CS990 Database Fundamentals
Andrea Gibb

Logical Design

Part (PartID, Description)
BulkPart (PartID*, Location)
ToOrder (PartID*, LeadTime)
Bought (PartID*, Cost)

UsedIn (PartID*, UsedFor*, Quantity)

CustomerOrder (OrderID, OrderDate)
OrderLine (OrderID*, PartID*, Quantity)

Supplier (SupplierID, Name, Address)
SupplierPart (SPID, Price, SupplierID*)

BankAccount (SupplierID*, SortCode, AccountNumber)

Creating and loading the database

CREATE TABLE Part (PartID NUMBER NOT NULL, Description NVARCHAR2(255), CONSTRAINT pk_PartID PRIMARY KEY (PartID));

CREATE TABLE BulkPart (PartID NUMBER NOT NULL, Location NVARCHAR2(255), CONSTRAINT bp_pk_PartID PRIMARY KEY (PartID), CONSTRAINT bp_fk_PartID FOREIGN KEY (PartID) REFERENCES Part (PartID));

CREATE TABLE ToOrder (PartID NUMBER NOT NULL, LeadTime NVARCHAR2(255), CONSTRAINT to_pk_PartID
PRIMARY KEY (PartID), CONSTRAINT to_fk_PartID FOREIGN KEY (PartID) REFERENCES Part (PartID));

CREATE TABLE Bought (PartID NUMBER NOT NULL, Cost NVARCHAR2(255), CONSTRAINT b_pk_PartID PRIMARY KEY (PartID), CONSTRAINT b_fk_PartID FOREIGN KEY (PartID) REFERENCES Part (PartID));

CREATE TABLE UsedIn (PartID NUMBER NOT NULL, UsedFor NUMBER NOT NULL, Quantity NVARCHAR2(255), CONSTRAINT ck_PartID_UsedFor
PRIMARY KEY (PartID, UsedFor), CONSTRAINT ui_fk_PartID FOREIGN KEY (PartID) REFERENCES Part (PartID), CONSTRAINT ui_fk_UsedFor FOREIGN KEY (UsedFor) REFERENCES Part (PartID));

CREATE TABLE CustomerOrder (OrderID NUMBER NOT NULL, OrderDate NVARCHAR2(255), CONSTRAINT o_pk_OrderID PRIMARY KEY (OrderID));

```
CREATE TABLE OrderLine (OrderID NUMBER NOT NULL, PartID NUMBER NOT NULL, Quantity
NVARCHAR2(255), CONSTRAINT ck_OrderID_PartID
PRIMARY KEY (OrderID, PartID), CONSTRAINT ol_fk_OrderID FOREIGN KEY (OrderID) REFERENCES
CustomerOrder (OrderID), CONSTRAINT ol_fk_PartID FOREIGN KEY (PartID) REFERENCES Part (PartID));


CREATE TABLE Supplier (SupplierID NUMBER NOT NULL, Name NVARCHAR2(255), Address
NVARCHAR2(255), CONSTRAINT s_pk_SupplierID PRIMARY KEY (SupplierID));


CREATE TABLE SupplierPart (SPID NUMBER NOT NULL, Price NVARCHAR2(255), SupplierID NUMBER NOT
NULL, CONSTRAINT sp_pk_SPID PRIMARY KEY (SPID), CONSTRAINT sp_fk_SupplierID FOREIGN KEY
(SupplierID) REFERENCES Supplier (SupplierID));


CREATE TABLE BankAccount (SupplierID NUMBER NOT NULL, SortCode NVARCHAR2(255), AccountNumber
NVARCHAR2(255), CONSTRAINT bk_pk_SupplierID PRIMARY KEY (SupplierID), CONSTRAINT
bk_fk_supplierID FOREIGN KEY (SupplierID) REFERENCES Supplier (SupplierID));
```

