**IT Specialist Certificate in Databases Exam Objectives @ Certiport**

Candidates for this exam are seeking to prove introductory knowledge of how to design,

create, and query relational databases, such as **MySQL, Microsoft SQL Server, or Oracle.**

Candidates should have at least 150 hours of instruction, or hands-on experience

with database design and object management, data retrieval and manipulation, and

related troubleshooting.

Candidates should be able to write ANSI SQL statements and troubleshoot syntax errors and runtime errors.

The syntax in the exam is provided in ANSI SQL.

To be successful on the test, the candidate is also expected to have the following

prerequisite knowledge and skills:

* 8th grade reading skills
* Some familiarity with data organization concepts
* Critical thinking and problem-solving skills
* Digital literacy skills, including the ability to research, create content, and solve problems using technology

# 1. Database Design

1.1 Given a scenario, design tables for storing data

• Identify entities, rows/records, columns/fields

1.2 Given a scenario, identify the appropriate primary key

• Primary key, composite/compound key

1.3 Given a scenario, choose data types to meet requirements

• Definition and importance of data types; how data types affect storage

requirements; data types for storing text, numbers, dates and times, and

Boolean values

1.4 Given a scenario, design relationships between tables

• How to establish relationships using primary and foreign keys, entity-

relationship diagrams (ERDs), referential integrity

1.5 Normalize a database

• Reasons for normalization, how to normalize a database to third normal

form (3NF)

1.6 Given a scenario, identify data protection measures

• Backups, restore, principle of least privilege, GRANT, WITH GRANT

OPTION, REVOKE, purpose of roles

# 2. Database Object Management using DDL

2.1 Construct and analyze queries that create, alter, and drop tables

• Create, alter, and drop tables by using proper ANSI SQL syntax; NULL and

NOT NULL

2.2 Construct and analyze queries that create, alter, and drop views

• Create, alter, and drop views by using proper ANSI SQL syntax; purpose of

Views

2.3 Construct and analyze stored procedures and functions

• Input and output parameters, return values, purpose of stored procedures

2.4 Given a scenario, choose between clustered and non-clustered indexes

• When to use clustered vs. non-clustered indexes, syntax for creating indexes

# 3. Data Retrieval

3.1 Construct and analyze queries that select data

• INNER JOIN, LEFT JOIN, RIGHT JOIN, CROSS JOIN (Cartesian product),

and FULL OUTER JOIN; self joins; combine result sets by using UNION and

INTERSECT; DISTINCT; column alias; computed columns

3.2 Construct and analyze queries that sort and filter data

• ORDER BY, WHERE, LIKE, BETWEEN, AND, OR, NOT, TOP (LIMIT), IN, NOT

IN, ANY, ALL, NULL, NOT NULL, comparison operators

3.3 Construct and analyze queries that aggregate data

• GROUP BY, HAVING, MIN, MAX, COUNT, AVG (AVERAGE), SUM

# 4. Data Manipulation using DML

4.1 Construct and analyze INSERT statements

• INSERT INTO SELECT, INSERT INTO VALUES

4.2 Construct and analyze UPDATE statements

• Update data in a single table

4.3 Construct and analyze DELETE statements

• Delete data from a single table

# 5. Troubleshooting

5.1 Troubleshoot data object management query failures

• Syntax and runtime errors

5.2 Troubleshoot data retrieval query failures

• Syntax and runtime errors

5.3 Troubleshoot data manipulation query failures

• Syntax and runtime errors

Practice Exam

Q1: You plan to create a database for your organization. Employee information such as name and email address will be stored in the EMP table. Information about each department, including the department name and location, will be stored in the DEPT table. Which statement describes an entity in this database?

**X A1: Each column in the DEPT table represents an entity**

**V A2: Employees and departments are distinct entities**

E: In a relational database, entities represent real-world objects.

* It is typically these objects we want to store information about, and this is done by creating tables for each object.
* Entities do not define how employees are linked to departments.
* This defines a relationship in a relational database.
* Relationships are enforced using keys, like primary keys.
* An employee’s entities do not include name and email address; these are the employee’s attributes or properties, each attribute is stored in its own column.
* Each column in the DEPT table does not represent an entity. A column stores an attribute for an entity, such as ADDRESS or DEPT\_NAME.

Q2: You plan to create a database for your organization. Employee information such as name and email address will be stored in the EMP table. Information about each department, including the department name and location, will be stored in the DEPT table. How many records will you create in this database?

**A: One for each employee and one for each department**

E: You will create one record for each employee and one record for each department.

* A record, or row, represents an individual entity, such as an individual employee.
* Every row in a table should be uniquely identified, not include repeating attributes, and include data dependent on a key column.
* You will not create one record for each employee and department.
* Database records should stand on their own but should not include attributes for other entities, for example, an employee record should not include a department name.
* In this question, EMP and DEPT reference tables, which are collections of related records.
* In a database, attributes like name or email address should be stored in columns, this allows each row or record to include that attribute.

Q3: You are designing a database and you need to create five columns. How should you do this?

**A: Create five fields in a table**

E: You should create five fields in a table.

* In a relational database, the terms column and field are often interchangeable.
* Each field holds an attribute for the candidate row or record.
* For example, in the EMPLOYEES database, email addresses could be stored in the EMAIL field or column.
* You should not create five rows in a table, a row is composed of fields.
* You should not create five records in a table, a record is another name for a row.
* You should not create five tables in a database, tables are composed of records & fields.

Q4: For each organization member, you need to keep track of the member’s unique membership number, name and address, membership expiration date, and the payment date and amount for each dues payment received from the member. A table in your database would represent:

**A: The members of the organization**

E: A table in your database would represent the members of the organization.

* A table, also referred to as an entity, contains information about multiple instances of a specific type of object.
* In this case, you need to store information about each member, and information about each member’s monthly payments for membership dues.
* Therefore, you would create a table to store the member information and a table to store payment information.
* Within the table representing the members of the organization, you would define columns, also known as fields, for each attribute to be tracked for each member, such as membership number, name, address, and membership expiration date.
* In this case, the table that represents payments received from members would include the payment date and payment amount, and an additional column containing the membership number, to be able to relate the payments back to the associated member.
* The address for each member would be an attribute of the member and would be represented by a column in your Member table.
* The membership expiration date for each member would be an attribute of the member, and would be represented by a column in your Member table.

Q5: You are creating a table that has a column named Description. The Description column must store a string between 1 and 3,500 characters in length. The Description column must support text written in multiple languages. Which data type will require the least amount of storage space?

**A: nvarchar(3500)**

E: The nvarchar(3500) data type will meet the requirements, using the least amount of storage space.

* The nvarchar data type is used to store a Unicode string of varying length.
* A Unicode string is required to support all the special characters of multiple languages.
* The amount of disk space consumed is 2 \* the number of characters + 2
* You should not use the varbinary data type, this is used to store binary data, not character data that supports Unicode.
* You should not use the nchar(3500) data type, this is used to store fixed-length data, each field will consume 7,000 bytes (2 \* 3,500), regardless of the actual length of the data.
* You should not use the varchar(3500) data type, this is used to store variable-length strings, but does not support Unicode characters.

Q6: You are creating a table to store employee data. To ensure data integrity, one column in the table should prevent duplicates. Which constraint should you use?

A: Primary key

E: You should use a primary key constraint.

* Primary key constraints prevent duplicates and empty or null values in a column.
* Primary keys are often used on identity columns, for example, each employee may have a unique, not-null employee ID.
* You should not use a foreign key constraint, this forces referential integrity, it does not enforce uniqueness.
* This ensures that relationships between tables remain valid.
* A foreign key could be used to ensure an employee is not placed in a department that does not exist.
* You should not use a default constraint, this adds a default value to a field if one is not supplied when a record is created, it does not prevent duplicates.
* You should not use a NOT NULL constraint, this prevents empty fields in a column, where you may want to prevent employee records that do not include a mailing address.

Q7: Which column should be the primary key?

A: employee\_id

E: The employee\_id column should be the primary key.

* Primary key constraints prevent duplicates and empty or null values in a column.
* Primary keys are often used on identity columns, for example, you are storing information about employees, and each employee should have a unique, not-null employee ID.
* The department\_id column should not be the primary key, it is likely at least two employees will work in the same department, and a primary key prevents duplicate values, however department\_id would be a good foreign key.
* The manager\_id column should not be the primary key, it is likely at least two employees will work for the same manager, and a primary key prevents duplicate values, however manager\_id would be a good foreign key, in which a column in a table references another column in the same table, in this case, each manager\_id should be one employee’s employee\_id.
* You should not create a new column with a NOT NULL constraint, as primary keys do not allow null values.

Q8: Your company is merging with another organization and department records must be combined. However, you have discovered some departments share the same department ID. You need to ensure each department record can be uniquely identified. Which will you use?

A: Composite primary key

E: You will use a composite primary key.

* A composite primary key consists of two or more columns, this ensures each field in the key is populated (NOT NULL) and the combination of fields is unique.
* You could create a composite primary key on the department ID and department name columns, ensuring the combination is unique and NOT NULL.
* You should not use a self-referencing foreign key, here, a column in a table references another column in the same table, for example, the manager ID for each employee should reference the manager’s employee ID.
* A default constraint adds a default value to a field if one is not supplied when a record is created, it does not prevent duplicates.
* A NOT NULL constraint prevents empty fields in a column, for example, you may want to prevent department records that do not include a mailing address.

Q9: You are designing a table. Several fields will store small numbers up to 256. All numbers will be positive. Which data type meets this requirement while requiring the least amount of storage?

**A: smallint**

E: The smallint data type meets this requirement while requiring the least amount of storage, smallint can store values between -32,768 and 32,767, and smallint requires 2 bytes of storage.

* Tinyint cannot be used to meet this requirement, tinyint only stores numbers from 0 to 255.
* Int can store the required values, but int uses 4 bytes of storage.
* Bigint can store the required values, but bigint uses 8 bytes of storage.

Q10: You need to store 1, 0, or NULL values in a Microsoft SQL Server table column. 0 should be treated as false and 1 should be treated as true. No other data should be allowed. Which data type should you use?

A: bit

E: You should use the bit data type, as bit can store one of three values; 0, 1, or NULL.

* Bit converts true values to 1, and false values to 0, NULL means empty or not defined.
* You should not use the int data type, this stores numbers from -2\*31 to 2\*31.
* You should not use the numeric(1) data type, this can store any single digit number.
* You should not use the tinyint data type, this stores numbers from 0 to 255.

Q11: Your database design for a book seller includes two tables, BOOKS and AUTHORS. You need to ensure that each book references a valid author. Which constraint should you use?

**A1: NOT NULL**

**A2: Foreign key**

E: You should use a foreign key.

* A foreign key constraint enforces referential integrity, this ensures that relationships between tables remain valid.
* In this scenario, you could create a foreign key between the authorID columns in the BOOKS and AUTHORS tables, this ensures that every book record contains an author ID that exists in the AUTHORS table.
* You should not use a check constraint, this can evaluate column value against a simple expression, for example, a check constraint on the BOOKS table could verify that a book’s price is not over a certain value, check constraints do not enforce referential integrity.
* You should not use a unique constraint, this prevents duplicates in a column, and do not enforce referential integrity.
* You should not use a NOT NULL constraint, this prevents empty fields in a column, for example, you may want to prevent book records that do not include a book title.

Q12: You create the EMPLOYEES and DEPARTMENTS tables in your database to store information about your organization. Each employee can only belong to one department, while each department might have more than one employee. What should you include in your ERD for your database?

**A: A many-to-one relationship between EMPLOYEES and DEPARTMENTS**

E: The ERD for your database should include a many-to-one relationship between EMPLOYEES and DEPARTMENTS.

