Practical Assignment: Database Implementation and Programming

An electronic copy of any document relevant to this assignment is available on Teams and the I:/ drive.

Learning Outcomes:

1. Implement troubleshooting techniques to solve complex database performance issues.
2. Design and develop relational databases to meet specified requirements that are subject to high-availability, high-reliability, security, and performance constraints.

Assessment

This assignment is contributes 30% towards your total mark for ID705 Databases 3. Mark allocations are shown on the task pages. The total for this assignment is 100 + 10 bonus marks.

Due Date

Monday 20th May 11:59 pm (Midnight) You must attend the Marking Demo
Tuesday 21st May

Submission

Push your scripts to the GitHub Classroom repo for Assignment
Create an assignment database on the supplied SQL server
named: YourUserName_ID705Assignment1

Implement Quote Compilation Database 30/110

Write a script that creates the Quote Compilation database design set out on the ERD. Run the script against your YourUserName_IN705Assignment1 database to create the objects Ensure your script includes these specific requirements:

- The tables, columns and other objects must be named as shown on the ERD or in this assignment. 10/30
- The column datatypes are not set; you must choose suitable column datatypes. Price data should handle NZD, TimeToFit data should be able to store minutes (decimal) *10 marks* 5/30
- All table columns are mandatory except: Contact.ContactEmail, Contact.ContactWWW, Contact.ContactMobilePhone, Contact.ContactFax, Quote.QuotedPrice. 2/30
- Primary keys are: Category(CategoryID), Contact(ContactID), Customer(CustomerID), Supplier(SupplierID), Component(ComponentID), Quote(QuoteID), QuoteComponent(QuoteID, ComponentID), AssemblySubcomponent(AssemblyID, SubcomponentID) 3/30
- Set suitable column defaults. 2/30
- Set check constraints to ensure that all price and timeToFit columns are non-negative. 3/30

Foreign key	**Parent key**	**Child key**
FK_Component_Category	Category.CategoryID	Component.CategoryID
FK_Supplier_Contact	Contact.ContactID	Supplier.SupplierID
FK_Customer_Contact	Contact.ContactID	Customer.CustomerID
FK_Component_Supplier	Supplier.SupplierID	Component.SupplierID
FK_Quote_Customer	Customer.CustomerID	Quote.CustomerID
FK_QuoteComponent_Quote	Quote.QuoteID	QuoteComponent.QuoteID

FK_Subcomponent_Component	Component.ComponentID	AssemblySubcomponent. SubcomponentID
FK_Assembly_Component	Component.ComponentID	AssemblySubcomponent. AssemblyID
FK_QuoteComponent_Component	Component.ComponentID	QuoteComponent. ComponentID

Populate the database using the PopulateQuoteCompilationWithComponents.sql 20/110

Overview:
Contact

The ContactID field should be an identity column.
Modify the script to use @@identity to capture the data generated for the primary key.

Supplier

Use the data captured above to populate the SupplierID (rather than hardcoding it)

Component combined 10/20

Try to use the data captured above to populate the SupplierID
This insert script uses the getCategoryID function to populate the CategoryID field

Create support routines

Create a function: getCategoryID that accepts CategoryName and returns the CategoryID

combined 10/20

Create a function: getAssemblySupplierID() that returns the contactID for 'BIT Manufacturing'

Create a procedure: createAssembly that accepts two parameters @componentName and componentDescription and inserts into the Component table.
Use 0 for ListPrice, TradePrice, TimeToFit.
Populate SupplierID and CategoryID by calling the functions getAssemblySupplier and getCategoryID- pass it 'Assembly'

Create a procedure addSubComponent that accepts assemblyName, subComponentName and quantity and inserts into the AssemblySubComponent (you will need a self join on Component)

Create the assemblies and subcomponents

Implement design elements on Quote Compilation database

50/110

In this section you will add new design features to the Quote Compilation design. All the work will be in your YourUserName_IN705Assignment1database. You may need to drop or alter constraints along the way.

