1. Memory Calculations

(a) \[ (3 \text{ minutes}) \left( 60 \frac{\text{seconds}}{\text{minute}} \right) \left( 25 \frac{\text{frames}}{\text{second}} \right) \left( 400 \times 600 \frac{\text{pixels}}{\text{frame}} \right) \left( 3 \frac{\text{bytes}}{\text{pixel}} \right) = 3.24 \times 10^9 \text{ bytes}. \]

\[ (3.24 \times 10^9 \text{ bytes}) \div \left( 1024^3 \frac{\text{bytes}}{\text{GB}} \right) \approx 3.02 \text{ GB}. \]

(b) \[ (3.24 \times 10^9 \text{ bytes}) \div \left( 20 \frac{\text{original bytes}}{\text{compressed bytes}} \right) = 1.62 \times 10^8 \text{ bytes} \]

\[ (1.62 \times 10^8 \text{ bytes}) \div \left( 1024^2 \frac{\text{bytes}}{\text{MB}} \right) \approx 154 \text{ MB} \]

(c) \[ (1.62 \times 10^8 \text{ bytes}) \times \left( 8 \frac{\text{bits}}{\text{byte}} \right) \div \left( 768,000 \frac{\text{bits}}{\text{second}} \right) \div \left( 60 \frac{\text{seconds}}{\text{minute}} \right) \approx 28.1 \text{ minutes} \]

Note that communication line speeds are normally quoted in decimal bits per second; therefore, the “K” in “768 Kb/s” means 1000, not 1024.

2a. Database Design Practice Material

2a.i: Rent-a-Wreck

- **RESERVATION**
  - M

- **CUSTOMER**
  - 1

- **RENTAL**
  - M

- **VEHICLETYPE**
  - 1

- **VEHICLE**
  - M

**VEHICLETYPE** (TypeId, Description, DailyRate, WeeklyRate)

**VEHICLE** (LicensePlate, Manufacturer, Model, Year, DateAcquired, MileageAcquired, Notes,TypeId)

TypeId foreign key to VEHICLETYPE
CUSTOMER(CustomerID, DriversLicense, LicenseState, FirstName, MiddleName, LastName, BirthDate, Address, City, State, Zip, Phone, AlternatePhone)

RESERVATION(ReservationID, TypeID, CustomerID, PickupTime, ReturnTime)
  TypeID foreign key to VEHICLETYPE
  CustomerID foreign key to CUSTOMER

RENTAL(RentalID, LicensePlate, CustomerID, RentTime, PromisedReturn, RentalMileage, RentalFuelLevel, ActualReturnTime, ReturnFuelLevel, ReturnMileage, Notes)
  LicensePlate foreign key to VEHICLE
  CustomerID foreign key to CUSTOMER

One could also include a foreign key indicating that a given rental was a result of a given reservation. It could either be in RENTAL (blank for walk-ins), or RESERVATION (blank for “no-shows”). However, the problem text did not explicitly require such information be stored.

2a.ii. Plumbing Parts

PART(PartNum, Description, Inventory)

CONTAINS(PartNum, ContainedPartNum, Quantity)
  PartNum foreign key to PART
  ContainedPartNum foreign key to PART

SUPPLIER(SupplierID, Name, Phone, Address, City, State, Zip)

SUPPLIES(SupplierID, PartNum, LotSize, LotPrice)
  SupplierID foreign key to SUPPLIER
  PartNum foreign key to PART

RETAILPART(PartNum, WholesalePrice, ShippingWeight, ResalePrice)
  PartNum foreign key to PART

REPLACEMENTPART(PartNum, ReplacementPrice, ShippingWeight)
  PartNum foreign key to PART
Note that RETAILPART and REPLACEMENTPART are both subtypes of PART. Also, the text suggests that each supplier only supplies a given part in a single lot size. If you wanted to allow a supplier to offer multiple lot sizes, each with its own price, you could change the primary key of SUPPLIES to \((\text{SupplierID, PartNum, LotSize})\).