* ERDs are used to design relational databases and show relationships between tables.
* In this scenario, many employees may be linked or related to one department.
* You should not include a one-to-one relationship between EMPLOYEES and DEPARTMENTS in your ERD, this would mean each department can have only one employee.
* You should not include a foreign key on DEPARTMENTS referencing EMPLOYEES in your ERD, logically, a department can be created and exist without employees.
* A foreign key on DEPARTMENTS referencing EMPLOYEES would prevent this, as a department record could only be created if an employee was already in that department.
* You should not include a primary key on EMPLOYEES referencing DEPARTMENTS in your ERD, this is not used to enforce referential integrity and are composed of columns in a single table.

Q13: CREATE TABLE departments

{

Department\_id int,

Name varchar(255),

Location varchar(255)

};

What should you do to ensure referential integrity?

**A1: Create a primary key on the department\_id column in EMPLOYEES.**

**A2: Create a foreign key on the manager\_id column in EMPLOYEES.**

E: You should create a foreign key on the manager\_id column in EMPLOYEES.

* A foreign key constraint enforces referential integrity, to ensure that relationships between tables remain valid.
* Importantly, a foreign key can also ensure that relationships between columns in the same table remain valid.
* In this scenario, a foreign key on the manager\_id column referencing the employee\_id column will ensure that every manager is a valid employee, known as a self-referencing foreign key.
* You should not create a foreign key on the department\_id column in departments, a primary key will likely be created on this column; foreign keys are used to enforce referential integrity between a child key and a parent key, in this table, department\_id is a parent key.
* You should not create a primary key on the department\_id column in employees; a foreign key should be created on this column that references the department\_id column in the departments table, primary keys are not used to enforce referential integrity.
* You should not create a foreign key on the employee\_id column in employees, a primary key will likely be created on this column; foreign keys are used to enforce referential integrity between a child key and a parent key, in this table, employee\_id is a parent key.

Q14: You are creating a relational database that will be used to track car rentals. Each renter can rent multiple cars, a car can be rented by multiple drivers. You need to reduce the amount of redundant data. How should you store the car rental data?

**A: Create a CARS table, a RENTERS table, and a CARRENTALS table**

E: This scenario involves a many-to-many relationship between two entities, Renters & Cars.

* When you need to model a many-to-many relationship in a relational database, you should create a junction table that contains foreign key relationships with the primary key in the tables that contain the detailed information for the two entities.
* The detailed information about each car is stored in the CARS table, and the detailed information about each renter is stored in the RENTERS table.
* The CARRENTALS table references the entities involved in a specific rental.
* You should not create only a CARS table and a RENTERS table, if you created only these tables, you would not be able to associate a renter with a specific car for a rental period.
* You should not create only a CARS table and a CARRENTALS table, if you created only these tables, you would need to repeat renter information for each car rental, this would not reduce redundant data.
* You should not create only a CARRENTALS table, if you stored all data related to cars, renters, and car rentals in a single table, you would be storing a lot of redundant data because details about the car and the renter would be stored for each rental.

Q15: You are designing a database to track customer and sales information. You want to minimize redundant data that will be stored in the database. Each customer can place one or more orders; for each order, the customer may order one or more products sold by your company. Which tables should your database design include?

**A: A CUSTOMERS table, an ORDERS table, an ORDERDETAILS table, and a PRODUCTS table.**

E: Your database design should include a CUSTOMERS table, an ORDERS table, an ORDERDETAILS table, and a PRODUCTS table.

* Tables are used to store information about unique entities.
* To minimize the amount of redundant data that is stored, you should design your tables to include only information about a single entity.
* Each table would then include columns, specific only to that entity.
* In this case, the CUSTOMERS table would include attributes relating only to customers, such as the customer’s name.
* The ORDERS table would include attributes relating only to orders, such as the order date and the customer placing the order.
* The ORDERDETAILS table would include attributes for each item ordered, such as the quantity ordered.
* The PRODUCTS table would include attributes for each of the products sold by your company, such as the product name and quantity on hand.

Q16: You are creating a relational database that will store information about physicians and their patients. Each patient may have more than one physician, and each physician may have multiple patients. What should you include?

**A1: Two tables and a single many-to-many relationship**

**A2: Three tables and two one-to-many relationships**

E: You should include three tables and two one-to-many relationships.

* You would create a PHYSICIAN table and a PATIENT table.
* In this case, the relationship between physicians and patients is a many-to-many relationship, because each physician can have multiple patients, and each patient can have multiple physicians.
* To implement a many-to-many relationship in a relational database, you create an additional table known as a junction or intersection table.
* This table contains a column that references the primary key of the table on one side of the relationship, and another column that references the primary key of the table on the other side of the relationship.
* In this case, you might have a PHYSICIAN table with a primary key on the physicianID column containing data about each physician, and a PATIENT table with a primary key on the patientID column containing information about each patient.
* To implement the many-to-many relationship between PHYSICIANS and PATIENTS in this case, you would create a new table containing two columns, one that references the primary key, physicianID, in the PHYSICIAN table, and the other references the primary key, patientID, in the PATIENT table.
* Each row in the table would represent a patient-physician combination, you can include additional columns in the junction table if needed, to represent attributes for each patient-physician combination.

Q17: You have already created the PRODUCTS table and the STORES table, and need to create the INVENTORY table. Which statement should you issue?

**A: Create a table that has a single PRIMARY KEY constraint, two FOREIGN KEY constraints, and two DEFAULT constraints.**

E: You should create a table that has a single PRIMARY KEY constraint, two FOREIGN KEY constraints, and two DEFAULT constraints.

* To do so, you would issue the following statement:
  + CREATE TABLE Inventory
  + (
  + ProductID int NOT NULL REFERENCES Products(ProductID),
  + StoreID int NOT NULL REFERENCES Stores(StoreID),
  + InStock int NOT NULL DEFAULT(0),
  + OnOrder int NOT NULL DEFAULT(0),
  + CONSTRAINT FK\_Inventory PRIMARY KEY CLUSTERED
  + (
  + ProductID ASC,
  + StoreID ASC
  + )
  + )
* The requirements call for a composite primary key that consists of the storeID and productID columns.
* Therefore, you need to use the CONSTRAINT keyword to specify the columns that should be included in the primary key, the columns are also foreign keys.

Q18: The TESTSCORES table has these characteristics:

* Each student has a unique studentID that references the STUDENTS table.
* Each instructor has a unique instructorID that references the INSTRUCTORS table.
* Each student may take multiple tests on the same date.
* Each student may take tests on multiple dates.

What column, or columns, should you use for the primary key?

**A1: studentID, test, date, and instructorID**

**A2: studentID, test, and date**

E: You should create a primary key based on the studentID, test, and date columns.

* A primary key can consist of multiple columns, provided the combination of values across all key columns results in a unique value.
* In this example, you need three values to guarantee uniqueness: the studentID, test, and date.
* You should not use only the studentID and date, the student might take multiple tests on the same date, making this combination non-unique.
* You should not use only the studentID, test, date, and instructorID, the instructorID column allows nulls, a column that allows nulls cannot be included in a primary key.
* You should not use only test as the primary key value, multiple students take a test, so the test column is not a unique identifier.

Q19: EmpID int IDENTITY(1,1) PRIMARY KEY

You need to create a foreign key with the same column name in the SALARYHISTORY table that references the EmpID in the EMPLOYEE table. What must you ensure?

**A1: The EmpID column in the SALARYHISTORY table has a CHECK constraint.**

**A2: The two EmpID columns have a compatible data type.**

E: You must ensure that the two EmpID columns have a compatible data type.

* To create a foreign key to relate the two tables, the data types must be compatible.
* For example, you could not create a VARCHAR column in one table, that referenced an INT column in another table.
* You should not ensure that the EmpID column in the SALARYHISTORY table has the IDENTITY property; this causes values to be automatically assigned, this would not work because both columns would assign sequential numbers, instead of allowing you to store the parent table’s EmpID in the child table’s EmpID column.
* You should not ensure that the EmpID column in the SALARYHISTORY table has a CHECK constraint; this is used to ensure that a column has a value that meets a specified condition, such as a value that is within a given range, it is not required to create a foreign key relationship between two tables.
* You should not ensure that the SALARYHISTORY table does not contain any rows; you can add a FOREIGN KEY constraint to a table that already contains data, however, the data must conform to the defined relationship.

Q20: What should you do to create a table?

**A: Create an OWNERS table, a PETS table, and a VACCINATIONS table.**

**A: Create a column in the PETS table that references a column in the OWNERS table**

**A: Create a column in the VACCINATIONS table that references a column in the PETS table.**

E: The scenario indicates that there is a one-to-many relationship between owners and pets, and a one-to-many relationship between pets and vaccinations.

* The best way to map a one-to-many relationship is to create each entity as a separate table, and then add a column on the many sides of the relationship that references the entity on the one side of the relationship.

Q21: You want to ensure the table is 3NF compliant. What should you do?

Emp\_id name dept city

1 Mary HR NYC

2 Lam Sales Perth

3 Peter IT Moscow

4 Hiro Sales Perth

**A1: Create a new table to store city information.**

**A2: Move department information to a separate table**

E: Database normalization is used to reduce redundancies within tables.

* This can be done in stages, or forms, and a table is third normal form (3NF) compliant, when each record is atomic and non-key fields are dependent only on the primary key.
* In this question, employee records are not atomic, as they are mixed with department data.
* Additionally, the department name and city are not dependent on the presumed primary key (empID).
* These columns should be moved to a dedicated departments table and a new column linking employees to departments should be defined, using department\_id.
* You should not add a department\_id column to the employee table, adding a department\_id column to the employees table further denormalizes the table.
* Each employee record should be atomic and only include specific information about the employee, any related data should be moved to a different table.

Q22: You need to normalize the table. What should you do?

**A: Create a CONTRACTORS table with a reference to the MANAGERS table.**

E: Database normalization is used to reduce redundancies within tables.

* In this table, redundant data is stored in the contractor\_one, \_two, and \_three columns, as well as the contractors\_list.
* Ideally, the MANAGERS table will hold no contractor data, and can be reduced to just two or three columns.
* You should not consolidate the contractor\_one, \_two, and \_three columns into employee\_list, this will not normalize the data, the MANAGERS table should store data only for managers, not for contractors.
* You should not expand the contractors\_list into a column for each employee, expanding the column will have the same result as consolidating the contractor columns.
* This column is multi-valued and likely holds redundant data, both of which normalization attempts to prevent.

Q23: What is a reason to normalize a table?

**A: To reduce redundancy**

E: Normalization can be done in stages, or forms, and most database designs target third normal form (3NF).

* A database is 3NF-compliant when each record is atomic, or stands on its own, and non-key fields are dependent only on the primary key.
* Consolidating columns is not part of normalization; creating multi-value columns or fields is denormalization.
* Normalization will not reduce table count, this will increase as redundant data is moved into its own table.
* Normalization will not eliminate foreign keys, as redundant data is moved to new tables, it is likely foreign key count will increase.

Q24: What is one reason to denormalize a database?

A1: To improve data modification performance

A2: To improve data retrieval performance

E: One reason to denormalize a database is to improve data retrieval performance, by reducing the number of joins required for data retrieval queries that are frequently executed.

* Another reason to denormalize a database is to make it easier for users to execute ad hoc queries without an in-depth understanding of how tables are related.
* When a database is normalized, data is split into multiple tables to reduce redundant data.
* However, sometimes this normalization results in a large number of joins, because joins consume resources, you can sometimes improve performance by denormalizing the database and combining some tables.
* Denormalization does not improve data modification performance, and might even make it worse, because of the need to update the same data in multiple tables.
* Denormalization does not eliminate redundant data, the goal of normalization is to eliminate redundant data.
* Denormalization does not affect the privacy of confidential data, security features like privileges and encryption help protect the privacy of confidential data.

Q25: The requirement that each field value in a table is associated with only one row, is an example of normalizing a database to which form?

**A1: First normal form (1NF)**

**A2: Second normal form (2NF)**

E: When normalizing a database to 2NF, you identify any columns that have the same value for more than a single primary key value, and separate those columns into a different table, you then create foreign key relationships between the tables.

