Chapter 1 – Introduction to Databases – Answers to Review Questions

1.1 *List four examples of database systems other than those listed in Section 1.1.*

Some examples could be:

- A system that maintains component part details for a car manufacturer
- An advertising company keeping details of all clients and adverts placed with them
- A training company keeping course information and participants' details
- An organization maintaining all sales order information

1.2 *Discuss each of the following terms:*

Data	For end users, this constitutes all the different values connected with the various objects/entities that are of concern to them. See also Section 1.3.4.
Database	See Section 1.3.1
Database Management System	See Section 1.3.2
Database Application Program	See Section 1.3.3
Data Independence	This is essentially the separation of underlying file structures from the programs that operate on them, also called program-data independence. See also Sections 1.2.2 and 1.3.1.
Security	The protection of the database from unauthorized users, which may involve passwords and access restrictions. See also Section 1.6.
Integrity	The maintenance of the validity and consistency of the database by use of particular constraints that are applied to the data. See also Section 1.6.
Views	These present only a subset of the database that is of particular interest to a user. Views can be customized, for example, field names may change, and they also provide a level of security preventing users from seeing certain data. See also Section 1.3.3.

1.3 *Describe the approach taken to the handling of data in the early file-based systems. Discuss the disadvantages of this approach.*

Focus was on applications for which programs would be written, and all the data required would be stored in a file or files owned by those programs. See also Section 1.2.

Clearly, each program was responsible for only its own data, which could be repeated in other program's data files. Different programs could be written in different languages, and would not be able to access another program's files. This would be true even for those programs written in the same language, because a program needs to know the file structure before it can access it. See also Section 1.2.2.

1.4 *Describe the main characteristics of the database approach and contrast it with the file-based approach.*

Focus is now on the data first, and then the applications. The structure of the data is now kept separate from the programs that operate on the data. This is held in the system catalog or data dictionary. Programs can now share data, which is no longer fragmented. There is also a reduction in redundancy, and achievement of program-data independence. See also Section 1.3.

1.5 *Describe the five components of the DBMS environment and discuss how they relate to each other.*

See Section 1.3.3.

1.6 *Discuss the roles of the following personnel in the database environment:*

Data Administrator	See Section 1.4.1
Database Administrator	See Section 1.4.1
Logical Database Designer	See Section 1.4.2
Physical Database Designer	See Section 1.4.2
Application Developer	See Section 1.4.3
End-Users	See Section 1.4.4

1.7 *Discuss the three generations of DBMSs.*

The CODASYL and hierarchical approaches represented the first generation of DBMSs. They were based on the concept that smaller components come together as parts of larger components, and so on, until the final product is assembled. This structure, which conforms to an upside down tree, is also known as a hierarchical structure.

Relational DBMSs are referred to as second-generation DBMSs. In 1970, E. F. Codd of the IBM Research Laboratory produced his highly influential paper on the relational data model ("A relational model of data for large shared data banks," Codd, 1970). This paper was very timely and addressed the disadvantages of the former approaches. Many experimental relational DBMSs were implemented thereafter.

In response to the increasing complexity of database applications, two "new" systems have emerged: the object-oriented DBMS (OODBMS) and the object-relational DBMS (ORDBMS). However, unlike previous models, the actual composition of these models is not clear. This evolution represents third generation DBMSs.

1.8 *Discuss the advantages and disadvantages of database management systems.*

See Section 1.6

Chapter 2 –Database Environment – Answers to Review Questions

2.1 *Discuss the concept of data independence and explain its importance in a database environment.*

See Section 2.1.5

2.2 *To address the issue of data independence, the ANSI-SPARC three-level architecture was proposed. Compare and contrast the three levels of this model.*

See Section 2.1

2.3 *What is a data model? Discuss the main types of data models.*

An integrated collection of concepts for describing and manipulating data, relationships between data, and constraints on the data in an organization. See Section 2.3.

Object-based data models such as the Entity-Relationship model (see Section 2.3.1). Record-based data models such as the relational data model, network data model, and hierarchical data model (see Section 2.3.2). Physical data models describe how data is stored in the computer (see Section 2.3.3).

2.4 *Discuss the function and importance of conceptual modeling.*

See Section 2.3.4.

2.5 *Describe the types of facility you would expect to be provided in a multi-user DBMS.*

Data Storage, Retrieval and Update
A User-Accessible Catalog
Transaction Support
Concurrency Control Services
Recovery Services
Authorization Services
Support for Data Communication
Integrity Services
Services to Promote Data Independence
Utility Services

See also Section 2.4

2.6 *Of the facilities described in your answer to Question 2.5, which ones do you think would **not** be needed in a standalone PC DBMS? Provide justification for your answer.*

Concurrency Control Services - only single user.
Authorization Services - only single user, but may be needed if different individuals are to use the DBMS at different times.
Utility Services - limited in scope.
Support for Data Communication - only standalone system.

