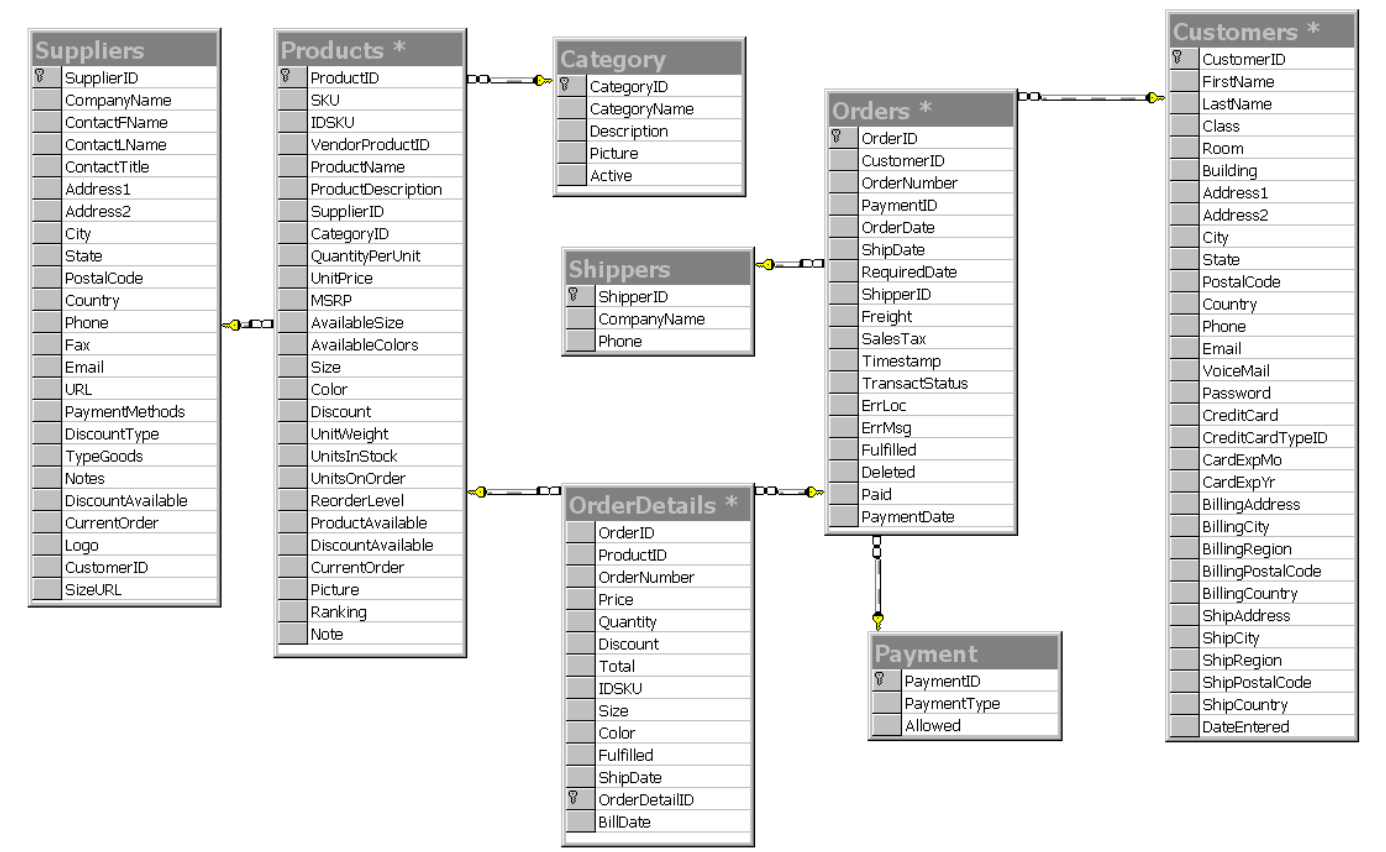
Database Assignment

1. Physical Entity Relationship diagram of database.

|  |  |  |  |
| --- | --- | --- | --- |
| Table | Primary Key | Related Table & It's Primary Key | Foreign Key to Relate Tables |
| Suppliers | SupplierID | Products - Product ID | SupplierID |
| Products | ProductID | Category - CategoryID | CategoryID |
| Orders | OrderID | OrderDetails - OrderDetailsID | OrderID |
| Customers | CustomerID | Orders - OrderID | CustomerID |
| Shippers | ShipperID | Orders - OrderID | ShipperID |
| Payment | PaymentID | Orders - OrderID | PaymentID |



1. Explain about searching performance. How will you handle replication in SQL for searching & Reporting?

* Creating the underlying database to support it. Building the application code and business logic into the database rather than in web page. This will make the application more portable and will allow the solution to scale upwards from a desktop database like MS access to enterprise-level databases like SQL server & and Oracle 8.

3. Explain what major factors are taken into consideration for performance.

- Customers table- holds customer information like address, shipping address and billing address, etc.

- Orders table – holds information on when the order was places including customer ID, date of order, order shipping date, etc.

- Products table – holds product information like product name, description, size, colour, unit price.

Order details table – holds information on each product ordered on one order including the product ordered, quantity, unit price any discounts, etc.

1. Mention about Indexing, Normalization and Denormalization.

**Normalization:**

1. **Entities:**
   * Products
   * Customers
   * Orders
   * Order Items (line items within orders)
   * Reviews
2. **Normalization Approach:**
   * Organize data to minimize redundancy and maintain data integrity.
   * Use separate tables for each entity and establish relationships using foreign keys.
   * For example, you have a “Customers” table, an “Orders” table, and an “Order Items” table, each linked by customer and order IDs.
3. **Advantages:**
   * Ensures data accuracy and consistency, reducing the risk of anomalies.
   * Simplifies data updates, as changes are made in one place.
   * Supports complex relationships, like multiple customers placing multiple orders.

**Denormalization:**

1. **Entities:**
   * Products
   * Orders
   * Customers
   * Reviews (with product and customer details denormalized)
2. **Denormalization Approach:**
   * Optimize for read-heavy workloads, especially for generating reports and product recommendations.
   * Combine data from multiple tables into a single table or a set of denormalized tables.
   * For example, you have a “Product Reviews” table that includes customer and product information, reducing the need for joins.
3. **Advantages:**
   * Improves query performance by reducing the number of joins.
   * Enhances reporting capabilities, making it easier to generate product reviews and recommendations.
   * Speeds up analytics tasks, such as calculating customer lifetime value.
4. How will you handle scaling, if required at any point of time.

- Use indexes- indexes help speed up queries by creating an index of frequently accessed data. This can significantly improve performance, particularly for large databases.

- Partition data- Partitioning involves dividing a large table into smaller, more manageable parts. This can improve performance by allowing the database to access data more quickly.

- Use buffer cache- can significantly improve performance. This is particularly useful for read-heavy workloads, and while it is always enabled in PostgreSQL, it can be tweaked for optimized performance.

- Consider data distribution- turning the database into smaller, more manageable partitions and then distributing (sharding) them across multiple cluster nodes.

1. Mention all the assumptions you are taking for solutions.

* Avoid Redundancy. Redundant information in a database schema can cause several problems. Primary Keys and Unique Identifiers. Every table must have a primary key. ...
* Null Values.
* Referential Integrity.
* Atomicity.
* Normalization.
* Data Typing.
* Indexing.