* The first normal form (1NF) ensures that a table is created for each entity, and that a primary key is identified for each table; however, it does not require the analysis of each column to ensure that it is dependent on the primary key.
* The third normal form (3NF) takes normalization a step further than 2NF, by ensuring that each piece of data in the table depends only on the primary key.
* The fourth normal form (4NF) provides even stronger normalization, by extracting columns that exhibit multi-valued dependencies to a different table.

Q26: EmployeeID, Spouse, Child1, Child2, Child3

Which statement correctly identifies the normalization level of this table?

**A: The table is not normalized.**

E: When you normalize a database, you must first normalize to the first normal form (1NF), this requires that you eliminate repeating groups, in this case, Child1, Child2, and Child3.

* When normalizing a database to a second normal form (2NF), you identify any columns that have the same value for more than a single primary key value, and separate those columns into a different table, you then can create foreign key relationships between the tables.
* The third normal form (3NF) takes normalization a step further than 2NF, by ensuring that each piece of data in the table depends only on the primary key.

Q27: Each teacher is associated with exactly one campus, the address in the table is the address for the campus where the teacher teaches. What change would you need to make to normalize the database to the third normal form (3NF)?

**A1: Create a separate table for campus address information, use the campus as the primary key, add a teacherID column to the table and relate it to the teacherID column in the teachers table.**

**A2: Create a separate table for campus address information, use the campus as the primary key, create a foreign key that relates the campus column in the teachers table to the primary key of the new table.**

E: When a database is normalized to 3NF, each attribute must be related to the primary key.

Q28: What is a characteristic of a database that is normalized to the first normal form (1NF)?

**A1: All tables are related using foreign keys.**

**A2: Tables do not contain multiple columns that represent similar attributes.**

E: All repeating groups of similar attributes are excluded, also each unique entity is represented by a single table that has a primary key that uniquely identifies each row.

* First normal form (1NF) does not ensure that all tables are related using foreign keys.
* Second normal form (2NF) ensures that related tables are related using foreign keys.
* However, some database tables may not be related at all, 2NF also ensures that you have separate tables for different value sets that are applicable to more than one record.
* 1NF does not ensure that all data is stored in a single table, data for different entities is stored in separate tables.
* 1NF does not ensure that tables contain only columns that are dependent on the table’s primary key, this is a characteristic of third normal form (3NF).

Q29: What is one result of normalizing a database?

**A: Data redundancy is minimized.**

E: A normalized database stores data about each entity in a separate table, this reduces the amount of redundant data stored in the table, normalized databases typically have more tables having fewer columns.

* Data redundancy is not increased with a normalized database, it is decreased because each table stores only information about a single entity.
* The number of tables is not fewer with a normalized database, it will have more tables than a database that is not normalized.
* Tables will not have more columns with a normalized database, it tends to have tables with fewer columns because each table contains only data for a single entity.

Q30: You are creating a relational database. Your current data model meets the following requirements:

* Each table represents a single entity identified by a primary key.
* Repeating columns have been eliminated from each table.
* All columns in each table depend only on the table’s primary key.
* Foreign keys have been implemented to relate the tables.

Your database is considered to be:

**A: Normalized to third normal form (3NF)**

E: Your database is considered to be normalized to the third normal form (3NF).

* A database normalized to the first normal form (1NF) has separate tables with no repeating attributes, with each table row identified by a primary key.
* A database normalized to the second normal form (2NF) has these characteristics, and also uses foreign keys to relate the tables.
* A database normalized to the third normal form (3NF) has all of these characteristics, and also has had any columns that are not totally dependent on the table’s primary key removed.
* Your database is not considered to be denormalized, this is the opposite of normalization.
* In a denormalized database, redundant data is stored in the database, and is sometimes used to improve query performance, or to be able to store historical data that has changed over time.
* Your database is not considered to be fragmented, a result of how the data is physically stored on the disk, not how it is logically designed.

Q31: A database that includes a table that contains repeating columns is said to be normalized. Is this statement correct?

**A: No; it is denormalized.**

E: When you normalize a database, you must first normalize to the first normal form (1NF) by eliminating all repeating columns.

* Fragmentation is a result of how the data is physically stored on the disk, not how it is logically designed.
* Defragmentation is the process of reducing fragmentation on a disk, by physically placing the disk contents in contiguous locations.

Q32: You create a base backup. Several days later, you decide to create a second backup, however, you only want to capture data that has changed. Which should you perform?

**A1: A partial backup**

**A2: A differential backup**

E: A differential backup captures changes since the last full backup.

* In a restore scenario, you would restore first the full backup, followed by the differential backup.
* You should not perform a copy-only backup, this captures the entire database (copy-only full backup) or the transaction logs (copy-only log backup), these backups capture more than recent changes.
* You should not perform a full backup, this captures the entire database, full backups are also known as base backups, as they serve as the base for differential backups.
* You should not perform a partial backup, this captures a limited subset of database filegroups, partial backups can speed recovery of very large databases.

Q33: Your server has multiple databases for each department in your organization. As employees move from one department to another, they are granted access to new databases. You are concerned about authorization creep. What should you do?

**A1: Eliminate the use of WITH GRANT OPTION**

**A2: Revoke unnecessary privileges**

E: You should revoke unnecessary privileges.

* As employees move between departments or change job titles, existing privileges are often not reviewed, revoking unnecessary privileges is also known as the principle of least privilege.
* You should not eliminate the use of WITH GRANT OPTION, this is used to allow a user to grant a privilege they have been given to another user. The issue in this scenario is not that users have been granted privileges, it is that those privileges are not regularly audited.
* You should not eliminate the use of GRANT, this is used to grant privileges to a user, required for users to access a database.
* You should not add employees to new roles. Roles, which are granted permissions, are typically defined based on job descriptions, and are then assigned to users with that job. Roles can help with authorization creep, however, like privileges, role assignments must be audited regularly.

Q34: As a database administrator, you try to record all permissions assigned to users in a spreadsheet. Recently, you have discovered some users have permissions you did not assign. What should you do to resolve this issue?

**A1: Change the grantor for each permission.**

**A2: Check user permissions for WITH GRANT**

E: You should check user permissions for With Grant.

* If a user has been granted permission using WITH GRANT OPTION, they can assign, or grant, the same permissions to other users, this makes permissions assignments difficult to track.
* You should not always use WITH GRANT OPTION, this is used to allow a user to grant a privilege they have been given to another user. You should avoid using WITH GRANT OPTION if you need to centrally manage and record permissions assignments.
* You should not ensure users have Grant and With Grant, users who have these for a permission can assign that permission to other users.
* You should not change the grantor for each permission, the grantor being the person who grants a permission, however, by locating the grantor, you could revoke their ability to grant permissions to other users.

Q35: You have a database that uses three read/write filegroups. The database is configured to use the Full recovery model.

You are planning the backup strategy for the database, you need to meet the following requirements:

* Disk space required to store the backups should be minimized
* The number of transaction log backups that need to be replayed during restoration should be minimized

What should your backup plan include?

**A1: Database backups, differential backups, and transaction log backups**

**A2: Database backups, differential backups, and transaction log backups**

E: A database backup backs up the entire database.

* This backup can serve as a base for one or more differential backups.
* Differential backups back up only the data that has changed since the base backup.
* Differential backups can reduce the number of transaction logs that must be replayed during restoration, because only the transaction log backups taken since the last differential backup need to be replayed.
* Your plan also needs to include transaction log backups, because the recovery model is set to Full.
* If transaction log backups are not taken, the transaction log will never be truncated and will eventually reach its maximum size, if the transaction log is full, no data modification operations can occur.

Q36: Saturday 22:00 - Database Backup

Monday - Friday 22:00 - Differential Backup

Every Two Hours - Transaction Log Backup

Drive D fails on Friday afternoon, and you need to recover the data. What should you do first?

**A: Back up the active transaction log**

E: The active transaction log is also called the tail of the log.

* You need to back up the active transaction log, so that you can replay it before you recover the database.
* If you do not back up and replay the active transaction log, all changes made since the last transaction log backup will be lost.
* You should not restore the Saturday backup first, restore it after backing up the active transaction log.
* You should not change the recovery mode to Simple, transaction log backups are not supported in the Simple recovery model.
* You should not restore the Thursday backup first, restore it after you restore the Saturday backup.

Q37: You have an application that requires permission to modify specific fields in the CUSTOMERS table in your database. Those fields should only be modifiable by the application. Permission to modify the values in fields should be granted to which type of object?

**A1: Database user**

**A2: Application role**

E: You should grant permission to an application role.

* An application role is used to grant permissions to a specific application, the application calls sp\_setapprole and provides a password to access the database under the security context of the role.
* You should not grant permission to a SQL Server login, a login is a server-level object that is used to grant the ability to establish a connection with an instance of SQL Server.
* If you grant permission to the login, any user who logs in using that login will be able to modify the values in the fields.
* You should not grant permission to a database user, a database user is associated with a login, Windows user, or Windows Group.
* You use a database user to grant access to database objects, however, the database user is not limited to being used by a specific application.
* You should not grant permission to a flexible database role, you can use a role to grant permission to database objects, however, any user in the role can perform the allowed actions, you cannot limit the scope of a flexible database role to a single application.

Q38: You create a stored procedure and need to allow a database user to run the stored procedure. Which statement should you use?

**A1: GRANT PERMISSION**

**A2: GRANT EXECUTE**

E: You should use the GRANT EXECUTE statement.

* The GRANT statement is used to give a database user, or group of users (defined with a role) specific permissions to database objects.
* The GRANT statement can include one or more permissions that should be allowed.
* The GRANT statement then uses an ON clause to identify the object(s) for which the permissions should be allowed, and a TO clause to identify the user(s) or roles that should be given the permissions.
* EXECUTE is the permission that allows a user to run a stored procedure, or call a user-defined function.
* You cannot use the GRANT PERMISSION statement, it is not a valid statement.
* You should not use the ALTER USER statement, this is used to modify a user’s name or schema, it cannot be used to assign permissions.
* You should not use the ALTER PROCEDURE statement, this is used to modify a stored procedure’s definition, it cannot be used to assign permissions.

Q39: What is one difference between an application role and a database role?

**A: A database role is assigned to users.**

E: A database role is assigned to users and groups.

* Permissions can be defined and granted to the role, and then all role members are allowed those permissions.
* Application roles allow access to database objects, but only through an application.
* Application roles are not assigned to users, instead, a single login is associated with an application.
* A system stored procedure is called when the application is launched that authenticates the application.
* Application users can then access the database through the application, but not directly.
* Application roles are not granted, and revoked like database roles.
* Application roles can provide permissions other than simply executing stored procedures, and can provide database access, however, access is provided only through a custom application.

Q40: You need to remove a column from a table. Which statement should you use?

**A: ALTER TABLE**

E: You should use the ALTER TABLE statement.

* This statement is used to modify a table’s structure, and allows you to add, remove, or modify table columns.
* To drop the salary column from the EMPLOYEES table, you could run the statement: ALTER TABLE employees DROP COLUMN salary;
* You should not use the DELETE statement, this is not used to remove columns, it is used to delete rows or records from a table.
* You should not use the DROP TABLE statement, this is not used to remove columns, dropping a table removes it from the database.
* You should not use the TRUNCATE TABLE statement, this is not used to remove columns, it is used to delete all records in a table.

Q41: You need to remove a table from your database. Which statement should you use?

**A: DROP TABLE**

E: You should use the DROP TABLE statement.

* This statement removes a table from a database; the DELETE TABLE statement is considered DDL (Data Definition Language), along with CREATE TABLE and ALTER TABLE statements.
* You should not use the TRUNCATE TABLE statement, this is used to delete all records in a table, you cannot choose a subset of records to delete.
* You should not use the UPDATE statement, this is not used to delete records, it is used to modify existing table records.
* You should not use the ALTER TABLE statement, this is not used to delete records, it is used to add, remove, or modify table columns.