INSERT DATA STATEMENTS

```
INSERT ALL
INTO Part (PartID, Description) VALUES (495049, 'Windscreen wipers, used to clear the window in a car')
INTO Part (PartID, Description) VALUES (495050, 'Gear Lever, used to change gears in car')
INTO Part (PartID, Description) VALUES (495051, 'Steering Wheel, used to steer the wheel in the direction of the
car')
INTO Part(PartID, Description) VALUES (495001, 'Gearbox Assembly')
SELECT * FROM dual;

INSERT ALL
INTO BulkPart (PartID, Location) VALUES (495051, 'East London')
SELECT * FROM dual;

INSERT ALL
INTO ToOrder (PartID, LeadTime) VALUES (495050, '24 hrs')
SELECT * FROM dual;

INSERT ALL
INTO Bought (PartID, Cost) VALUES (495049, '£9.99')
SELECT * FROM dual;

INSERT INTO UsedIn (PartID, UsedFor, Quantity) VALUES (495050, 495001, 1)

INSERT ALL
INTO CustomerOrder (OrderID, OrderDate) VALUES (1, '1.3.19')
INTO CustomerOrder (OrderID, OrderDate) VALUES (2, '2.3.10')
SELECT * FROM dual;

INSERT ALL
INTO OrderLine (OrderID, PartID, Quantity) VALUES (1, 495050, 5)
INTO OrderLine (OrderID, PartID, Quantity) VALUES (1, 495001, 1)
INTO OrderLine (OrderID, PartID, Quantity) VALUES (1, 495049, 2)
```

```
INTO OrderLine (OrderID, PartID, Quantity) VALUES (2, 495001, 1)
SELECT * FROM dual;

INSERT INTO Supplier (SupplierID, Name, Address) VALUES (1200, 'John Smith', '123 Glasgow Road');

INSERT INTO SupplierPart (SPID, Price, SupplierID)
VALUES (265432, '£24.99', 1200);

INSERT INTO BankAccount (SupplierID, SortCode, AccountNumber)
VALUES (1200, 808080, 00738384);
```

Querying the database

(i) carry out a join between two tables and use the group by clause.
(Explanation) See how many of each part have been ordered in total by grouping by PartID

```
SELECT PartID, SUM(Quantity)
FROM OrderLine
GROUP BY PartID;
```

(Output)

```
   PARTID SUM(QUANTITY)
---------- -------------
   495050             5
   495001             2
   495049             2
```

(ii) execute a sub-query.
(Explanation) Selects parts that are used in other parts by selecting all PartIDs from UsedIn

```
SELECT * FROM Part WHERE PartID IN (SELECT PartID FROM UsedIn);
```

(Output)

```
   PARTID
----------
DESCRIPTION
--------------------------------------------------------------------------------
   495050
Gear Lever, used to change gears in car
```

(iii) execute a correlated-query.
(Explanation) Selects all orderlines with an above average quantity

```
SELECT * FROM OrderLine X WHERE Quantity > (SELECT AVG(Quantity) FROM OrderLine WHERE X.OrderID = OrderID);
```

(Output)

```
  ORDERID     PARTID
---------- ----------
QUANTITY
--------------------------------------------------------------------------------
```

```
         1    495050
5
```

(iv) carry out a self join that uses primary key/foreign key attributes
(Explanation) Selects all orders that have parts in common by self joining the Orderline table on partID and excluding OrderLines with the same OrderID

```
SELECT DISTINCT t1.OrderID, t1.PartID FROM OrderLine t1, OrderLine t2 WHERE t1.PartID = t2.PartID AND t1.OrderID <>
t2.OrderID;
```

(Output)

```
  ORDERID    PARTID
---------- ----------
         1    495001
         2    495001
```

Database critique

For this database I created separate relations for super-type and sub-type entities, using identities to connect the super and sub tables, the disadvantage were many tables, however this strategy is best suited to the data because it eliminates all null values.

The strengths
The database is normalised to 3rd normal form, this reduces the data redundancy, once redundancy is removed, it is easy to change the data since data is present in only one place.
The advantages of using constraints in this database means all data must adhere to the same integrity constraints, having rules associated with the database prevents the entry of invalid information into tables. Referential integrity prevents the entry of duplicate data, it also prevents one table from pointing to a nonexistent field in another table. Having referential integrity in this database means a record can not be deleted if the value is referred to by a foreign key in another table. It also solves the problem of dangling references by allowing cascading deletes.

The weaknesses
Many tables for subtables.

Assumptions
Some columns were redundant resulting in a slight simplification in the logical design.
The entity “Used In” originally had UID as the primary key, however PartID and Used For became a composite key as together they are guaranteed to be unique
The entity bank account had a BAID primary key, this wouldn't work because each supplier only has one bank account and each account only has one supplier (1:1) so the primary key was changed to SupplierID