**UPDATE employee**

**SET salary = salary + 1000**

**WHERE salary < 70000;**

1. **How many attributes are present in the address fragment?**

**San Francisco, CA 94110 USA**

4

1. **The Package table has the following columns:**

**Which column should be designated the primary key for the Package table?**

TrackingNumber

1. **Which data type will store "2022-01-10 14:22:12" as a temporal value without loss of information?**

DATETIME

1. **Which SQL command is an example of data definition language (DDL)?**

Alter

1. **How would a database engine process an update that violates a RESTRICT referential integrity constraint?**

The update would be rejected by the database

1. **The Member table will have the following columns:  
   ID—positive integer  
   FirstName—variable-length string with up to 100 characters  
   MiddleInitial—fixed-length string with 1 character  
   LastName—variable-length string with up to 100 characters  
   DateOfBirth—date  
   AnnualPledge—positive decimal value representing a cost of up to $999,999, with 2 digits for cents  
   Write a SQL statement to create the Member table.  
   Do not add any additional constraints to any column beyond what is stated.**

CREATE TABLE Member (  
ID INT UNSIGNED,  
FirstName VARCHAR(100),  
MiddleInitial CHAR(1),  
LastName VARCHAR(100),  
DateOfBirth DATE,  
AnnualPledge DECIMAL(8,2) UNSIGNED  
);

1. **The Rating table has the following columns:  
   RatingCode—variable-length string, primary key  
   RatingDescription—variable-length string  
   (Title—variable-length string, maximum 30 characters  
   RatingCode—variable-length string, maximum 5 characters  
   Write a SQL statement to create the Movie table. Designate the RatingCode column in the Movie table as a foreign key to the RatingCode column in the Rating table.**

CREATE TABLE Rating (  
RatingCode VARCHAR(100) PRIMARY KEY,  
RatingDescription VARCHAR(255)  
);  
CREATE TABLE Movie (  
Title VARCHAR(30),  
RatingCode VARCHAR(5),

FOREIGN KEY (RatingCode) REFERENCES Rating(RatingCode)  
);

DESCRIBE Movie;

DESCRIBE Rating;

1. **Which restriction applies when using a materialized view?**

When using a materialized view in SQL, there are several restrictions and considerations to keep in mind:  
  
Query Complexity: Materialized views are generally designed to store the results of a specific query or set of queries. As a result, they are most effective for simple and static queries. Complex queries involving aggregations, joins, or subqueries may not be suitable for materialized views.  
  
Storage and Maintenance: Materialized views require storage space to store the precomputed results. The size of the materialized view can grow significantly depending on the underlying data and the complexity of the query. Additionally, the materialized view needs to be refreshed or updated periodically to reflect changes in the underlying data, which may require additional maintenance overhead.  
  
Data Freshness: Materialized views are not always up-to-date with the latest data changes in the underlying tables. The view's data is only as fresh as the last time it was refreshed. Therefore, if real-time or near real-time data accuracy is required, a materialized view may not be the best solution.  
  
Transaction Consistency: Materialized views may not always provide immediate transaction consistency. If the materialized view is not updated in real-time or synchronized with the underlying data in a transactional manner, there can be delays or inconsistencies between the materialized view and the actual data.  
  
View Size and Performance: Depending on the size and complexity of the materialized view, query performance can be impacted. Materialized views with a large number of rows, complex joins, or frequent updates may experience slower query performance due to the need for maintaining and refreshing the view.  
  
It is important to carefully evaluate the trade-offs and considerations when deciding to use materialized views. They can provide significant performance improvements for specific use cases but may require additional management and maintenance overhead compared to regular views or direct querying of the underlying tables.

1. **The Movie table has the following columns:  
   ID—integer, primary key  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   A new column must be added to the Movie table:  
   Column name: Score  
   Data type: decimal(3,1)  
   Write a SQL statement to add the Score column to the Movie table.**

ALTER TABLE Movie

ADD Score DECIMAL(3,1);

1. **The Movie table has the following columns:  
   ID—integer, primary key  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   Write a SQL statement to create a view named MyMovies that contains the Title, Genre, and Year columns for all movies. Ensure your result set returns the columns in the order indicated.**

CREATE VIEW MyMovies **AS**  
SELECT Title, Genre, Year  
FROM Movie;

SELECT \* FROM MyMovies;

1. **A database has a view named MovieView.  
   Write a SQL statement to delete the view named MovieView from the database.**

DROP VIEW MovieView;

1. **The Movie table has the following columns:  
   ID—integer  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer**
2. **Write a SQL statement to modify the Movie table to make the ID column the primary key.**

ALTER TABLE Movie  
**ADD** PRIMARY KEY (ID);

1. **The Movie table has the following columns:  
   ID—integer, primary key  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   The YearStats table has the following columns:  
   Year—integer  
   TotalGross—bigint unsigned  
   Releases—integer**
2. **Write a SQL statement to designate the Year column in the Movie table as a foreign key to the Year column in the YearStats table.**