Q42: You need to ensure that last\_name is not left blank. Which constraint should you use?

**A: NOT NULL**

E: You should use a NOT NULL constraint.

* A null value is unknown or undefined.
* If a record is added to the EMPLOYEES table, and last\_name is not specified, the field is considered NULL.
* You can prevent this by adding a NOT NULL constraint to the column.
* A FOREIGN KEY constraint enforces referential integrity, this ensures that relationships between tables remain valid, a FOREIGN KEY does not enforce NOT NULL.
* A CHECK constraint can evaluate column values against a simple expression, for example, a CHECK constraint on the BOOKS table could verify a book’s price is not over a certain value, a CHECK constraint does not enforce NOT NULL.
* A UNIQUE constraint prevents duplicates in a column, UNIQUE constraints do not enforce NOT NULL.

Q43: You need to change the datatype for a table column. You plan to use the ALTER TABLE statement. Which keyword(s) should you include?

**A1: MODIFY**

**A2: ALTER COLUMN**

E: You should include ALTER COLUMN.

* The ALTER TABLE statement is used to modify a table’s structure, and allows you to add, remove, or modify table columns.
* To modify a column’s datatype, you could issue a command like the following: ALTER TABLE employees ALTER COLUMN last\_name VARCHAR(100);
* You should not use the UPDATE statement, this is used to modify existing table records.
* You should not use the MODIFY clause, this is used by some database vendors as part of the ALTER TABLE command, however, the MODIFY clause is not part of the American National Standards Institute (ANSI) SQL standard, this means MODIFY is not ANSI-standard SQL syntax.
* You should not use the ADD keyword, this can be used in the ALTER TABLE statement to add a column to a table, this keyword is not used to modify a column’s datatype.

Q44: Which keyword must you include when redefining a view?

**A1: UPDATE**

**A2: ALTER**

E: You must include the ALTER keyword when redefining a view.

* When a view is modified, the previous view definition is overwritten, or replaced, with the new view definition.
* For example, to modify the previously created vw\_emp, you could issue this statement: ALTER VIEW view\_emp AS SELECT last\_name, dept\_id, manager\_id FROM employees
* You must not include the UPDATE keyword, this is used in the UPDATE statement to modify existing table records, not views.

Q45: Which keyword is used when removing a view from a database?

**A: DROP**

E: The DROP keyword is used when removing a view from a database.

* A view is a named query that displays a subset of columns and records from underlying tables, this allows you to provide users with only the data they need.
* To drop a view named view\_emp, you could execute: DROP VIEW view\_emp;
* The DELETE keyword is not used to remove views, you can use the DELETE statement to delete rows or records from a table.
* The TRUNCATE keyword is not used to remove views, the TRUNCATE statement deletes all records in a table.
* The ALTER keyword is not used to remove views, the ALTER VIEW statement can be used to modify a view.

Q46: You accidentally update data using a view, such that the modified records are no longer visible through the view. You need to prevent this from occurring in the future. What should you do?

**A1: Ensure the underlying table has a primary key.**

**A2: Modify the view and add a CHECK option**

E: You should modify the view and add a CHECK option.

* This ensures that any data manipulation using INSERT or UPDATE statements does not modify records in such a way that they are no longer selectable by the view.
* For example, if your view shows employees in department #20, adding a CHECK option would prevent you from changing the department number for any employee, your statement might look like:
* CREATE or ALTER VIEW view\_emp AS
* SELECT last\_name, dept\_id, manager\_id
* FROM employees
* WHERE dept\_id = 20
* WITH CHECK OPTION;
* You should not drop and re-create the view with the correct columns, this will not prevent data from being selected by the view, unless you include a check option in the new view definition.
* You should not add an inner join to the view definition, joins are used in queries to select data from multiple related tables.

Q47: Which keyword must be included for the statement below to execute successfully?

* CREATE VIEW view\_emp
* SELECT last\_name, dept\_id, manager\_id
* FROM employees;

**A1: WHERE**

**A2: AS**

E: You must include the AS keyword.

* In this statement, the AS keyword is required, if it is not included, the statement will throw a syntax error when executed.
* The correct statement is:
* CREATE VIEW view\_emp AS
* SELECT last\_name, dept\_id, manager\_id
* FROM employees;
* The ALTER keyword is not required, this is used when modifying an existing view.
* The WHERE keyword is not required, the WHERE clause allows you to specify a condition for a view.
* The TABLE keyword is not required, this is used when creating or altering tables.

Q48: Which keyword is required when creating a stored procedure?

**A1: EXEC**

**A2: AS**

E: The AS keyword is required when creating a stored procedure.

* Stored procedures are SQL code you can use to query and manipulate database tables.
* You could compare a stored procedure to a macro or a script.
* You can create a simple stored procedure using the following statement:
* CREATE PROC query\_emps AS
* SELECT \* FROM employees;
* The EXEC keyword is used to execute or run a stored procedure, such as:
* EXEC query\_emps;
* The PROCEDURE keyword is not required, you can use the PROC or the PROCEDURE keywords when creating a stored procedure.
* The WITH keyword allows you to specify additional procedure options, such as encrypting the procedure using “WITH ENCRYPTION”.

Q49: Which of the following is a valid stored procedure parameter?

**A: @myVariable**

E: @myVariable is a valid stored procedure parameter.

* Stored procedures are SQL code you can use to query and manipulate database tables, and parameters are variables you can supply to a stored procedure at runtime.
* &myVariable is not a parameter, the & operator is used to perform bitwise AND’ing operations; bitwise operations evaluate TRUE, FALSE, or NULL expressions, for example, TRUE & TRUE evaluates to TRUE.
* – – myVariable is not a parameter, two hyphens are used to identify comments, which are ignored by the SQL processing engine.
* %myVariable is not a parameter, % is the modulo arithmetic operator, when used in an expression, modulo returns the remainder from a division operation, for example, 5 % 2 returns 1.

Q50: You need a stored procedure that queries a table and returns a subset of table records based on an input value. What must your stored procedure include?

**A1: An EXEC statement**

**A2: A condition**

E: Your stored procedure must include a condition.

* Stored procedures are SQL code you can use to query and manipulate database tables, and parameters are variables you can supply to a stored procedure at runtime.
* In this scenario, your stored procedure will include a SELECT statement that queries a table, and a WHERE condition that filters records based on your input value, or parameter.
* The EXEC statement can be used to execute a stored procedure, it is not used to filter records based on an input value.
* Your stored procedure requires at least one parameter, which will be used to store your input value, you can specify the parameter when the stored procedure is executed.
* Constraints are used to ensure data and relationship integrity in a database, constraints are defined when a table is created or altered.

Q51: Which is required when creating a user-defined function in SQL?

**A1: An input parameter**

**A2: A RETURNS argument**

E: A user-defined function requires a RETURN clause.

* SQL supports built-in and user-defined functions.
* A user-defined function is similar to a stored procedure, but has more rigid requirements, for example, a user-defined function requires a RETURNS argument, this specifies a return value and the datatype for the returned value.
* An input parameter is not required in a user-defined function, however, like stored procedures, allowing a function to accept a parameter makes it more flexible.
* A SELECT statement is not required in a user-defined function, this is used to query a database.
* A WHERE clause is not required in a user-defined function, this is used in SELECT statements to filter records.

Q52: \_\_\_ dbo.calculate\_total (@sales\_price smallmoney) RETURNS smallmoney AS

BEGIN RETURN @sales\_price \* 1.08 END

Which is required to execute this statement successfully?

**A: CREATE FUNCTION**

E: The CREATE FUNCTION argument is required to complete this statement.

* SQL supports built-in and user-defined functions.
* A user-defined function is similar to a stored procedure, but has more rigid requirements.
* Among other requirements, a user-defined function requires a RETURNS argument, this specifies the datatype of the returned value, and a RETURN argument that provides the output the function will return to the caller.
* Stored procedures are SQL code you can use to query and manipulate database tables, stored procedure definitions do not include RETURNS, BEGIN, END, or RETURN arguments.
* In its simplest form, a view is a named SELECT statement, view definitions do not include RETURNS, BEGIN, END, or RETURN arguments.
* Tables are the basic building blocks of a database, and can be compared to spreadsheets; the CREATE TABLE statement does not include the arguments listed in the question.

Q53: You need to run the GetEmployees procedure to retrieve all employees in department #10. Which command should you run?

**A: EXEC GetEmployees @department = 10;**

E: You should run the EXEC GetEmployees @department = 10; command.

* The EXEC, or EXECUTE command, is used to call a stored procedure, if the procedure accepts parameters, you can specify those with a parameter name and a value.
* The SELECT \* FROM GetEmployees WHERE department = 10; is a query; the requirement states that you need to run the GetEmployees procedure, this query does not run the procedure.
* The SELECT dbo.getEmployees(10) AS employees; statement could be used to run a user-defined function named getEmployees.
* The RETURN @GetEmployees = 10; statement could be included in a function definition, it would not be used to run the GetEmployees procedure.

Q54: You are creating a stored procedure that accepts a UserID of type int, and retrieves a value that is the concatenated FirstName and LastName values for the user. Which parameter list should you use?

**A1: UserID int, FullName varchar(40) OUTPUT**

**A2: @UserID int, @FullName varchar(40) OUTPUT**

E: You need to pass in an input parameter, and retrieve a string using an output parameter.

* You define an output parameter by using the OUTPUT keyword, and a parameter name must begin with the @ character.
* You should not use the following parameter list: @UserID int; the only way to return a varchar from a stored procedure is to use an OUTPUT parameter.
* You can return an integer value using a RETURN statement, but you cannot return a varchar(40) value using the RETURN statement.

Q55: You need to create an index for a table. The index must store data rows in order by using the indexed column. Which type of index should you create?

**A1: Non-clustered**

**A2: Clustered**

E: You should create a clustered index.

* Indexes are designed to speed up database lookups.
* A clustered index sorts a table’s rows based on the values in the index key column; since a clustered index reorders the table’s rows, only one clustered index can be created for each table.
* A non-clustered index does not store rows; it is stored separately from the table, and holds a pointer to each indexed row.
* A hash index does not store table rows in order; it hashes key columns for storage in buckets, which can increase performance.
* A unique index is not required to store table rows in order; it enforces uniqueness on the key column.

Q56: You want to create a non-clustered index on the last\_name column of the EMPLOYEES table. How should you complete this statement:

CREATE INDEX idx\_last\_name

ON \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**A1: employees.last\_name**

**A2: employees (last\_name)**

E: You should use employees (last\_name) to complete the statement.

* Indexes are designed to speed up database lookups.
* When you define an index, you can specify the column(s) that will be included in the index.
* SELECT last\_name FROM employees, is a valid SQL query, however, it cannot be used to create a non-clustered table.
* You should not use employees.last\_name; in SQL, objects are associated with a schema, which is often a user.
* Schema references use dot (.) notation to specify the schema, and an object in the schema.
* For example, to reference the EMPLOYEES table in the sqladmin schema, you could use sqladmin.employees
* You should not use employees @last\_name; the @ symbol is used to identify variables or parameters in SQL.

Q57: You need to store a key value with a pointer to a record in a table. Which should you use?

**A: Non-clustered index**

E: You should use a non-clustered index.

* A non-clustered index stores a key value with a pointer to a record in a table.
* This is comparable to an index in a book, which is a pointer to a page number somewhere else in the book.
* Non-clustered indexes are stored separately from the tables that they index.
* You should not use a stored procedure, this is SQL code you can use to query and manipulate database tables.
* You should not use a user-defined function, functions perform operations on expressions, and return a value.
* You should not use a table view; views are named SELECT statements.

Q58: You have a database table named TransactionHistory that contains millions of rows. The table has a primary key named TrxID. Users frequently use a query that includes the region and status in the WHERE clause condition. You need to optimize performance of this query. What should you do?

