**CREATE INDEX:**

A.

CREATE INDEX supplier\_id\_indx on PaymentDetails (supplier\_id);

-- Find all the payments for a certain supplier id

B.

CREATE INDEX paid\_employee\_idx on PaymentDetails (paid\_employee\_id);

-- Find all the payments for a certain employee

C.

CREATE INDEX paid\_date\_idx on PaymentDetails (paid\_date);

-- If you run queries to get payments for certain dates

**JUSTIFICATIONS:**

**A.**

**Justification:**

Within the tropical fish store cost database, the PaymentDetails table features a column named supplier\_id indicating the supplier associated with a payment. To optimize queries focused on retrieving payments for a specific supplier, it is warranted to create an index on the supplier\_id column.

Scenario:

Regularly, store managers or financial personnel need to access information regarding payments made to a particular supplier. In the absence of an index on the supplier\_id column, queries targeting payments associated with a specific supplier would necessitate a full table scan of the PaymentDetails table. As the payment records accumulate, this could lead to suboptimal query performance.

Benefits of Index:

By establishing an index on the supplier\_id column, the database engine gains the capability to efficiently locate and retrieve rows linked to a particular supplier. The index significantly improves the speed of queries that filter, sort, or join based on the supplier\_id. This efficiency boost is particularly valuable when managing a substantial volume of payment transactions.

With this index in place, queries aiming to find all payments for a certain supplier become more efficient. The database can swiftly access the pertinent rows based on the indexed supplier\_id, resulting in enhanced query performance.

**B.**

**Justification:**

In the tropical fish store cost database, the PaymentDetails table includes a column named paid\_employee\_id representing the employee responsible for a particular payment. To optimize queries that involve retrieving payments associated with a specific employee, creating an index on the paid\_employee\_id column is justified.

Scenario:

Store managers or financial staff frequently need to find all payments made to or by a specific employee. Without an index on the paid\_employee\_id column, queries searching for payments related to a particular employee would require a full table scan of the PaymentDetails table. As the number of payments increases, this could lead to slower query performance.

Benefits of Index:

By creating an index on the paid\_employee\_id column, the database engine can efficiently locate and retrieve rows associated with a specific employee. The index improves the speed of queries that filter, sort, or join based on the paid\_employee\_id. This enhancement in query performance becomes particularly valuable when dealing with a large volume of payment records.

With this index in place, queries like finding all payments for a certain employee become more efficient. The database can quickly access the relevant rows without the need for a full table scan, resulting in faster and more responsive query execution.

**C.**

**Justification:**

In the tropical fish store cost database, the PaymentDetails table contains a column named paid\_date that represents the date on which a payment was made. To optimize queries that involve retrieving payments for specific dates, creating an index on the paid\_date column is justified.

Scenario:

Store managers or financial staff often need to query and analyze payments made on specific dates. Without an index on the paid\_date column, date-based queries would necessitate a full table scan of the PaymentDetails table. As the number of payment records grows, this could result in slower query performance and decreased efficiency.

Benefits of Index:

By creating an index on the paid\_date column, the database engine can swiftly locate and retrieve rows corresponding to a specific date or range of dates. The index enhances the speed of queries involving date-based filtering, sorting, or grouping. This is particularly advantageous when dealing with a substantial amount of payment data.

With this index in place, queries such as retrieving payments for certain dates become more efficient. The database can directly access the relevant rows based on the indexed dates, leading to improved query performance.

**Justification for Primary Keys:**

Scenario:

In the tropical fish store cost database, each table is designed with a primary key to uniquely identify each record. The primary key serves as a fundamental component of database integrity and data organization.

Benefits of Primary Keys:

Uniqueness: The primary key ensures that each record in a table is uniquely identified. In the EmployeeData, SupplierDetails, PaymentDetails, InventoryDetails, MaintenanceDetails, and AquaticSpeciesList tables, the primary key (employee\_id, supplier\_id, payment\_id, item\_id, maintenance\_id, and species\_id respectively) guarantees the uniqueness of each employee, supplier, payment, inventory item, maintenance activity, and aquatic species.

Data Integrity: Primary keys help maintain data integrity by preventing duplicate or null values in the key columns. This ensures accurate and reliable data representation.

Referential Integrity: Primary keys serve as the basis for establishing relationships between tables. Foreign keys in other tables can reference the primary key, creating links between related records.

Efficient Data Retrieval: Searching for, updating, or deleting specific records is more efficient with primary keys. The database engine can quickly locate the exact record based on the primary key.

**Justification for Foreign Keys:**

Scenario:

In the tropical fish store cost database, foreign keys are used to establish relationships between tables, ensuring referential integrity and facilitating data consistency.

Benefits of Foreign Keys:

Referential Integrity: Foreign keys in the PaymentDetails, InventoryDetails, MaintenanceDetails, and AquaticSpeciesList tables establish relationships with the primary keys in other tables (EmployeeData, SupplierDetails, and PaymentDetails). This ensures that references to records in one table correspond to valid records in another, maintaining referential integrity.

Enforcement of Relationships: Foreign keys enforce relationships between tables, preventing actions that would violate the integrity of the database. For example, a payment record cannot reference a non-existent employee or supplier due to the foreign key constraint.

Cascade Operations: Foreign keys can be configured to perform cascade actions, such as cascading deletes or updates. This automates the maintenance of referential integrity by updating or removing related records when changes occur in the referenced table.

**Justification Notes from Josh**

My team wanted something that would make this project nice and simple the way we liked it and we came out with this “supplier\_id\_indx on PaymentDetails (supplier\_id);, paid\_employee\_idx on PaymentDetails (paid\_employee\_id); and paid\_date\_idx on PaymentDetails (paid\_date);” we just thought that since our project will be about employees why not have a index that involves a id tag.”)