Week 2A SQL an...
File

remeber the content

Updated saved memory

已思考 5s

Got it — I've saved the **Week 2A "SQL and R – Movie Ratings"** assignment requirements (movies/ratings → SQL DB → load into R → handle missing ratings → GitHub submission, no passwords in code, etc.) for future reference.

Week 2A SQL and R Assignment Ov...

Sources

helping me write the code in SQL with the vars in the pdf

已思考 1m 7s

Here's a clean **starter SQL script** that matches the exact table/column "vars" your PDF suggests:

- `users(user_id, name)`
- `movies(movie_id, title)`
- `ratings(user_id, movie_id, rating)` Week 2A SQL and R Assignment Ov...

Below are **PostgreSQL (recommended)** and **SQLite** versions, plus a couple **JOIN queries** (also called out in the checklist). Week 2A SQL and R Assignment Ov...

PostgreSQL version (recommended)

Notes:

- “Uses **PK/FK** and a **junction table** (ratings) like the roadmap describes.

  Week 2A SQL and R Assignment Ov... ”
- “ `rating` allows decimals (so you can store 3.5). If you want **integers only**, change `NUMERIC(2,1)` → `INTEGER` and the CHECK accordingly.”

sql

```sql
-- =========================
-- Week 2A: Movie Ratings DB
-- PostgreSQL
-- =========================

DROP TABLE IF EXISTS ratings;
DROP TABLE IF EXISTS movies;
DROP TABLE IF EXISTS users;

-- 1) Dimension tables
CREATE TABLE users (
  user_id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  name    TEXT NOT NULL UNIQUE
);

CREATE TABLE movies (
  movie_id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  title    TEXT NOT NULL UNIQUE
);

-- 2) Junction table (many-to-many)
CREATE TABLE ratings (
  user_id INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
  movie_id INTEGER NOT NULL REFERENCES movies(movie_id) ON DELETE CASCADE,
  rating  NUMERIC(2,1),  -- use INTEGER if you want strict 1–5 integers
  PRIMARY KEY (user_id, movie_id),
  CHECK (rating IS NULL OR (rating >= 1 AND rating <= 5))
);
```

```
-- 3) Seed users
INSERT INTO users (name) VALUES
  ('M'), ('J'), ('W'), ('HP'), ('ZH'), ('CO'), ('ZY');

-- 4) Seed movies/items (edit titles however you want)
INSERT INTO movies (title) VALUES
  ('Zootopia 2'),
  ('Stranger Things 5'),
  ('Captain America 4'),
  ('Now You See Me 3'),
  ('Snow White'),
  ('The SpongeBob Movie: Search for SquarePants');

-- 5) Insert ratings (missing ratings can simply be omitted)
--    This pattern avoids hard-coding ids by joining on name/title.
INSERT INTO ratings (user_id, movie_id, rating)
SELECT u.user_id, m.movie_id, v.rating
FROM (VALUES
  ('M','Zootopia 2',4),
  ('M','Stranger Things 5',5),
  ('M','Captain America 4',3),
  ('M','Now You See Me 3',4),
  ('M','Snow White',1),
  ('M','The SpongeBob Movie: Search for SquarePants',4),

  ('J','Stranger Things 5',5),
  ('J','The SpongeBob Movie: Search for SquarePants',4),

  ('W','Zootopia 2',5),
  ('W','Stranger Things 5',3),

  ('HP','Captain America 4',1),

  ('ZH','Now You See Me 3',4),
  ('ZH','Snow White',3.5),

  ('CO','Zootopia 2',4),
  ('CO','The SpongeBob Movie: Search for SquarePants',3),

  ('ZY','Stranger Things 5',4),
  ('ZY','Now You See Me 3',2)
) AS v(user_name, movie_title, rating)
JOIN users  u ON u.name  = v.user_name
JOIN movies m ON m.title = v.movie_title;

-- 6) A tidy "analysis-ready" view (this is what you usually pull into R)
SELECT
  u.user_id,
  u.name       AS user_name,
  m.movie_id,
  m.title      AS movie_title,
  r.rating
FROM ratings r
JOIN users  u ON u.user_id  = r.user_id
JOIN movies m ON m.movie_id = r.movie_id
ORDER BY u.name, m.title;
```

Optional: make “missing ratings” explicit as `NULL`

Your checklist mentions verifying missing ratings using `NULL` .

Week 2A SQL and R Assignment Ov...