**A: Create a composite non-clustered index on region and status**

E: A non-clustered index on columns frequently used in a WHERE clause can improve query performance.

* A non-clustered index can be created on an individual column, or multiple columns.
* An index created on multiple columns is known as a composite index.
* You should not create two clustered indexes, one on region and one on status.
* A table can only have one clustered index, because it represents the actual order of the data.
* When a primary key is created on a table, a clustered index is automatically created by default if one does not already exist.
* You should not create a composite clustered index on region and status; the table contains a primary key.
* When a primary key is created, a clustered index is created on the column, if no clustered index is already present.
* Because this table contains a primary key, it already has a clustered index, and more than one is not allowed.

Q59: SELECT @title, name AS full\_name, dept (department) FROM employees WHERE employees.dept = ‘IT’;

Which is a column alias?

**A: full\_name**

E: Column aliases can be used to change the default name for a column returned in a query.

* Column aliases only impact the immediate query, they do not modify the underlying database.
* @title is a variable or parameter; it is used to pass values to stored procedures at runtime.
* (department) in this context would be passed as a value to a function named dept; functions perform operations on expressions, and return a value.
* employees.dept is a column; you can use table.column to better qualify which column you are referencing; this is especially useful in cases where multiple tables share columns with the same name.

Q60: An EMPLOYEES table stores each employee’s monthly salary. You frequently run reports that require each employee’s annual salary. What should you do to include annual salary in the EMPLOYEES table definition?

**A1: Create an employee view.**

**A2: Add a computed column**

E: A computed column acts as a virtual column, that is based on a simple expression.

* In this scenario, the EMPLOYEES table stores each employee’s monthly salary in a column named monthly\_salary.
* Using this information, you could add a computed column to the table’s definition using a statement like: annual\_salary AS (monthly\_salary \* 12)
* A calculated field is like a computed column; however, a calculated field uses an expression in a query or SELECT statement, it is not used when creating or altering a table.
* Views are named SELECT statements that can be queried like a table; views are not used when creating or altering a table.
* Column aliases can be used to change the default name for a column returned in a query; column aliases only impact the immediate query, they do not modify the underlying database.

Q61: You have two database tables, LOAN and BORROWER. The BORROWER table contains 20 rows, and the LOAN table contains 60 rows. You execute the following statement:

SELECT BorrowerID, BorrowerName, LoanNumber FROM Borrower, Loan

How many rows will the query return?

**A1: 20**

**A2: 1,200**

E: When no JOIN is provided in a query, the query performs a CROSS JOIN and returns a Cartesian product.

* A Cartesian product includes a row in the result for each possible combination of results from the other tables.
* The number of rows returned equals the number of rows in the first table (BORROWER) multiplied by the number of rows in the second table (LOAN).
* Sometimes, Cartesian products are used for unique purposes, such as generating test data, however, they should be used with caution because they may return extremely large result sets.
* The query will not return 20, 60, or 120 rows; because no JOIN condition is included, a Cartesian product containing 1,200 rows will be returned.

Q62: SELECT LoanNumber, OriginalLoanAmt, InterestRate FROM Loan WHERE BorrowerID IN

(SELECT BorrowerID WHERE BorrowerState = ‘TX’)

What is this an example of?

**A: A subquery**

E: A subquery, sometimes called an inner query, is a query that is included within another query.

* First, the subquery executes, and returns a result to the containing, or outer, query.
* The outer query can use a subquery wherever an expression is allowed, such as in a WHERE or HAVING clause condition.
* Subqueries can return a single value, or multiple values.
* However, the outer query must use the appropriate type of operator.
* For this query, first, the inner query executes, returning multiple BorrowerID values for all the borrowers with a BorrowerState = ‘TX’.
* Because this result could contain multiple values, the outer query must use an operator that is capable of accepting multiple values, such as the IN operator.
* Attempting to use an operator such as =, which requires a single value, would result in an error.
* This statement is not an example of a CROSS JOIN; this returns a result set that contains all the combinations of rows from two tables that do not have a column in common, also known as a Cartesian product.
* This statement is not an example of a trigger; this is a database object that fires, or executes, in response to an event, such as when a DML (Data Manipulation Language) statement, or DDL (Data Definition Language) statement is executed.

Q63:

| emp\_id | l\_name | job\_title | dept | hire\_date |
| --- | --- | --- | --- | --- |
| 121 | FELD | CPA | 50 | 2011-10-24 |
| 128 | PEARL | SR CPA | 50 | 2015-12-22 |
| 152 | OTOZ | CPA | 50 | 2021-01-04 |
| 140 | YEN | LAB MGR | 40 | 2019-02-16 |
| 105 | PAUL | ADMIN | 20 | 2003-07-11 |
| 115 | CHULE | IT MGR | 20 | 2009-09-27 |

Which clause was used to produce this output?

**A: ORDER BY dept DESC**

E: The ORDER BY clause can be added to a SELECT statement to sort output.

* By default, ORDER BY sorts in ascending order (ASC); you can sort in descending order by using the DESC argument:
* SELECT emp\_id, l\_name, job\_title, dept, hire\_date FROM employees ORDER BY dept DESC;
* The GROUP BY clause is used to apply aggregate functions, such as COUNT() or AVG(), on groups of records.
* The WHERE clause is used to filter records; WHERE dept NOT IN (20, 40, 50) will display all rows except for those in departments 20, 40, or 50.
* The FROM clause is used to specify source tables, or views for a SELECT statement; column names should be listed in the SELECT clause.

Q64: You need to filter records before a group function is applied. Which of the following should you use to complete this requirement?

**A1: GROUP BY**

**A2: WHERE**

E: The WHERE clause is used to filter rows, before the group function is applied.

* This means that only the records that meet the condition in the WHERE clause will be processed by the group function; this is because the WHERE clause is evaluated by the SQL engine, before the GROUP BY clause and the aggregate function are processed.
* A GROUP BY clause tells the database how records should be grouped or aggregated, before the group function is applied; a GROUP BY clause is not used to filter records.
* The HAVING clause is used to filter records after a group function has been applied; it does not filter records before the function is processed.
* The IN operator can be used in the WHERE clause to specify a list of items, only rows that meet one of the list items will meet the condition.

Q65: SELECT emp\_id, l\_name, job\_title, dept, hire\_date FROM employees ORDER BY dept, 5 DESC;

Which statement describes the output of this query?

**A1: Results will be sorted for all employees in department 5 in descending order.**

**A2: Results will be sorted by department in ascending order, then by hire date in descending order.**

E: The results will be sorted by department in ascending order.

* This is because dept is listed first in the ORDER BY clause, and by default, ascending order (ASC) is used if no sort order is specified.
* Since the ORDER BY clause includes another column, results will then be further sorted; in this case, instead of using a column name, a column position is used, based on the order of columns in the SELECT clause, so the 5th position is hire\_date.
* Results will not be sorted for all employees in department 5 in descending order; to filter rows in a SELECT statement, a WHERE clause is required.
* The first 5 rows of results will not be sorted in descending order; one way to use different sorts on the same underlying table would be to use the UNION operator.
* The results will not be sorted by department, and then by each of the 5 columns separately; to sort by a column, it must be listed in the ORDER BY clause by name or positional reference.

Q66: You need to return a list of employees whose names contain the letter A. Which operator should you use?

**A: LIKE ‘%A%’**

E: The LIKE ‘%A%’ operator will return a list of employees whose names contain the letter A.

* You can use the LIKE operator in the WHERE clause of an SQL statement.
* This allows you to filter records based on a partial match, using wildcards.
* When used with the LIKE operator, the percent sign (%) represents zero to many characters.
* In this scenario, LIKE ‘%A%’ will match any string with the letter A anywhere in the string, including as the first or last character.
* The LIKE ‘A%’ operator will match any string where the first character is the letter A.
* The LIKE ‘%A’ operator will match any string where the last character is the letter A.
* The LIKE ‘A’ operator will only match strings with a single character, the letter A.

Q67: Which query should you run, in order to return a list of departments where the second character is the letter N?

**A: SELECT dept\_id, dname FROM dept WHERE dname LIKE ‘\_N%’**

E: This query will return a list of departments, where the second character of the department name is the letter N.

* You can use the LIKE operator in the WHERE clause of an SQL statement.
* This allows you to filter records based on a partial match, using wildcards.
* When used with the LIKE operator, the percent sign (%) represents zero to many characters, and the underscore (\_) represents a single character.
* SELECT dept\_id, dname FROM dept WHERE dname IN (‘ N’, ‘N ‘) will return departments whose first characters is a space, and the second character is N, or three-character department names where the first and last characters are spaces.
* SELECT dept\_id, dname FROM dept WHERE dname LIKE ‘\*N%’ will return all departments names that begin with \*N.
* SELECT dept\_id, dname FROM dept WHERE dname LIKE ‘ N%’ will return all departments names that begin with a blank space, followed by the letter N.

Q68: You need to return all employees who earn a salary from $3,000 to $5,000, inclusive. Which query should you use?

**A: SELECT name, salary FROM emp WHERE salary BETWEEN 3000 AND 5000;**

E: You can use the BETWEEN operator in the WHERE clause to filter records based on a range of values.

* The BETWEEN operator works on dates, numbers, or text, and the lower & upper bounds are included in the results.
* In this scenario, this means that employees earning $3,000, $5,000, or any salary in between, will be returned.
* SELECT name, salary FROM emp WHERE salary > 3000 AND salary < 5000, will return employees earning $3,001 to $4,999; the lower & upper bounds will be excluded.
* SELECT name, salary FROM emp WHERE salary IN (3000, 5000), will return employees earning exactly $3,000 to $5,000; the IN operator is used to match a list of values.
* SELECT name, salary FROM emp WHERE salary < 5000 OR salary > 3000, will return all values; with an OR operator, only one condition must be met for a row to be returned.

Q69: You suspect some of the records in the EMPLOYEES table were added without specifying a salary. To locate these records, how should you complete the following query?

SELECT emp\_id, last\_name, job\_title, salary FROM employees

**A: WHERE salary IS NULL**

E: A NULL value in a database is unknown.

* This means that it is not equal to anything.
* To locate null values, you must use the IS NULL operator.
* If you want to search for non-null values, you can negate the statement by using the NOT keyword, IS NOT NULL.
* You cannot test for null values using the = operator; WHERE salary = 0 would only return records where the salary had been entered as a value, 0.
* The WHERE NOT salary clause needs an additional operator to be complete; for example, to complete the query as requested, you could write: WHERE NOT salary IS NOT NULL; this is not the recommended approach, as the logic can be difficult to understand, but it will return all null salaries.
* The WHERE salary = ‘ ‘ will return salaries with a blank space, which is not null.

Q70: SELECT TOP 2 salary FROM employees;

What will this query return?

**A: The first two salaries in the employees table**

E: You can use the TOP operator to return a fixed number of rows from a table.

* This is particularly useful on large tables, where you want to preview a subset of records.
* The TOP operator supports a fixed number, as in this scenario, or a percentage of rows with the PERCENT keyword.
* As written, this query does not return the two highest salaries in the employees table; however, if you add an ORDER BY clause to the statement, the records will be ordered before the top 2 records are returned; for example, ORDER BY salary DESC.
* As written, this query does not return the two lowest salaries in the employees table; however, you could accomplish this by adding ORDER BY salary to the query.
* As written, this query does not return the top 2% of salaries from the employees table; you would need to change your SELECT clause to SELECT TOP 2 PERCENT salary.

Q71: Your query returns the results below. You do not want records for employees in departments 40 or 50. Additionally, this query should work, even if new employees are added to new departments. Which WHERE clause should you use?

empID last\_name dept

105 PAUL 20

115 CHULE 20

121 FELD 50

128 PEARL 50

140 YEN 40

152 OTOZ 50

**A1: WHERE dept NOT BETWEEN 40 AND 50;**

**A2: WHERE dept <> 40 AND dept <> 50;**

E: The <> operator means not-equal.

