```sql
-- Demonstrates views for aggregating data
-- Uses longlist.db

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

-- Returns book IDs and unrounded ratings
SELECT "book_id", AVG("rating") AS "rating" FROM "ratings"
GROUP BY "book_id";

-- Returns book IDs and rounded ratings
SELECT "book_id", ROUND(AVG("rating"), 2) AS "rating" FROM "ratings"
GROUP BY "book_id";

-- Adds book IDs, rounded ratings, title, and year columns
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";

-- Defines book IDs, rounded ratings, title, and year columns as 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";

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

-- Creates 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";

-- Drops the view "average_book_ratings"
DROP VIEW "average_book_ratings";

-- Shows that CTEs are views accessible 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";
```

```sql
-- Demonstrates views for partitioning data
-- Uses longlist.db

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

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

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

-- Creates view of 2021 longlisted books
CREATE VIEW "2021" AS
SELECT "id", "title" FROM "books"
WHERE "year" = 2021;
```

```sql
-- Demonstrates views for securing data
-- Uses 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');

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

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

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

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

-- Queries the view
SELECT "origin", "destination", "rider" FROM "analysis";
```

```sql
-- Demonstrates views for simplifying data access
-- Uses longlist.db

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

-- Joins 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";

-- Creates a view from the query to join authors with their book titles
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";

-- Returns first five rows from view
SELECT * FROM "longlist" LIMIT 5;

-- Finds books written by Fernanda Melchor, now using a view
SELECT "title" FROM "longlist" WHERE "name" = 'Fernanda Melchor';
```

```sql
-- Demonstrates soft deletes
-- Uses mfa.db

-- Views data in "collections" table
SELECT * FROM "collections";

-- Views schema of collections table
.schema collections

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

-- Views updated data in "collections table"
SELECT * FROM "collections";

-- Views updated schema of collections table
.schema collections

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

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

-- Creates a view to show only items in collections that are NOT deleted
CREATE VIEW "current_collections" AS
SELECT "id", "title", "accession_number", "acquired" FROM "collections" WHERE "deleted" = 0;

-- Selects from "current_collections" view to see non-deleted items
SELECT * FROM "current_collections";

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

-- Creates 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;

```

```sql
-- Creates 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;

-- Creates trigger to insert a new item into 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;
```