-- Aggregating
-- longlist.db

-- View ratings table
SELECT * FROM "ratings";

-- Find unrounded ratings
SELECT "book_id", AVG("rating") AS "rating" FROM "ratings"
GROUP BY "book_id";

-- Find rounded ratings
SELECT "book_id", ROUND(AVG("rating"), 2) AS "rating" FROM "ratings"
GROUP BY "book_id";

-- Add titles and year
SELECT "book_id", "title", "year", ROUND(AVG("rating"), 2) AS "rating" FROM "ratings"
JOIN "books" ON "ratings"."book_id" = "books"."id"
GROUP BY "book_id";

-- Convert to a view
CREATE VIEW "average_book_ratings" AS
SELECT "book_id" AS "id", "title", "year", ROUND(AVG("rating"), 2) AS "rating" FROM "ratings"
JOIN "books" ON "ratings"."book_id" = "books"."id"
GROUP BY "book_id";

-- Average book ratings by year nominated
SELECT "year", ROUND(AVG("rating"), 2) AS "rating" FROM "average_book_ratings"
GROUP BY "year";

-- Create temporary view of average ratings by year
CREATE TEMPORARY VIEW "average_ratings_by_year" ("year", "rating") AS
SELECT "year", ROUND(AVG("rating"), 2) AS "rating" FROM "average_book_ratings"
GROUP BY "year";

-- Leave SQLite to show average_ratings_by_year is temporary
.quit

-- When back in SQLite, drop the view "average_book_ratings"
DROP VIEW "average_book_ratings";

-- Highlight that CTEs can be views that are useful for the duration of a query
WITH "average_book_ratings" AS (
SELECT "book_id", "title", "year", ROUND(AVG("rating"), 2) AS "rating" FROM "ratings"
JOIN "books" ON "ratings"."book_id" = "books"."id"
GROUP BY "book_id"

SELECT "year" ROUND(AVG("rating"), 2) AS "rating" FROM "average_book_ratings"
GROUP BY "year";
-- Partitioning
-- longlist.db

-- Query for 2022 longlisted books
SELECT "id", "title" FROM "books"
WHERE "year" = 2022;

-- Create view of 2022 longlisted books
CREATE VIEW "2022" AS
SELECT "id", "title" FROM "books"
WHERE "year" = 2022;

-- Query for 2021 longlisted books
SELECT "id", "title" FROM "books"
WHERE "year" = 2021;

-- Create view of 2021 longlisted books
CREATE VIEW "2021" AS
SELECT "id", "title" FROM "books"
WHERE "year" = 2021;
-- Securing
-- rideshare.db

CREATE TABLE "rides" (  
    "id" INTEGER,  
    "origin" TEXT NOT NULL,  
    "destination" INTEGER NOT NULL,  
    "rider" TEXT NOT NULL,  
    PRIMARY KEY("id")  
);

INSERT INTO "rides" ("origin", "destination", "rider")  
VALUES  
('Good Egg Galaxy', 'Honeyhive Galaxy', 'Peach'),  
('Castle Courtyard', 'Cascade Kingdom', 'Mario'),  
('Metro Kingdom', 'Mushroom Kingdom', 'Luigi'),  
('Seaside Kingdom', 'Deep Woods', 'Bowser');

-- Reveal all rides information
SELECT * FROM "rides";

-- Reveal only subset of columns
SELECT "id", "origin", "destination" FROM "rides";

-- Make clear that rider is anonymous
SELECT "id", "origin", "destination", 'Anonymous' AS "rider" FROM "rides";

-- Create a view
CREATE VIEW "analysis" AS
SELECT "id", "origin", "destination", 'Anonymous' AS "rider" FROM "rides";

-- Query the view
SELECT "origin", "destination", "rider" FROM "analysis";
-- Simplifying
-- longlist.db

-- Find books written by a particular author
SELECT "title" FROM "books"
WHERE "id" IN (
  SELECT "book_id" FROM "authored"
  WHERE "author_id" = (SELECT "id" FROM "authors"
  WHERE "name" = 'Fernanda Melchor')
);

-- Open separate terminal window to run the below

-- Join authors with their book titles
SELECT "name", "title" FROM "authors"
JOIN "authored" ON "authors"."id" = "authored"."author_id"
JOIN "books" ON "books"."id" = "authored"."book_id";

-- Create a view from the query
CREATE VIEW "longlist" AS
SELECT "name", "title" FROM "authors"
JOIN "authored" ON "authors"."id" = "authored"."author_id"
JOIN "books" ON "books"."id" = "authored"."book_id";

-- View longlist
SELECT * FROM "longlist" LIMIT 5;

-- Drag tab to right to split window, then reveal the improved query

-- Query on the view
SELECT "title" FROM "longlist" WHERE "name" = 'Fernanda Melchor';
-- Soft deletions

-- mfa.db

-- View data
SELECT * FROM "collections";

-- View updated schema of collections table
.schema collections

-- Add a "deleted" column to "collections" table
ALTER TABLE "collections" ADD COLUMN "deleted" INTEGER DEFAULT 0;

-- View data
SELECT * FROM "collections";

-- View updated schema of collections table
.schema collections

-- Instead of deleting an item, update its deleted column to be 1
UPDATE "collections" SET "deleted" = 1 WHERE "title" = 'Farmers working at dawn';

-- Select all items from collections that are not deleted
SELECT * FROM "collections" WHERE "deleted" = 0;

-- Create a view with the same name as the collections table
CREATE VIEW "current_collections" AS
SELECT "id", "title", "acquisition_number", "acquired" FROM "collections" WHERE "deleted" = 0;

-- Select all from collections to see non-deleted items
SELECT * FROM "current_collections";

-- Fail to delete an item from the view
DELETE FROM "current_collections" WHERE "title" = 'Imaginative landscape';

-- Create trigger to delete items from a view
CREATE TRIGGER "delete"
INSTEAD OF DELETE ON "current_collections"
FOR EACH ROW
BEGIN
    UPDATE "collections" SET "deleted" = 1 WHERE "id" = OLD."id";
END;
-- Create trigger to revert an item's deletion
CREATE TRIGGER "insert_when_exists"
INSTEAD OF INSERT ON "current_collections"
FOR EACH ROW
WHEN NEW."accession_number" IN (SELECT "accession_number" FROM "collections")
BEGIN
  UPDATE "collections" SET "deleted" = 0 WHERE "accession_number" = NEW."accession_number";
END;

-- Create trigger to insert a new item to collections
CREATE TRIGGER "insert_when_new"
INSTEAD OF INSERT ON "current_collections"
FOR EACH ROW
WHEN NEW."accession_number" NOT IN (SELECT "accession_number" FROM "collections")
BEGIN
  INSERT INTO "collections" ("title", "accession_number", "acquired")
  VALUES (NEW."title", NEW."accession_number", NEW."acquired");
END;