* You can use the AND operator if you require multiple conditions that must evaluate to true.
* In this scenario, each row is evaluated against both conditions, and only rows that meet both conditions are included in the results.
* WHERE dept NOT BETWEEN 40 AND 50 will filter out departments 40 & 50; however, it will also filter out departments 41 through 49; the BETWEEN operator is used to filter ranges of values.
* WHERE dept NOT IN (20) will filter out all departments, except department 20; the IN operator is used to compare a list of values.
* WHERE NOT dept = 40 OR dept = 50 will filter records for department 40, but will not filter records for department 50; this is because the NOT operator only applies to the first condition in the statement; you could make this clause work by adding parentheses ( ), so the NOT operator applies to both conditions: WHERE NOT (dept = 40 OR dept = 50).

Q72: SELECT empID, last\_name, hire\_date FROM employees

WHERE hire\_date BETWEEN ‘10/24/2011’ AND ‘2/17/2019’;

Which WHERE clause will return the same results?

**A: WHERE hire\_date >= ‘10/24/2011’ AND hire\_date <= ‘2/17/2019’;**

E: When you use the BETWEEN operator to provide a range, the boundaries are inclusive; you can match a BETWEEN operator using >= with your lower bound, and <= with your upper bound, combined with the AND operator to provide the same results.

* WHERE hire\_date IN (‘10/24/2011’, ‘2/17/2019’) returns any records where the hire date is exactly 10/24/2011 or 2/17/2019; the IN operator is used to filter based on a list of values.
* WHERE hire\_date > ‘10/24/2011’ AND hire\_date < ‘2/17/2019’ returns most of the same rows as the BETWEEN operator; however, the bounds are excluded, which means records with hire dates 10/24/2011 or 2/17/2019 will not be returned.
* WHERE hire\_date <> ‘10/24/2011’ AND hire\_date <> ‘2/17/2019’ will return all records, except those with hire dates 10/24/2011 or 2/17/2019; the <> operator means not-equal.

Q73: You plan to use a subquery in your WHERE clause to return a list of product IDs found in invoices. Based on the result, you need to determine the current stock level for each product based on product ID. Which of the following should you use in your WHERE clause to meet this requirement?

**A1: =ALL**

**A2: =ANY**

E: A subquery is an SQL statement embedded within a parent query.

* You can use subqueries in various SQL clauses, including in the WHERE clause.
* In this scenario, your parent query will evaluate each record in the products table, against a list of product IDs returned from your subquery; =ANY will match any result returned from the subquery.
* You can use the BETWEEN operator in the WHERE clause, to filter records based on a range of values; however, you can only perform one-to-one comparisons with the BETWEEN operator, you cannot compare against a list results returned from a subquery.
* The OR operator allows you to check multiple conditions in a WHERE clause; only one condition must evaluate to true for a record to be returned.
* =ALL works with subqueries like =ANY; however, the current record must match all of the results returned by the subquery.

Q74: You have employees in departments 20, 30, 40, and 50. You want to return only records for employees in departments 30 and 40. How will you complete this statement?

SELECT emp\_id, l\_name, job\_title, dept, hire\_date

FROM employees

\_\_\_\_\_ dept \_\_\_\_\_ (20, 50);

**A: WHERE dept NOT IN (20, 50)**

E: The WHERE clause is used to filter the records returned by a query.

* In this scenario, you can either explicitly include records from departments 30 & 40, or you can filter out records from other departments.
* The NOT keyword is used to negate the list provided in parentheses with the IN operator.
* This directs the SQL engine to gather all the records from employees, and remove records where dept is 20 or 50.
* The <> operator means not-equal; the operator cannot process a list of values like the IN operator can; to use <> in this query, you could write the WHERE clause like this: WHERE dept <> 20 AND dept <> 50
* The HAVING clause is used to filter records based on the output of a group function.
* The GROUP BY clause is used to aggregate records before a group function, such as MIN or AVG, is applied.

Q75: You need to return a list of genres where the average book price is more than $5. Which two clauses should you add to the following query?

**A1: WHERE AVG(price) > 5.00**

**A2: GROUP BY genre HAVING AVG(price) > 5.00**

E: This query calls the AVG( ) group function for the price column.

* A GROUP BY clause tells the database how records should be grouped, or aggregated, before the group function is applied; in this case, you want the average price by genre, so you need to group by genre.
* When filtering the results of a group function, you must use the HAVING clause, you can then use a group function as a filter.
* You cannot filter the output of a group function using a WHERE clause; the WHERE clause only processes rows before the group function is applied.
* In this scenario, you are filtering based on an average book price for each genre.
* The WHERE clause is evaluated by the SQL engine, before the GROUP BY clause and the aggregate function are processed.
* You are not grouping by price or (AVG) price, you are grouping by genre and then looking at the average price for all books in that genre.

Q76: You need to know how many employees receive a salary. Which function should you use?

| empID | first\_name | last\_name | job\_title | dept | hire\_date | sal |
| --- | --- | --- | --- | --- | --- | --- |
| 105 | DEWANE | PAUL | ADMIN | 20 | 2003-07-11 | 8600 |
| 115 | MINNA | CHULE | IT MGR | 20 | 2009-09-27 | 9000 |
| 121 | GREN | FELD | CPA | 50 | 2011-10-24 | null |
| 128 | MIKKA | PEARL | SR CPA | 50 | 2015-12-22 | 8200 |
| 140 | CHIN | YEN | LAB MGR | 40 | 2019-02-16 | 4500 |
| 152 | PLANK | OTOZ | CPA | 50 | 2021-01-04 | null |

**A: COUNT(sal)**

E: The COUNT(sal) function will tell you how many employees receive a salary.

* COUNT( ) is a group, or aggregate function; this means it operates on groups of records.
* Importantly, the COUNT( ) function ignores NULL, or empty values.
* If you only plan to return a single grouping for the entire table, a GROUP BY clause is not required.
* COUNT(empID) will tell you how many employees you have; this will include employees who have an empID, but do not have a salary defined (NULL).
* SUM(empID) will perform simple addition on empID values, and will return a single sum for all values.
* SUM(sal) will return a sum of all non-null salaries.

Q77: You need to list the total number of employees in each department, based on dept\_id. Which query should you run?

**A: SELECT dept\_id, COUNT(\*) FROM emp GROUP BY dept\_id;**

E: When you want to apply a group function to a column, you include that column in the SELECT clause as well as the group function you want to apply.

* You also need a GROUP BY clause to tell the database to aggregate records before applying the group function.
* The COUNT(\*) specifies a wildcard that counts all records in the grouping and can include records with null values.
* SELECT COUNT(dept\_id) FROM emp WHERE dept\_id > 0, will return a single value; this will count all dept\_id entries that have a dept\_id greater than 0.
* SELECT dept\_id FROM emp GROUP BY dept\_id, will return a list of unique dept\_id’s from the emp table.
* SELECT dept\_id AS total\_employees GROUP BY emp, is missing a FROM clause and will return a syntax error; the AS keyword allows you to return a column alias in place of a column name.

Q78: SELECT TOP 1 salary FROM employees ORDER BY salary;

Which statement gives the same result as the above statement?

**A1: SELECT MAX(salary) FROM employees;**

**A2: SELECT MIN(salary) FROM employees;**

E: SELECT MIN(salary) FROM employees; returns the same results.

* The query in the question will sort the records in ascending order, based on salary and then select the top, or first, record, which will be the lowest salary in the table.
* You could accomplish the same by selecting the minimum salary using the MIN( ) group function.
* SELECT MAX(salary) FROM employees will return the highest salary in the employees table.
* SELECT AVG(salary) FROM employees will group all the salaries, and calculate the average salary.
* SELECT COUNT(salary) FROM employees will return the number of non-null salaries in the employees table.

Q79: You write a query that returns the number of books written by each author. You want to display only authors who have written at least 10 books. What should you add to your query?

**A1: WHERE**

**A2: HAVING**

E: The HAVING clause will allow you to display only authors who have written at least 10 books.

* When filtering the results of a group function, you must use the HAVING clause; you can then use a group function as a filter.
* In this scenario, your query might look like this:
* SELECT author, COUNT(title) FROM books
* GROUP BY author HAVING COUNT(title) >= 10;
* The IN operator is used to compare a list of items, it is not used to filter the output of a group function.
* You can use the BETWEEN operator in the WHERE clause to filter records based on a range of values, you cannot use BETWEEN to filter the output of a group function.
* The WHERE clause is used to filter records based on one or more conditions; the WHERE clause can filter the input to a function, but cannot filter the output of a group function.

Q80: SELECT COUNT(isbn) FROM books;

Which statement should you run to find the missing 3 records?

**A: SELECT title, isbn FROM books WHERE isbn IS NULL;**

E: The original query returns a count of books with a non-null isbn.

* You can deduce that 3 records must have null isbn values, and you can locate those records by using IS NULL in the WHERE clause.
* The key is understanding that all group functions ignore null values, except for COUNT(\*).
* SELECT COUNT(\*) FROM books will return a count of rows in the books table, unlike other group functions, COUNT(\*) will count rows with null values.
* SELECT COUNT(isbn) FROM books HAVING COUNT(isbn) > 0 should return the same count as the original query, however, it will not help you find the missing 3 results.
* SELECT TOP 3 isbn FROM books ORDER BY isbn DESC will return the three numerically highest isbn’s, however, it will not help you find the missing records.

Q81: You need to construct a query that returns the total salaries for each job title in each department. Which approach should you use?

**A1: List the department and job title columns in the HAVING clause**

**A2: Include the department and job title columns in the SELECT clause**

E: Group functions can operate across more than one column.

* You just need to be sure you include both columns in the GROUP BY clause.
* You could run a query like the following:
* SELECT department, job\_title, SUM(salary) FROM employees
* GROUP BY department, job\_title;
* You should not use the JOIN operator to join a department query with a job title query, the JOIN operator is used to combine results from related tables.
* You do not need to list the department and job title columns in the HAVING clause, this is used to filter the output of a group function.
* You should not include a GROUP BY clause for each column separately, a single GROUP BY clause can include multiple columns.

Q82: Which group function has the option to ignore null values?

**A1: SUM**

**A2: COUNT**

E: The COUNT group function has the option to ignore null values.

* A null value is unknown, or not specified, and using COUNT with an asterisk ignores null values in a row.
* For example, SELECT COUNT(\*) FROM employees would return the total number of rows in the employees table, including rows with null values.
* The MIN function does not ignore null values, MIN returns the minimum value in a column and works on numeric, character, and date values.
* The SUM function does not ignore null values, SUM returns the total for a numeric column.
* The AVG function does not ignore null values, AVG returns the average value for a numeric column.

Q83: Your employees table contains information about employees, including each employee’s first and last name stored as VARCHAR.

What does the following query return?

SELECT SUM(last\_name) FROM employees;

**A: A data type error**

E: The query returns a data type error.

* The SUM operator works with numeric data, not character data like VARCHAR.
* SUM returns the total for all non-null numeric values in a column.
* To return the number of rows with a non-null last name, you could run:
* SELECT COUNT(last\_name) FROM employees;
* You could use the LEN function to return the length of each last name:
* SELECT LEN(last\_name) FROM employees;
* To return the number of characters for all last names, you could use a subquery in the FROM clause.

Q84: Which two statements regarding the output of this query are correct?

**A1: Total salaries for department 20 will not be included, and total salaries for each department must be at least $10,000.**

**A2: Total salaries for department 20 will not be included, and only salaries above $5,000 will be included in each department’s total.**

E: The WHERE clause filters out salaries for department 20 using the NOT operator.

* Only salaries above $5,000 will be included in each department’s total; the WHERE clause filters out salaries less than or equal to $5,000, so each department’s sum will not include these salaries.
* Total salaries for each department must be at least $10,001, the > operator in the HAVING clause excludes total salaries of $10,000 or lower.
* Salaries for department 20 will be filtered by the WHERE clause before the group function is applied, no salaries for department 20 will be included in the sum.
* Only one record will be returned for each department, group functions will group records based on the column you specify.

