)

# Method 2:
Automatic cleanup with on.exit() (good for functions)

analyze_library_data <- function()
  {con <- dbConnect(duckdb(), "data/seattle.duckdb")
  on.exit(dbDisconnect(con, shutdown = TRUE)) # Always runs, even if error

  # Your analysis code here
  result <- tbl(con, "seattle_checkouts") |>
    summarise(total = sum(Checkouts)) |>
    collect()

  return(result)
# Connection closes automatically when function exits }
```

```
# Method 3: Check for existing connections

if (exists("con") && !is.null(con)) {dbDisconnect(con, shutdown = TRUE)}
```

### Why close connections?

- **Free memory:** Releases database resources
- **Prevent errors:** Avoids “connection already exists” issues
- **Good practice:** Professional database hygiene
- **File locks:** Allows other processes to access the database file

### The `shutdown = TRUE` parameter:

- Completely shuts down the DuckDB instance
- Required for persistent databases (those saved to files)
- Ensures the database file is properly closed

## Key Takeaways from Module 3

### What You’ve Accomplished Today

**Congratulations!** You’ve just:

- Connected R to a high-performance analytical database
- Executed complex joins and aggregations on millions of rows

Used window functions for advanced time-series analysis

- Leveraged both SQL and dplyr syntax interchangeably
- Built persistent databases for reproducible analytics
- Optimized queries for maximum performance

## Essential DuckDB Patterns to Remember

### 1. The Connection Pattern:

```
...  
con <- dbConnect(duckdb(), "database.duckdb") # Create persistent DB  
seattle_tbl <- tbl(con, "seattle_weather")  
...
```

### 2. The SQL-dplyr Bridge Pattern:

```
...  
# See what dplyr generates  
query |>  
  show_query()  
  
# Use SQL directly when needed  
dbGetQuery(con, "SELECT ...")  
  
# Mix both approaches  
tbl(con, "table") |>  
  filter(...) |>  
  show_query()  
...
```

### 3. The Window Function Pattern:

```
...  
data |>  
  group_by(category) |>  
  window_order(date) |>  
  mutate(rank = row_number(),  
         cumulative = cumsum(value),  
         pct_change = (value - lag(value)) / lag(value))  
...
```

## When to Choose Each Tool

Task	Arrow	DuckDB
<b>Reading large files</b>	Best	Good
<b>Simple filtering</b>	Fast	Fast
<b>Complex joins</b>	Limited	Excellent



Task	Arrow	DuckDB
<b>Window functions</b>	No	Excellent
<b>SQL queries</b>	No	Native
<b>Persistent storage</b>	No	Yes
<b>Memory efficiency</b>	Excellent	Good

### Coming Up: Advanced Applications

Now you have the complete toolkit:

- **dplyr**: Intuitive data manipulation grammar
- **Arrow**: Efficient large file processing
- **DuckDB**: Database-powered analytics

In real projects, you'll often use all three:

1. **Arrow** to efficiently read your data
2. **DuckDB** for complex analytics and joins
3. **dplyr** as your consistent interface to both

### Quick Self-Assessment

I feel confident that I can:

- Connect R to DuckDB databases
- Choose between Arrow and DuckDB for different tasks
- Use window functions for advanced analytics
- Build reproducible analytical pipelines

I'd like more practice with:

- SQL queries
- Window function syntax
- Performance optimization
- Multi-table joins
- Mixed SQL/dplyr workflows

**Ready to apply these tools to your own data? You now have professional-grade capabilities for analyzing datasets of any size!**

*NSF Acknowledgement: This material is based upon work supported by the National Science Foundation under Grant #DGE-2222148. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.*