- Design and implement a stored procedure ***createCustomer(Name, Phone, PostalAddress, Email, WWW, Fax, MobilePhone) Output: CustomerID*** that inserts a new customer to the database. The stored procedure will accept parameters that will provide the data for the new customer. As appropriate, make parameters optional depending on the properties of the underlying columns. The procedure returns the new CustomerID created by the procedure. — 10/50

- Design and implement two stored procedures ***createQuote(QuoteDescription, QuoteDate, QuoteCompiler, CustomerID, QuoteID OUTPUT)*** and ***addQuoteComponent(QuoteID, ComponentID, Quantity)*** that inserts a new quote data to the database. The stored procedures will accept parameters that will provide the data for the new quote. As appropriate, make parameters optional depending on the properties of the underlying columns. Make createQuote.QuoteDate optional, defaulting to the current datetime. Notice that QuoteComponent table captures the component prices and timeToFit at the moment the QuoteComponent is created. — 10/50

- Use the stored procedures *createCustomer, createQuote* and *addQuoteComponent* to populate the database with the customer-quote information (attached). Hand in the script you wrote to perform this task. — 5/50

- ** This is in place of the uC in the diagram ** Design and implement the triggers necessary to implement cascade update on constraints ***FK_Assembly_Component*** and ***FK_Subcomponent_Component***. — 10/50

- Design and implement a trigger ***trigSupplierDelete*** that reacts to a Supplier record being deleted. If the supplier has one or more related component(s) the deletion will be cancelled and the error message 'You cannot delete this supplier. XYZ has K related components.' printed in the messages window [substitute XYZ -> SupplierName, K -> number of related components]. — 10/50

- Design and implement a stored procedure *updateAssemblyPrices()* that will efficiently and completely update the trade price and list price of all assemblies. The trade price is the total of all subcomponent trade prices. The list price is the total of all subcomponent list prices. — 5/50

- Design and implement a stored procedure *testCyclicAssembly(assemblyID):isCyclic* that will test if an assembly directly or indirectly contains itself as a subcomponent. The procedure returns a bit (1 = true; assembly is cyclic, 0 = false; assembly is not cyclic) as the test result — Bonus 10

Sample quote information 1)

Customer:	Bimble & Hat 123 Digit Street, Dunedin Ph: 444 5555 Email: guy.little@bh.biz.nz
Quote:	Craypot frame
Components:	3 of 15mm square strap assembly 8 of 1250mm length BMS.5.15 ms bar 24 of BMS10 bolt 24 of NMS10 nut 200ml of anti-rust paint, blue 150 minutes of artisan labour 120 minutes of apprentice labour 45 minutes of designer labour

2)

Customer:	Bimble & Hat…
Quote:	Craypot stand
Components:	2 of 2565mm length BMS.15.40 ms bar 4 of BMS15 bolt 4 of NMS15 nut 100ml of anti-rust paint, blue 90 minutes of apprentice labour 15 minutes of designer labour

3)

Customer:	Hyperfont Modulator (International) Ltd. 3 Lambton Quay, Wellington Ph: (4) 213 4359 Email: sue@nz.hfm.com
Quote:	Phasing restitution fulcrum
Components:	3 of 15mm corner brace assembly 1 of 15mm small corner 320 minutes of artisan labour 105 minutes of designer labour 0.5 litre of anti-rust paint, red

Quote Compilation Database Design

My light engineering company designs and builds equipment for industry. We quote for jobs that are advertised for public tender as our main method of gaining work. We need an information system that will support our quote compiling effort.

All our quotes are comprised of components that fall into one of three categories: parts, labour and assemblies.