Q85: You need a report that shows the average salary for each job title by department ID.

How should you complete the query?

**A1: SELECT dept\_id, job\_title, AVG(salary) AVG\_SAL FROM employees GROUP BY dept\_id, job\_title;**

**A2: SELECT dept\_id, job\_title, AVG(salary) AVG\_SAL FROM employees GROUP BY dept\_id, job\_title;**

E: You can use the AVG function to calculate the average salary for each job by department.

* You need to include both columns in the SELECT clause, along with the AVG function referencing the salary column.
* In this query, AVG\_SAL is a column alias, with group functions, if you do not specify a column alias, the computed column will appear without a name in your results.
* As a rule, any column you include in your SELECT clause that is not part of a group function, must also be listed in your GROUP BY clause.
* Otherwise, the statement will throw an error; if the only columns you list in the SELECT clause are part of group functions, a GROUP BY clause is not required.
* For example, you could run the statement:
* SELECT AVG(salary), MIN(salary), MAX(salary) FROM employees;

Q86: You need a query that returns the department number and average salary for all departments where the average salary is more than $6,000.

**A: SELECT dept, AVG(sal) FROM employees GROUP BY dept HAVING AVG(sal) > 6000**

E: The SELECT clause specifies the data you want, in this case, you use the average, AVG, aggregate function to calculate the average salary.

* A GROUP BY clause tells the database how records should be grouped, or aggregated, before the group function is applied, in this case, you want the average salary by department, so you need to group by department.
* When filtering the results of a group function, you must use the HAVING clause, you can then use a group function as a filter, in this case, the average salary must be greater than 6000.

Q87: SELECT TOP 20 OrderID, SUM(LineTotal) AS OrderTotal FROM OrderDetails

GROUP BY OrderID

GROUP BY OrderTotal DESC

What is the result?

**A1: A syntax error because the ORDER BY clause is incorrect.**

**A2: The 20 orders with the highest total**

E: The result is the 20 orders with the highest total.

* The GROUP BY clause allows you to perform an aggregate calculation over rows that have a matching value in a column.
* In this case, the statement groups all rows associated with a specific order, and sums the values in the LineTotal column for each order.
* The ORDER BY clause is specified after the GROUP BY clause, and orders the results in descending order by the total summed value, by including the alias for the summed value and the DESC keyword.
* The ORDER BY clause can only reference aggregate values, identified with or without an alias, or a column referred to in the GROUP BY clause.
* Then, the TOP clause returns the first 20 rows in the ordered result, which in this case, would be the 20 orders with the highest total.
* The result is not the 20 orders with the lowest total; to return the 20 orders with the lowest total, you would omit the DESC keyword in the ORDER BY clause to use a default ascending sort order, or use the ASC keyword to specify a sort from lowest to highest.

Q88: You run the following statement:

CREATE TABLE departments

(

Department\_id int NOT NULL,

Name varchar(255) NOT NULL,

Location varchar(255)

);

You need to add a row to your new departments table. Which statement should you run?

**A: INSERT INTO departments VALUES (10, ‘SALES’, ‘NYC’);**

E: The INSERT INTO statement requires the name of the destination table, the VALUES keyword, and a value for each column in the same order as the table specification.

* This means if department\_id is the first column in the table, you must provide the department\_id first in your values list.
* You can circumvent this requirement by including a list of columns after the table name in the INSERT INTO statement.
* INSERT INTO departments (10, ‘SALES’, ‘NYC’) is missing the VALUES keyword, and this will return a syntax error.
* You can use a subquery in an INSERT INTO statement to insert rows from another table; however, you must specify the source table name and ensure you match the data types for the columns you are populating.
* INSERT VALUES INTO departments (10, ‘SALES’, ‘NYC’) puts the VALUES keyword before the table name, and this will return a syntax error.

Q89: What will be the result of this statement?

INSERT INTO departments (department\_id, location) VALUES (100, ‘ROME’);

**A1: A row is added.**

**A2: A constraint violation error is returned.**

E: The statement will return a constraint violation error.

* The table specification includes NOT NULL constraints for the department\_id and name columns.
* This means that an INSERT statement must include values for both columns.
* In this scenario, a department\_id is specified, but a name is not; as a result, the NOT NULL constraint will prevent the record from being added.
* A row will not be added unless values are provided for both the department\_id and name columns.
* The INSERT statement is not used to modify rows, this is done using the UPDATE statement.
* The syntax for the statement is correct, as long as the constraints are met, the statement will execute without errors.

Q90: You create a table for testing using the following statement:

CREATE TABLE test\_emp

(

empID int,

Dept int

)

You need to populate the table with records from the employees table.

**A: INSERT INTO test\_emp**

**SELECT emp\_id, dept FROM employees;**

E: You can use a subquery in an INSERT INTO statement to insert rows from another table.

* To do so, you must specify the source table name and ensure you match the position and data types for the columns you are populating.
* Your subquery can be as complex as needed, and can include a WHERE clause, computed columns, and even JOIN statements; be sure you do not violate any constraints in the destination table.
* In this scenario, the source table is employees, and the destination table is test\_emp.
* You are selecting the empID and dept columns from employees for insertion into the test\_emp table.
* As a WHERE clause is not included, all rows in the employees table will be selected.

Q91: What statement will correctly run?

**A1: SET IDENTITY\_INSERT Members OFF**

**GO**

**INSERT INTO Members (MemberID, FirstName, LastName) VALUES**

**(75, ‘Renee’, ‘Smith’)**

**SET IDENTITY\_INSERT Members ON**

**A2: SET IDENTITY\_INSERT Members ON**

**GO**

**INSERT INTO Members (MemberID, FirstName, LastName) VALUES**

**(75, ‘Renee’, ‘Smith’)**

**SET IDENTITY\_INSERT Members OFF**

E: By default, you cannot specify a value for an IDENTITY column within an INSERT statement.

* However, you can override this behaviour by using SET IDENTITY\_INSERT, to turn the identity insert option ON.
* After you issue the INSERT statement, you should turn identity insert OFF, because only a single table can be enabled for identity insert within a session.

Q92: You need to add rows to the ReorderList table that includes the ProductID, the current date, and a Quantity of 100 for all products that have a Quantity of 0 in the ProductInventory table.

Which statement should you use?

**A: INSERT INTO ReorderList (ProductID, ReorderDate, Quantity)**

**SELECT ProductID, getdate( ), 100**

**FROM ProductInventory**

**WHERE Quantity = 0**

E: The INSERT statement allows you to use a query as the source for the rows.

* To do so, you specify the query instead of a VALUES list.
* In this case, you need to select the ProductID from the source table, use the getdate( ) function to provide the current date, and use a literal value for the Quantity column in the ReorderList table.

Q93: Which statement will successfully insert data into the Product table?

CREATE TABLE Product

(

ProdID int PRIMARY KEY,

CategoryID int NOT NULL,

Description varchar(35),

QtyOnHand int CHECK (QtyOnHand > 0 AND QtyOnHand <= 50)

)

**A1: INSERT INTO Product VALUES**

**(101, NULL, ‘Outdoor Extension Cord’, 25)**

**A2: INSERT INTO Product VALUES**

**(101, 1, ‘Outdoor Extension Cord’, 25)**

E: The INSERT statement is used to insert rows into a table.

* The VALUES clause includes the data values to be inserted.
* The INSERT statement can include a column list, after the name of the table to insert data into only selected columns.
* If you do not include a column list, then you must specify a value in the VALUES clause for each column in the table, in the order that the columns are defined.
* All columns that are defined as a primary key must be assigned a unique value in the VALUES clause, unless they include the IDENTITY property.
* If you attempt to insert a row with a primary key value that already exists in the table, an error is returned indicating that the PRIMARY KEY constraint has been violated.
* Also, all columns that have a NOT NULL constraint must be assigned a value, unless they include a DEFAULT definition.

Q94: Which of the following is a required clause when issuing an UPDATE statement?

**A: SET**

E: The SET clause is a required clause.

* You specify the columns or variables that should be updated and, when a FROM clause is not also used, their values.
* A FROM clause is not required, it is used when updating columns or variables with values retrieved from another table or view.
* An OUTPUT clause is not required, it is used to retrieve information about the rows that were modified by the statement.
* A WHERE clause is not required, it is used to limit the rows affected by the UPDATE statement to those that meet specific criteria.

Q95: When would you use a .WRITE clause within an UPDATE statement?

**A: To update the value in a column that has the varchar(max) data type.**

E: You use a .WRITE clause in an UPDATE statement when you need to update the values of columns that have a large object data type, such as varchar(max) or varbinary(max) columns.

* You use the FROM clause to update columns with values in a different table.
* You use a SET clause to update columns with calculated values.
* You use a COMMIT statement to commit changes made within a manual transaction to the database.
* There is no need to issue any statement to commit changes made within automatic transactions, they are committed automatically.

Q96: Which keyword should you include in an UPDATE statement to set a column’s value equal to a value that is stored in another table?

**A1: LIKE**

**A2: FROM**

E: You should include the FROM keyword.

* An UPDATE statement can be used to modify one or more column values for rows in a table.
* The UPDATE statement can explicitly specify the values to which the columns should be updated, either as constants or expressions, or can retrieve values from another table or view to use, to perform the update.
* To obtain values from other tables or views, you include the FROM clause with an optional WHERE clause after the SET clause in your UPDATE statement.
* Then, you can reference the information retrieved in your SET clause to update the original table.
* If you include a table alias in the FROM clause, then it must be used when you reference columns in the table.
* If no alias is specified, you can reference the column using the table name instead.
* You cannot include the INTO keyword; it is not used with an UPDATE statement.
* The INTO keyword is used with a SELECT statement to create a new table that includes the rows returned by the SELECT statement.
* It is also an optional keyword in the syntax of an INSERT statement.

Q97: Which statement will increase the CreditLimit for the customer with a CustID of 350 by 5%?

**A: UPDATE Customer SET CreditLimit = CreditLimit \* 1.05 WHERE CustID = 350**

E: The UPDATE statement is used to update one or more rows in a table or view.

* The SET clause includes the columns to be updated; each column to be updated along with an assignment operator (=), and the value to which the column should be updated is included.
* Multiple columns may be included, separated by commas, to update more than one column value in a table.
* The value on the right side of the assignment operator can be a constant, or an expression.
* If you want to update a column using information from another table, you can also use a subquery as long as it returns a single value.
* The WHERE clause restricts which rows are updated.
* In this case, the statement sets the CreditLimit column equal to the current value multiplied by 1.05, which effectively increases the credit limit by 5%.
* This statement includes the WHERE clause condition, to ensure that only the customer with a CustID value of 350 is updated.

Q98: What happens if you issue a DELETE Employees statement, and omit the WHERE and TOP clauses?

**A1: The Employees table is deleted**

**A2: All rows of the Employees table are deleted**

E: Issuing the statement causes all rows of the Employees table to be deleted.

* The DELETE statement is a DML (Data Manipulation Language) statement that deletes one or more rows from a table.
* When the DELETE statement is not qualified by a WHERE clause, or by the TOP keyword, all rows in the table are deleted.
* The Employees table itself is not deleted, only the data within it; to delete the Employees table, you would need to use the DROP TABLE statement, a DDL (Data Definition Language) statement.
* The Employees database is not deleted, you need to use the DROP DATABASE statement, a DDL (Data Definition Language) statement.
* Issuing the statement does not cause only the first row to be deleted; to delete only the first row, you would use the following T-SQL (Transact-SQL) query: DELETE TOP (1) Employees
* Issuing the statement does not cause no rows to be deleted; no rows would be deleted if the statement included a WHERE clause, but none of the rows met the WHERE clause condition.