If you want **every user–movie pair** to exist (rated or not), do this:

sql

```
-- Create a full user x movie grid with NULL ratings (run after seeding users/mov
INSERT INTO ratings (user_id, movie_id, rating)
SELECT u.user_id, m.movie_id, NULL
FROM users u
CROSS JOIN movies m
ON CONFLICT (user_id, movie_id) DO NOTHING;
```

Copy code

Then you can run:

sql

```
-- Find explicit missing ratings
SELECT u.name, m.title
FROM ratings r
JOIN users u  ON u.user_id = r.user_id
JOIN movies m ON m.movie_id = r.movie_id
WHERE r.rating IS NULL
ORDER BY u.name, m.title;
```

Copy code

SQLite version (if you’re using SQLite)

sql

```sql
-- ========================
-- Week 2A: Movie Ratings DB
-- SQLite
-- ========================

PRAGMA foreign_keys = ON;

DROP TABLE IF EXISTS ratings;
DROP TABLE IF EXISTS movies;
DROP TABLE IF EXISTS users;

CREATE TABLE users (
  user_id INTEGER PRIMARY KEY,
  name    TEXT NOT NULL UNIQUE
);

CREATE TABLE movies (
  movie_id INTEGER PRIMARY KEY,
  title    TEXT NOT NULL UNIQUE
);

CREATE TABLE ratings (
  user_id  INTEGER NOT NULL,
  movie_id INTEGER NOT NULL,
  rating   REAL, -- use INTEGER if you want strict integers
  PRIMARY KEY (user_id, movie_id),
  FOREIGN KEY (user_id)  REFERENCES users(user_id)  ON DELETE CASCADE,
  FOREIGN KEY (movie_id) REFERENCES movies(movie_id) ON DELETE CASCADE,
  CHECK (rating IS NULL OR (rating >= 1 AND rating <= 5))
);
```

```
INSERT INTO users (name) VALUES
  ('M'), ('J'), ('W'), ('HP'), ('ZH'), ('CO'), ('ZY');

INSERT INTO movies (title) VALUES
  ('Zootopia 2'),
  ('Stranger Things 5'),
  ('Captain America 4'),
  ('Now You See Me 3'),
  ('Snow White'),
  ('The SpongeBob Movie: Search for SquarePants');

INSERT INTO ratings (user_id, movie_id, rating)
SELECT u.user_id, m.movie_id, v.rating
FROM (
  VALUES
  ('M','Zootopia 2',4),
  ('M','Stranger Things 5',5)
  -- ...continue like Postgres version...
) AS v(user_name, movie_title, rating)
JOIN users  u ON u.name  = v.user_name
JOIN movies m ON m.title = v.movie_title;
```

The 2 queries you'll use most in R

sql

```
-- (1) Tidy joined table (best for importing to R)
SELECT u.name AS user_name, m.title AS movie_title, r.rating
FROM ratings r
JOIN users  u ON u.user_id  = r.user_id
JOIN movies m ON m.movie_id = r.movie_id
ORDER BY user_name, movie_title;

-- (2) Per-movie summary (ignores NULL automatically in AVG in Postgres; in SQLit
SELECT m.title,
       COUNT(r.rating) AS n_ratings,
       AVG(r.rating)   AS avg_rating
FROM movies m
LEFT JOIN ratings r ON r.movie_id = m.movie_id
```

```
GROUP BY m.title
ORDER BY avg_rating DESC;
```

Copy code

If you tell me **which database you're actually using (Postgres or SQLite)**, I'll tailor it to your exact setup (e.g., identity/serial choice, upsert style, and a "single command" import-friendly `.sql` file layout).

Sources

how to connect to R

To pull your SQL tables into **R**, the standard approach is **DBI + a database driver**:

- **PostgreSQL** → `DBI + RPostgres`
- **SQLite** → `DBI + RSQLite`

Below are copy-paste templates for both 👇

Option A: PostgreSQL → R (recommended)

1) Install packages

r Copy code

```r
install.packages(c("DBI", "RPostgres", "dplyr"))
```

2) Connect

r Copy code

```r
library(DBI)
library(RPostgres)

con <- dbConnect(
  RPostgres::Postgres(),
  host     = "localhost",
  port     = 5432,
  dbname   = "movie_ratings",
  user     = "postgres",
  password = "YOUR_PASSWORD"  # better: use env var, see note below
)
```