ALTER TABLE Movie

ADD CONSTRAINT Year

FOREIGN KEY (Year) REFERENCES YearStats(Year);

1. **The Movie table has the following columns:  
   ID—integer, primary key  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   Write a SQL statement to create an index named idx\_year on the Year column of the Movie table.**

CREATE INDEX idx\_year **ON** Movie (Year);

1. **The Movie table has the following columns:  
   ID—integer, primary key, auto-increment  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   The following data needs to be added to the Movie table:  
   Title Genre RatingCode Year  
   Pride and Prejudice Romance G 2005  
   Write a SQL statement to insert the indicated data into the Movie table.**

INSERT INTO Movie (Title, Genre, RatingCode, Year)  
VALUES ('Pride and Prejudice', 'Romance', 'G', 2005);

1. **The Movie table has the following columns:  
   ID—integer, primary key  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   Write a SQL statement to delete the row with the ID value of 3 from the Movie table.**

DELETE FROM Movie WHERE ID = 3; **ID—integer, primary key  
Title—variable-length string  
Genre—variable-length string  
RatingCode—variable-length string  
Year—integer  
Write a SQL statement to update the Year value to be 2022 for all movies with a Year value of 2020.**

**UPDATE** Movie  
**SET** Year = 2022  
WHERE Year = 2020;

1. **The Movie table has the following columns:  
   ID—integer, primary key  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   Write a SQL query to retrieve the Title and Genre values for all records in the Movie table with a Year value of 2020. Ensure your result set returns the columns in the order indicated.**

SELECT Title, Genre  
FROM Movie  
WHERE Year = 2020;

1. **The database contains a table named Movie.  
   Write a SQL query to return all data from the Movie table without directly referencing any column names.**

SELECT \*  
From Movie;

1. **The Movie table has the following columns:  
   ID—integer, primary key  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   Write a SQL query to display all Title values in alphabetical order A–Z.**

SELECT Title  
FROM Movie  
**ORDER BY** Title ASC;

1. **The Movie table has the following columns:  
   ID—integer, primary key  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   Write a SQL query to output the unique RatingCode values and the number of movies with each rating value from the Movie table as RatingCodeCount. Sort the results by the RatingCode in alphabetical order A–Z. Ensure your result set returns the columns in the order indicated.**

SELECT RatingCode, COUNT(\*) AS RatingCodeCount

FROM Movie

GROUP BY RatingCode

ORDER BY RatingCode;

1. **Which query illustrates performing an outer join of the Movie table with a different table?**

A screenshot of a computer

Description automatically generated

1. **Assume there are two tables, A and B.  
   Which rows will always be included in the result set if Table A is inner joined with Table B?**

Only rows in Tables A and B that share the join condition

1. **The Movie table has the following columns:  
   ID—integer, primary key  
   Title—variable-length string  
   Genre—variable-length string  
   RatingCode—variable-length string  
   Year—integer  
   Write a SQL query to return how many movies have a Year value of 2019.**

SELECT **COUNT(\*) AS** MovieCount  
FROM Movie  
WHERE Year = 2019;

1. **The Movie table has the following columns:  
   ID - integer, primary key  
   Title - variable-length string  
   Genre - variable-length string  
   RatingCode - variable-length string  
   Year - integer  
   The YearStats table has the following columns:  
   Year - integer  
   TotalGross - bigint unsigned  
   Releases - integer  
   Write a SQL query to display both the Title and the TotalGross (if available) for all movies. Ensure your result set returns the columns in the order indicated.**

SELECT Movie.Title, YearStats.TotalGross  
FROM Movie  
LEFT JOIN YearStats ON Movie.Year = YearStats.Year;

Thanks for reaching out to me.   Here is the 2nd attempt study plan:

1. Complete zyBooks content listed below.  I realize you’ve already gone through some of this content.  Reviewing the labs again will help with memorizing the statements.
2. If you’d like, take the [extra practice test](https://westerngovernorsuniversity-my.sharepoint.com/:w:/g/personal/maria_schenk_wgu_edu/EbIn3oVR3p1DmYmxvgDc3FsBUc3Ez6fADdMZgKPRIkefhQ?e=PL2ydH) and check it with the answer key.
3. Practice Test Coding Questions/Pre-assessment Coding Questions –This [document](https://westerngovernorsuniversity-my.sharepoint.com/:w:/g/personal/maria_schenk_wgu_edu/EeK2HHFpECRBlwIafyqO3VYBJXOBG_yPj6BbJS9jxMGmAQ?e=CtkiJA) provides a correlation of the practice test coding questions with the pre-assessment coding questions (along with answers).
4. Take this additional [Practice test](https://protect-us.mimecast.com/s/bw9CClYkq5czPEADKfGjDUq?domain=forms.office.com" \t "_blank).
5. When you’ve completed this, please click the Request Approval button. \*\*Please note that I am not able to approve until the zyBooks requirements are completed.