Q99: Which statement will remove all rows from the Materials table that have a Status value of ‘Obsolete’ but do not have a value for the VendorID column?

**A: DELETE FROM Materials WHERE Status = ‘Obsolete’ AND VendorID IS NULL**

E: The DELETE statement is used to remove one or more rows from a table.

* Following the DELETE keyword, you can specify the FROM keyword, but it is not required.
* The DELETE or DELETE FROM is then followed by the name of the table or view.
* A WHERE clause controls which rows are deleted; only the rows that meet the WHERE clause condition are removed from the table.
* If you do not include a WHERE clause, all rows in the table are deleted; in this case, this statement includes a compound condition in the WHERE clause that will delete all rows that have both a Status value of ‘Obsolete’ and no VendorID value.

Q100: You need to ensure that if a row in a parent table is deleted, the related child rows in another table are also deleted.

What should you use?

**A: ON DELETE CASCADE**

E: You should use ON DELETE CASCADE.

* To ensure that all rows in a child table are deleted if the related parent row is deleted, you can specify the ON DELETE CASCADE clause for your FOREIGN KEY constraint.
* You should not use ON DELETE NO ACTION; this would prevent a parent row from being deleted if it had related child rows in another table.
* A WHERE clause allows you to control which rows in a given table are deleted, but would not control how related rows in another table are affected.
* You should not use an OPTION clause in the DELETE statement; the OPTION clause is used to specify query hints that will allow for optimization of a query’s execution.

Q101: Later, after employees have been added to the table, you attempt to drop the departments table and receive an error.

What should you do to troubleshoot this issue?

**A1: Verify none of the department\_id values appear in the departments table.**

**A2: Disable or delete the FOREIGN KEY constraint on the employees table.**

E: Foreign keys enforce data integrity by ensuring relationships between tables remain valid.

* If you drop the departments table, employees will be in departments that no longer exist, this violates the foreign key constraint; to prevent this, you must either disable or delete the foreign key.
* Deleting all the rows in the employees table will not solve the problem; the foreign key creates a link between the tables, and is not dependent on rows existing in another table; additionally, if you attempt to run the DROP statement again, you will receive the same error.
* The foreign key will still be enforced even if none of the department\_id values appear in the employees table.
* The TRUNCATE TABLE statement is used to delete all the rows in a table, this will not allow you to drop the departments table.

Q102: What should you do to troubleshoot the error?

**A1: Try creating the employees table before creating the departments table.**

**A2: Provide a name for the PRIMARY KEY constraint in the employees table.**

E: PRIMARY KEY constraints enforce uniqueness and not-null.

* You can define a PRIMARY KEY inline, as is done on the departments table, or at the table level.
* If you choose the latter approach, you must also supply a name for the PRIMARY KEY.
* The last line of the CREATE TABLE statement could be written:
* CONSTRAINT [PK\_employees] PRIMARY KEY (empID)
* If you try creating the employees table before creating the departments table, you will receive an error; you cannot define a FOREIGN KEY that points to a table that does not exist.
* Column names do not need to be unique between tables, if you need to use both columns in the same query, you can use dot notation (.), such as employees.department\_id and departments.department\_id.
* You should not move the FOREIGN KEY from the employees to the departments table; it makes sense that employees are added to departments that exist, this is referential integrity.

Q103: You need to remove the location column and run the following command, which returns an error:

ALTER TABLE departments DELETE COLUMN location;

What should you do to troubleshoot the issue?

**A1: Drop the constraints first.**

**A2: Replace the DELETE keyword**

E: You should replace the DELETE keyword.

* ALTER TABLE statement can be used to add, modify, or remove table columns.
* To remove a column, you should issue this command:
* ALTER TABLE departments DROP COLUMN location;
* You do not need to drop the constraints; the location column does not have any constraints that might cause an error if the column is dropped.
* You do not need to delete all the values in a column prior to deleting the column itself.
* You cannot include a WHERE clause in an ALTER TABLE statement; the WHERE clause is used in queries to filter records based on conditions.

Q104: CREATE INDEX emp\_title ON employees@job\_title;

What two actions should you take to troubleshoot this error?

**A1: Put the column name in parentheses**

**A2: Remove the @ symbol & put the column name in parentheses**

E: You should remove the @ symbol and put the column name in parentheses.

* An index is used to speed data lookups; when you create an index, you must specify an index name, and the table, and/or columns that will be used to build the index.
* The correct statement in this scenario is:
* CREATE INDEX idx\_emp\_title ON employees(job\_title);
* The @ symbol is used to identify variables or parameters in SQL, while the % symbol can be used with the LIKE operator as a wildcard.
* The AS keyword is used when defining column or table aliases, it is not used in the CREATE INDEX statement.
* Constraints are associated with tables, not indexes; you can define constraints when creating or altering a table.

Q105: CREATE VIEW view\_emp AS

SELECT last\_name, dept\_id, manager\_id FROM employees;

Later, when querying the view, you receive an error:

Could not use view or function ‘view\_emp’ because of binding errors.

What should you do to resolve this issue?

**A1: Create the view using WITH CHECK OPTION**

**A2: Recreate and populate the employees table**

E: In its simplest form, a view is a named SELECT statement.

* If the underlying table(s) the view queries are dropped, you will receive an error when querying the view.
* You should not use the CREATE OR ALTER VIEW statement; if the view did not exist, you would get an error indicating the view was an invalid object.
* You should not create the view using WITH CHECK OPTION; a check option ensures any data manipulation through the view does not put underlying records outside the view’s reach.
* You should not add a WHERE clause to the view definition; this will not address the underlying issue, which is a missing table; a WHERE clause is used to filter query results.

Q106: You need to return total salaries for each job; however, the following statement does not complete successfully:

SELECT job\_title, SUM(sal) FROM employees;

What should you do to troubleshoot the error?

**A1: Add a WHERE clause.**

**A2: Add a GROUP BY clause.**

E: You should add a GROUP BY clause.

* The GROUP BY clause is used to apply aggregate functions, such as COUNT( ) or SUM( ), on groups of records.
* As a rule, any column you include in your SELECT clause that is not part of a group function, must also be listed in your GROUP BY clause.
* The correct statement is:
* SELECT job\_title, SUM(sal) FROM employees GROUP BY job\_title;
* The HAVING clause is used to filter records after a group function has been applied; this clause is not required when using group functions.
* The WHERE clause is used to filter rows before the group function is applied; this means that only the records that meet the condition in the WHERE clause will be processed by the group function; this clause is not required when using group functions.

Q107: You are trying to return a list of all employees who have the word manager in their title by using wildcards. You run the following query:

SELECT empid, first\_name, last\_name, job\_title FROM employees WHERE job\_title = ‘%MANAGER%’;

However, your query fails to return results.

What should you do to troubleshoot this issue?

**A: Replace = with LIKE**

E: You should replace = with LIKE.

* You can use the LIKE operator in the WHERE clause of an SQL statement.
* This allows you to filter records based on a partial match, using wildcards.
* When used with the LIKE operator, the percent sign (%) represents zero to many characters, and the underscore (\_) represents a single character.
* The correct query is:
* SELECT empid, first\_name, last\_name, job\_title
* FROM employees WHERE job\_title LIKE ‘%MANAGER%’;
* The IN operator is used to match a list of values; you cannot use wildcards with the IN operator.
* You can use the BETWEEN operator in the WHERE clause to filter records based on a range of values; this operator works on dates, numbers, or text, and the lower & upper bounds are included in the results; BETWEEN does not support wildcards.
* The \* symbol can be used to perform multiplication, it can also be used in a SELECT statement to indicate all rows: SELECT \* FROM employees;

Q108: The following statement returns an error:

SELECT empid, first\_name, last\_name, dept AS department\_id, sal

FROM employees WHERE sal BETWEEN 5000 AND 9000

AND employees.department\_id IN (40, 50);

What should you do to troubleshoot this?

**A: Use the column name in the WHERE clause**

E: You should use the column name in the WHERE clause.

* In the SELECT clause, you have defined a column alias, department\_id, for the dept column.
* However, you cannot use the alias in the WHERE clause, you must use the original column name.
* The correct query is:
* SELECT empid, first\_name, last\_name, dept AS department\_id, sal
* FROM employees WHERE sal BETWEEN 5000 AND 9000
* AND dept IN (40, 50);
* You should not remove the second AND operator; if you want to include multiple conditions in your WHERE clause, you can use the AND or OR operators.
* You should not put the second condition in a separate WHERE clause; multiple WHERE clauses in the same base query are not allowed, you should use AND or OR instead.

Q109: The following statement fails when attempting to update the table:

UPDATE employees SET empid = 104 WHERE dept = 50;

What should you do to troubleshoot the issue?

**A1: Rename the dept column to department\_id.**

**A2: Ensure the condition is true for only one record.**

E: You should ensure the condition is true for only one record.

* The employees table has a PRIMARY KEY defined on the empid column.
* If you try to set more than one record to empid = 104, you will violate the PRIMARY KEY, which requires uniqueness.
* You do not need to ensure dept 50 exists in the departments table; however, if you insert records into the employees table, the FOREIGN KEY requires that the dept exists in the departments table.
* You do not need to rename the dept column to department\_id, the column names used in a FOREIGN KEY do not need to match.
* You are already using the correct data type for the updated column, the empid requires an int and you have supplied an int (empid = 104).

Q110: You attempt to run the following statement and receive an error that a constraint has been violated:

INSERT INTO employees VALUES

(140, ‘CHIN’, ‘YEN’, ‘LAB MGR’, 40, ‘2/16/19’, 4500),

(128, ‘MIKKA’, ‘PEARL’, ‘SR CPA’, 40, ‘12/22/15’, 8200);

What should you do?

**A1: Verify empid is not duplicate.**

**A2: Verify department 40 exists.**

E: You should verify department 40 exists.

* The INSERT statement includes a value for each column; none of the values violate the PRIMARY KEY or NOT NULL constraints.
* This means that the FOREIGN KEY constraint is causing the error; the most likely cause is that employees are being added to a department that does not exist; FOREIGN KEY constraints ensure referential integrity.
* You should not verify empid is not duplicate, empid is the first value in parentheses being passed by the INSERT statement; the values are unique and non-null, which means they do not violate the PRIMARY KEY constraint on the empid column.
* A column list is only required in an INSERT statement if you do not plan to supply values for all columns, or you provide them in an order different from the table specification.
* In this case, the column list, or column specification as it is known, is not required.
* The INSERT statement includes a value for all columns; if it did not, you would need to include a column specification and ensure you were not violating any constraints.

Q111: The following command returns an error:

INSERT INTO employees VALUES

(140, ‘CHIN’, ‘YEN’, ‘LAB MGR’, 40, 2/16/19, 4500);

What should you do to troubleshoot the issue?

**A: Ensure the correct data types are used.**

E: You should ensure the correct data types are used.

* The value for hire\_date in the INSERT statement will be interpreted as division (2 divided by 16 divided by 19) of integers.
* The resulting value is not compatible with the DATE data type specified for the hire\_date column, you should enclose the date value in single quotes.
* Values are provided for all the columns in the table, this ensures the NOT NULL constraints are not violated.
* As values are provided for all columns, a column specification is not required.
* The salary column uses the int data type, some databases include a MONEY data type for currencies.

Q112: The following statement returns an error:

DELETE employees WHERE dept NOT = 5;

What should you do to troubleshoot?

**A: Move the NOT keyword to right after WHERE.**

E: You should move the NOT keyword to right after WHERE.

* The NOT operator can be used to negate a condition, the correct syntax for this statement is:
* DELETE employees WHERE NOT dept = 5;
* You could add the FROM keyword, but it is not required:
* DELETE FROM employees WHERE NOT dept = 5;
* Putting the 5 in single quotes will cause the SQL engine to interpret the value as character data.
* The TRUNCATE TABLE statement is used to delete all records in a table; you cannot include a WHERE clause in a TRUNCATE TABLE statement.