2a.iii. Custom Cabinetry Shop

![Database Diagram]

\[
\text{KINDOFCABINET(}\text{CabinetCode, BasePrice, Description, ShipWeight, CartonType, LaborHours, NumberKnobsOrHandles)}
\]

\[
\text{WOOD(}\text{WoodCode, Name, PercentMarkup)}
\]

\[
\text{FINISH(}\text{FinishCode, Name, PercentMarkup)}
\]

\[
\text{DOORSTYLE(}\text{DoorStyleCode, Name, Description, PercentMarkup)}
\]

\[
\text{HARDWARE(}\text{HardwareCode, Description, UnitPrice)}
\]

\[
\text{SALESREP(}\text{SalesRepID, FirstName, LastName, PhoneExtension)}
\]

\[
\text{CUSTOMER(}\text{CustomerID, CompanyName, ContactFirstName, ContactLastName, Phone, Address, City, State, Zip, MainRepID, BackupRepID)}
\]

\[
\text{MainRepID foreign key to SALESREP}
\]

\[
\text{BackupRepID foreign key to SALESREP}
\]

\[
\text{ORDER(}\text{OrderID, DatePlaced, DeliveryRequestDate, DateShipped, CustomerID, WoodCode, FinishCode, DoorStyleCode, HardwareCode)}
\]

\[
\text{CustomerID foreign key to CUSTOMER}
\]

\[
\text{WoodCode foreign key to WOOD}
\]

\[
\text{FinishCode foreign key to FINISH}
\]

\[
\text{DoorStyleCode foreign key to DOORSTYLE}
\]

\[
\text{HardwareCode foreign key to HARDWARE}
\]

\[
\text{ORDERDETAILS(}\text{OrderID, CabinetCode, NumberInOrder)}
\]

\[
\text{OrderId foreign key to ORDER}
\]

\[
\text{CabinetCode foreign key to KINDOFCABINET}
\]
In this problem, students often misplace the relationships connecting WOOD, FINISH, DOOR-STYLE, and HARDWARE to the rest of the design. Remember that all the cabinets in an order must have the same combination of wood, finish, door style, and hardware, and that the customer may choose any possible combination. So, OrderID directly determines WoodCode, FinishCode, DoorStyleCode, and HardwareCode, and ORDER is thus in 1-to-many relationships with WOOD, FINISH, DOORSTYLE, and HARDWARE. If you were to place the foreign keys WoodCode, FinishCode, DoorStyleCode, and HardwareCode in ORDERDETAILS instead of ORDERS, that would mean that each order could contain a mix of different woods, finishes, and so forth. If you were to place these foreign keys in KINDOF CABINET, then each kind of cabinet could only be produced in one combination of wood, finish, door style, and hardware.

2a.iv. Law Firm

CLIENT(ClientID, Name, Phone, Address, City, State, Zip)

PROBONO(ClientID, ReferralDate, CaseWorkerName, CaseWorkerPhone, LegalAidClientID)
  ClientID foreign key to CLIENT

ATTORNEY(EmployeeNumber, Name, HourlyRate, SupervisorEmployeeNumber)
  SupervisorEmployeeNumber foreign key to ATTORNEY

CASE(CaseNumber, ClientContactName, DateOpened, DateClosed, Description,
  ClientID, ResponsibleAttorney)
  ClientID foreign key to CLIENT
  ResponsibleAttorney foreign key to ATTORNEY

BILLEDHOURS(CaseNumber, EmployeeNumber, DateBilled, Hours)
  CaseNumber foreign key to CASE
  EmployeeNumber foreign key to ATTORNEY

BILLEDHOURS could also be depicted as an entity in its own right, in which case it would be in many-to-one relationships with CASE and ATTORNEY. You could also choose a synthetic key for BILLEDHOURS, instead of the composite key shown. The usual composite key (CaseNumber, EmployeeNumber) would not work, because the same employee may bill hours to a case on more than one date (as happens in the example data).
2b. Database Normalization Practice Material

Note that for all these problems, a practical version of the database would probably have more fields, and some of these fields (like names) would be broken down into more parts.

2b.i. Courses, Students, and Grades

MAJOR(MajorID, MajorName)

STUDENT(StudID, StudName, MajorID)
  MajorID foreign key to MAJOR

INSTRUCTOR(InstructorID, InstructorName, InstructorOffice)

COURSE(CourseID, CourseTitle, InstructorID)
  InstructorID foreign key to INSTRUCTOR

GRADE(StudID, CourseID, Grade)
  StudID foreign key to STUDENT
  CourseID foreign key to COURSE

2b.ii. Interfunctional Teams

DEPARTMENT(DepartmentCode, Name)

EMPLOYEE(EmployeeID, Name, DepartmentCode, Email, Phone)
  DepartmentCode foreign key to DEPARTMENT

TEAM(TeamID, Name, DateFormed)

MEMBER(EmployeeID, TeamID)
  EmployeeID foreign key to EMPLOYEE
  TeamID foreign key to TEAM
2b.iii. Print Advertising Placements

```
AD (AdID, Description)
PLACED (AdID, MagazineID, IssueDate)
MAGAZINE (MagazineID, MagazineName, PublisherID)
   PublisherID foreign key to PUBLISHER
PUBLISHER (PublisherID, PublisherName)
```

Note that since a given ad can be placed in the same magazine more than once, we should either include IssueDate in the primary key for PLACED, or use a synthetic key. The composite key is probably better, since it enforces the constraint that a given ad may appear at most once per issue. Note that there is no need for an ISSUE entity in this example, since there are no attributes to put in it. If there were any attributes in the original table determined by (MagazineID, IssueDate), then we would need an ISSUE entity, with primary key (MagazineID, IssueDate). An example of such an attribute would be the number of pages in the issue; however, there were no such attributes.