3) Check tables

r Copy code

```r
dbListTables(con)
```

4) Pull a joined “analysis-ready” table into R

r Copy code

```r
sql <- "
SELECT u.name AS user_name, m.title AS movie_title, r.rating
FROM ratings r
JOIN users  u ON u.user_id  = r.user_id
JOIN movies m ON m.movie_id = r.movie_id
ORDER BY user_name, movie_title;
"

ratings_df <- dbGetQuery(con, sql)
ratings_df
```

5) Close connection

r Copy code

```r
dbDisconnect(con)
```

✅ Credential best practice (no passwords in code):

Use environment variables:

```r
Sys.getenv("PGPASSWORD")
```

Set once in your `.Renviron` (or Windows env vars), e.g.:

```ini
PGPASSWORD=your_password_here
```

Then connect without `password=` :

```r
con <- dbConnect(RPostgres::Postgres(),
                 host="localhost", port=5432,
                 dbname="movie_ratings", user="postgres")
```

Option B: SQLite → R (simplest)

1) Install packages

```r
install.packages(c("DBI", "RSQLite", "dplyr"))
```

2) Connect to a local `.sqlite` / `.db` file

```r
library(DBI)
library(RSQLite)

con <- dbConnect(SQLite(), dbname = "movie_ratings.sqlite")
dbListTables(con)
```

3) Read tables

r Copy code

```r
users   <- dbReadTable(con, "users")
movies  <- dbReadTable(con, "movies")
ratings <- dbReadTable(con, "ratings")
```

4) Or run a SQL join directly

r Copy code

```r
ratings_df <- dbGetQuery(con, "
SELECT u.name AS user_name, m.title AS movie_title, r.rating
FROM ratings r
JOIN users  u ON u.user_id  = r.user_id
JOIN movies m ON m.movie_id = r.movie_id
ORDER BY user_name, movie_title;
")
```

5) Disconnect

r Copy code

```r
dbDisconnect(con)
```

Quick R summary (after you've pulled `ratings_df`)

r Copy code

```r
library(dplyr)

ratings_df %>%
  group_by(movie_title) %>%
  summarise(
    n_ratings = sum(!is.na(rating)),
    avg_rating = mean(rating, na.rm = TRUE),
    .groups = "drop"
  ) %>%
  arrange(desc(avg_rating))
```

If you tell me **which one you're using (Postgres vs SQLite)** and what your **DB name** is (and whether it's local), I'll plug in the exact connection values you should use.

I dowload the pgadmin4

已思考 10s ›

Nice — **pgAdmin 4** is a great start. One important detail:

- **pgAdmin 4 is only a GUI client.**
- To connect from **R**, you still need a running **PostgreSQL server** (local on your laptop or remote/cloud).

Below is the practical "do this → then this" path.

1) Make sure PostgreSQL server is installed and running

If you installed "PostgreSQL" (not just pgAdmin)

You should have:

- a server running (usually on **localhost**)
- default port **5432**
- a username like **postgres**
- a password you set during installation

If you only installed pgAdmin 4

Then install **PostgreSQL** (the database server).
After that, pgAdmin will connect to the server, and R will connect the same way.

2) In pgAdmin: find the exact connection info R needs

In pgAdmin, right-click your server → **Properties** → **Connection** tab.
Copy these fields:

pgAdmin field	R `dbConnect()` argument
Host name/address	`host`
Port	`port`
Maintenance database (or the DB you created)	`dbname`
Username	`user`
Password	`password` (or environment variable)

Also: create a database like `movie_ratings` in pgAdmin if you haven't:

- Databases → right click → **Create** → **Database...**

3) R: connect to PostgreSQL (recommended way)

Install packages

r

Copy code

```r
install.packages(c("DBI", "RPostgres"))
```

Connect (basic)

r

Copy code

```r
library(DBI)
library(RPostgres)

con <- dbConnect(
  RPostgres::Postgres(),
  host = "localhost",
  port = 5432,
  dbname = "movie_ratings",
  user = "postgres",
  password = "YOUR_PASSWORD"
)

dbGetQuery(con, "SELECT version();")  # quick test
dbListTables(con)

dbDisconnect(con)
```

If this works, you're fully connected.