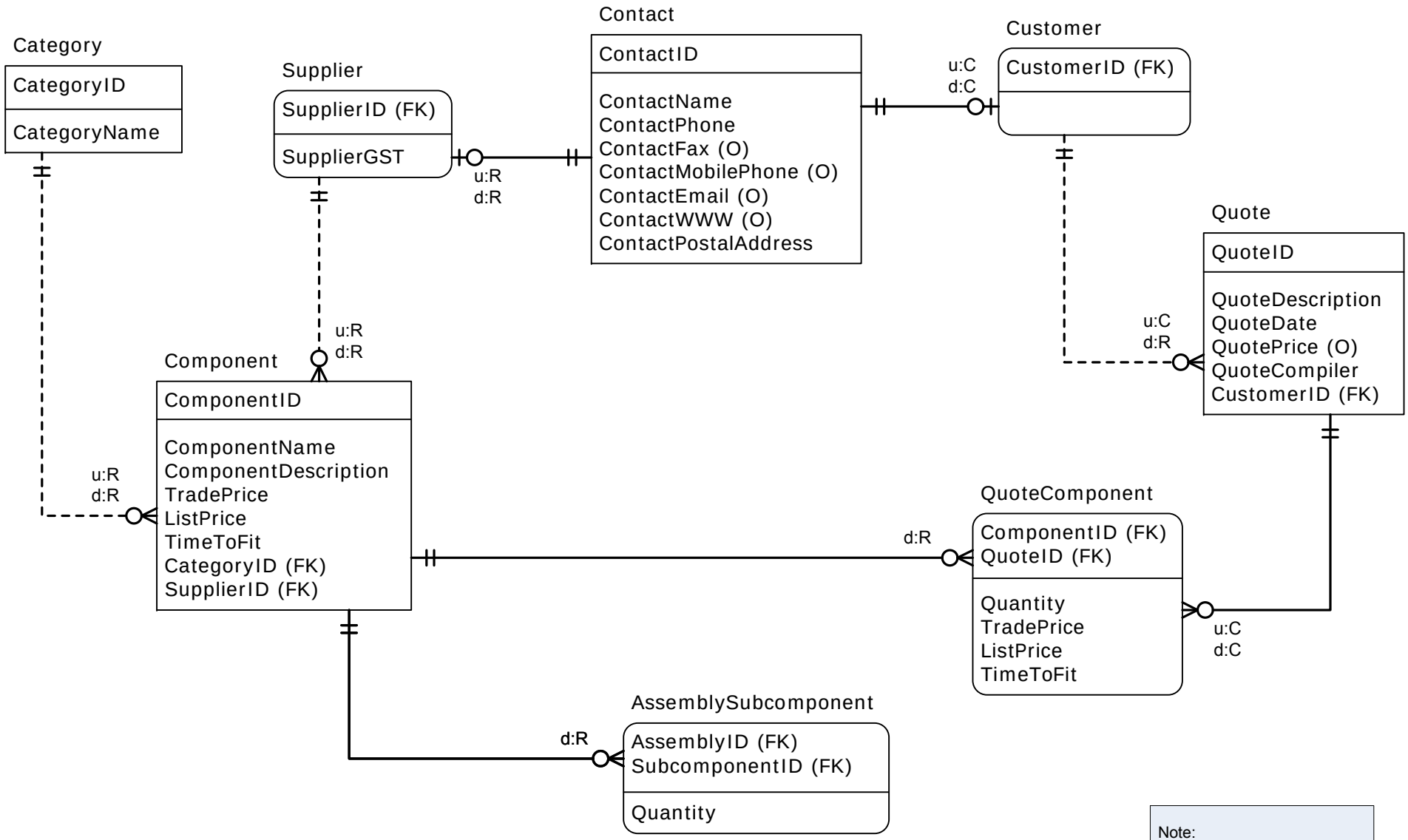
We buy parts from our suppliers. Most of our suppliers send us a parts list in an electronic format. The parts list has details such as parts code, description, list price and category – there is lots of other junk that we don't use. When we actually buy the parts from suppliers we usually have a lower 'trade' price – we usually have a standing discount with each supplier but sometimes we negotiate a special 'one–off' price. The parts categories are often in common with other suppliers' categories – we want to merge the similar categories into common categories across all suppliers to help us search for parts. On our IS we also want to be able to add parts produced by my company; some stuff is produced in–house.

Labour is another type of component – we are always the supplier. For each part we want to record an estimated 'time to fit' so we can judge how much labour to add.

Assemblies are special in–housed things; we build them as standard components that are always coming up in constructions. They are made up of combinations of parts, labour and, possibly, other assemblies. We want to 'construct' an assembly and add it to a library of assemblies in the IS so we can use it again.

When it comes to quoting we want to build up a list of parts, labour and assemblies that build the design at hand – then the IS will calculate the cost of the job. Importantly, the prices in a quote must NOT change when the underlying component prices change at a later date. We'll want to look back at past quotes and know the component prices when we constructed the quote – our accounting methods rely on this. We want to know the cost of a quote but be able to apply a price to the customer: the system will show us the profit we will make on the quote.

We often send the same quote to several customers: they may be consulting engineers contracting for a major job and we are sub–contractors. We need to send the same quote to several customers but ensure we change the address details on each quote. The system must record who received which quotes.


Category
CategoryID
CategoryName
Supplier
SupplierID (FK)
SupplierGST
u:R
d:R
Contact
ContactID
ContactName
ContactPhone
ContactFax (O)
ContactMobilePhone (O)
ContactEmail (O)
ContactWWW (O)
ContactPostalAddress
u:C
d:C
Customer
CustomerID (FK)
Quote
QuoteID
QuoteDescription
QuoteDate
QuotePrice (O)
QuoteCompiler
CustomerID (FK)
u:C
d:R
u:R
d:R
Component
ComponentID
ComponentName
ComponentDescription
TradePrice
ListPrice
TimeToFit
CategoryID (FK)
SupplierID (FK)
u:R
d:R
d:R
QuoteComponent
ComponentID (FK)
QuoteID (FK)
Quantity
TradePrice
ListPrice
TimeToFit
u:C
d:C
d:R
AssemblySubcomponent
AssemblyID (FK)
SubcomponentID (FK)
Quantity
Note:
u:C = on update cascade
d:C = on delete cascade
d:R = on delete no action