3. Query Exercises

(a)  
```
Field: Name Name EstimatedCost ActualCharged Description
Table: VENDOR PROJECT CONTRACT CONTRACT VENDOR-TYPE
Total: Sort: Show: Criteria:  > 1.2 *[EstimatedCost] “Plumber”
```

A common error here is to use 0.2 instead of 1.2. But that would produce a list of all plumbers who charged at least 20% of their estimate, not 20% over their estimate.

(b)  
```
Field: Name ProjectID EstimatedCost ActualCharged ActualDoneDate
Table: PROJECT PROJECT CONTRACT CONTRACT CONTRACT
Total: Group By Group By Sum Sum Group By*
Sort: Descending
Show: Criteria: <= #4/30/2006#
```
The “Group by” for ProjectID is necessary to keep different projects with identical names from being combined. Note that you could also use “Where” instead of “Group By” in the last column.

(c)

<table>
<thead>
<tr>
<th>Field:</th>
<th>Name</th>
<th>Description</th>
<th>ContractID</th>
<th>ActualCharged</th>
<th>VendorID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table:</td>
<td>VENDOR</td>
<td>VENDOR-TYPE</td>
<td>CONTRACT</td>
<td>CONTRACT</td>
<td>VENDOR</td>
</tr>
<tr>
<td>Total:</td>
<td>Group By</td>
<td>Group By</td>
<td>Count</td>
<td>Avg</td>
<td>Group By</td>
</tr>
<tr>
<td>Sort:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Criteria:</td>
<td>“Plumber”</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Or:</td>
<td>“HVAC”</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The “Group by” for VendorID is necessary to keep different vendors with the same names from being combined. The single criterion “Plumber” or “HVAC” would also work.

(d)

<table>
<thead>
<tr>
<th>Field:</th>
<th>Name</th>
<th>State</th>
<th>Description</th>
<th>JobSiteState</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table:</td>
<td>VENDOR</td>
<td>VENDOR</td>
<td>PROJECT</td>
<td>PROJECT</td>
<td>VENDOR-TYPE</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sort:</td>
<td></td>
<td></td>
<td></td>
<td>Ascending</td>
<td></td>
</tr>
<tr>
<td>Show:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Criteria:</td>
<td>&lt;&gt; [State] “Electrician”</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You could also write “[VENDOR].[State]” instead of “[State]”, but since VENDOR is the only table with an attribute named State, that isn’t required (in PROJECT, the state happens to be called JobSiteState).

(e)

<table>
<thead>
<tr>
<th>Field:</th>
<th>Name</th>
<th>ContractID</th>
<th>JobSiteState</th>
<th>ActualDoneDate</th>
<th>ActualCharged</th>
<th>VendorID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table:</td>
<td>VENDOR</td>
<td>CONTRACT</td>
<td>PROJECT</td>
<td>CONTRACT</td>
<td>CONTRACT</td>
<td>VENDOR</td>
</tr>
<tr>
<td>Total:</td>
<td>Group By</td>
<td>Count</td>
<td>Group By*</td>
<td>Where</td>
<td>Where</td>
<td>Group By</td>
</tr>
<tr>
<td>Sort:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Criteria:</td>
<td>“NJ” &gt; [ScheduleDoneDate]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Or:</td>
<td>“NJ” &gt; [EstimatedCost]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some minor variations are possible in how one detects late and over-budget contracts, but I think the setup above is the most natural. The “Group by” for VendorID is necessary to keep different vendors with the same names from being combined. Note that you must use “Where” in the criteria columns for the contract dates and costs, since these criteria must be applied before grouping. For JobSiteState, either “Group by” or “Where” should work. Note that the criterion “NJ” in the JobSiteState needs to be repeated, so that it is combined with both “late” (the first criterion line) and “over budget” (the second criterion line). If “NJ” only appeared on the first criterion line, then all over-budget contracts would be included, whether or not they were for projects in NJ.