4) Best practice: don't put passwords in your code (for GitHub)

Option A (recommended): put password in `.Renviron`

Create/edit the file `~/.Renviron` and add:

ini Copy code

```ini
PGPASSWORD=your_password_here
```

ChatGPT 5.2 Thinking

```
library(DBI)
library(RPostgres)

con <- dbConnect(
  RPostgres::Postgres(),
  host = "localhost",
  port = 5432,
  dbname = "movie_ratings",
  user = "postgres"
)
```

Option B (temporary for one session)

r Copy code

```r
Sys.setenv(PGPASSWORD = "your_password_here")
```

5) Pull your ratings data into R (the join you'll analyze)

r Copy code

```r
sql <- "
SELECT u.name AS user_name, m.title AS movie_title, r.rating
FROM ratings r
JOIN users  u ON u.user_id  = r.user_id
JOIN movies m ON m.movie_id = r.movie_id
ORDER BY user_name, movie_title;
"

ratings_df <- dbGetQuery(con, sql)
ratings_df
```

If you tell me what you see in pgAdmin:

- Host name/address
- Port
- Username
- Database name you created

…I'll plug them into the exact R `dbConnect()` line for your setup (Windows/local is usually `localhost:5432`).

show me the next steps

Here are the **next steps** from where you are now (pgAdmin 4 installed), in the exact order I'd do them.

1) Confirm PostgreSQL server is actually running

In **pgAdmin 4**:

1. Open pgAdmin → left panel **Servers**
2. If you see a server (often named **PostgreSQL 16** / **PostgreSQL 15**)
   - click it → enter password if asked
3. If you **do NOT** see a server, you likely installed **only pgAdmin** → install **PostgreSQL server** (database engine).

✅ Quick verification inside pgAdmin:

- Open **Query Tool** and run:

sql

```sql
SELECT version();
```

Copy code

2) Create your database (one time)

In pgAdmin:

- **Databases** → right click → **Create** → **Database...**
- Name: `movie_ratings` (or any name you want)

3) Run your SQL script to create tables + insert data

In pgAdmin:

1. Click your `movie_ratings` database
2. Open **Tools** → **Query Tool**
3. Paste your SQL (CREATE TABLE + INSERT)
4. Click **Execute**

✅ Confirm tables exist:

- In Query Tool run:

sql

```sql
SELECT table_name
FROM information_schema.tables
WHERE table_schema='public'
ORDER BY table_name;
```

Copy code

You should see: `movies` , `ratings` , `users` .

4) In R: install packages

r

```r
install.packages(c("DBI", "RPostgres", "dplyr"))
```

Copy code

5) In R: connect to the same database pgAdmin uses

In pgAdmin, right-click your server → **Properties** → **Connection** and copy:

- Host (usually `localhost`)
- Port (usually `5432`)
- Username (often `postgres`)
- Database name (`movie_ratings`)

Then in R:

r

```r
library(DBI)
library(RPostgres)

con <- dbConnect(
  RPostgres::Postgres(),
  host = "localhost",
  port = 5432,
  dbname = "movie_ratings",
  user = "postgres",
  password = "YOUR_PASSWORD"
)

dbGetQuery(con, "SELECT version();")
dbListTables(con)
```

Copy code

6) Pull your “analysis-ready” table into R

r

```
ratings_df <- dbGetQuery(con, "
SELECT u.name AS user_name, m.title AS movie_title, r.rating
FROM ratings r
JOIN users  u ON u.user_id  = r.user_id
JOIN movies m ON m.movie_id = r.movie_id
ORDER BY user_name, movie_title;
")

ratings_df
```

Copy code

7) Do your required basic summaries in R

r

```
library(dplyr)

ratings_df %>%
  group_by(movie_title) %>%
  summarise(
    n_ratings = sum(!is.na(rating)),
    avg_rating = mean(rating, na.rm = TRUE),
    .groups = "drop"
  ) %>%
  arrange(desc(avg_rating))
```

Copy code

8) Handle missing ratings (choose one strategy and show it)

Two common choices:

Strategy A (simple): missing ratings are just absent rows

- In SQL: you **don't insert** a row if someone didn't rate a movie
- In R: you summarize with `na.rm=TRUE` after joins or you keep counts of rated items

Strategy B (explicit NULLs): every user×movie exists, unrated = NULL

- Create a full grid (user × movie) with `NULL` ratings
- Then show `WHERE rating IS NULL` in SQL or `is.na()` in R

(If you want, I'll give you the exact SQL for Strategy B.)

9) Close connection

r Copy code

```r
dbDisconnect(con)
```

10) GitHub submission checklist

Put in your repo:

- SQL script (create tables + insert)
- R / Quarto file that connects, imports, summarizes, and explains missing strategy
- README explaining how to run it (and **no passwords in code**)

If you want this to be **zero-error**, tell me what pgAdmin shows in **Properties → Connection**:

- Host
- Port
- Username

- Your database name

  …and I’ll write the exact `dbConnect()` line for your machine.