2.7 *Discuss the function and importance of the system catalog.*

See Section 2.4, Service (2) – User-accessible catalog.

2.8 *Discuss the differences between DDL and DML. What operations would you typically expect to be available in each language?*

DDL - A language that allows the DBA or user to describe and name the entities, attributes, and relationships required for the application, together with any associated integrity and security constraints.

DML - A language that provides a set of operations to support the basic data manipulation operations on the data held in the database.

See Section 2.2.2.

2.9 *Discuss the differences between procedural DMLs and nonprocedural DMLs.*

Procedural DML - A language that allows the user to tell the system what data is needed and exactly how to retrieve the data.

Nonprocedural DML - A language that allows the user to state what data is needed rather than how it is to be retrieved.

See Section 2.2.2.

2.10 *Name four object-based data models.*

- Entity-Relationship (ER)
- Semantic
- Functional
- Object-oriented.

See Section 2.3.1.

2.11 *Name three record-based data models. Discuss the main differences between these data models.*

- **relational data model** - data and relationships are represented as tables, each of which has a number of columns with a unique name.
- **network data model** - data is represented as collections of *records*, and relationships are represented by *sets*. Compared with the relational model, relationships are explicitly modeled by the sets, which become pointers in the implementation. The records are organized as generalized graph structures with records appearing as *nodes* (also called *segments*) and sets as *edges* in the graph.
- **hierarchical data model** - restricted type of network model. Again, data is represented as collections of *records* and relationships are represented by *sets*. However, the hierarchical model allows a node to have only one parent. A hierarchical model can be represented as a tree graph, with records appearing as *nodes* (also called *segments*) and sets as *edges*.

See Section 2.3.2.

2.12 *What is a transaction? Give an example of a transaction.*

A transaction is a series of actions, carried out by a single user or application program, which accesses or changes the contents of the database.

See Section 2.4.

2.13 *What is concurrency control and why does a DBMS need a concurrency control facility?*

A mechanism to ensure that the database is updated correctly when multiple users are updating the database concurrently.

This avoids inconsistencies from arising when two or more transactions are executing and at least one is updating the database.

See Section 2.4.

2.14 *Define the term "database integrity". How does database integrity differ from database security?*

"Database integrity" refers to the correctness and consistency of stored data: it can be considered as another type of database protection. Although integrity is related to security, it has wider implications: integrity is concerned with the quality of data itself. Integrity is usually expressed in terms of *constraints*, which are consistency rules that the database is not permitted to violate.

See Section 2.4.

Chapter 3 – Database Architectures and the Web – Answers to Review Questions

3.1 *What is meant by the term 'client-server architecture' and what are the advantages of this approach? Compare the client-server architecture with two other architectures.*

The client is a process that requires some resource, and the server provides the resource. Neither need reside on the same machine. Advantages include:

- Better performance
- Likely reduction in hardware costs
- Reduction in communication costs
- Better consistency

See also Section 3.1.

3.2 *Compare and contrast the two-tier client-server architecture for traditional DBMSs with the three-tier client-server architecture. Why is the latter architecture more appropriate for the Web?*

See Figures 3.5 and 3.6. Architecture maps quite naturally to the Web with a Web browser acting as 'thin' client and Web server acting as an application server (with database server as third layer).

3.3 *What is an n-tier architecture?*

The three-tier architecture can be expanded to *n* tiers, with additional tiers providing more flexibility and scalability.

3.4 *What is middleware? Provide a classification service for middleware.*

Middleware is a generic term used to describe software that mediates with other software and allows for communication between disparate applications in a heterogeneous system. The need for middleware arises when distributed systems become too complex to manage efficiently without a common interface. The need to make heterogeneous systems work efficiently across a network and be flexible enough to incorporate frequent modifications led to the development of middleware, which hides the underlying complexity of distributed systems.

3.5 *What is a TP Monitor? What advantages does a TP Monitor bring to an OLTP environment?*

A TP Monitor forms the middle tier of a three-tier architecture. TP Monitors provide significant advantages, including:

- *Transaction routing:* The TP Monitor can increase scalability by directing transactions to specific DBMSs.
- *Managing distributed transactions:* The TP Monitor can manage transactions that require access to data held in multiple, possibly heterogeneous, DBMSs. For example, a transaction may require to update data items held in an Oracle DBMS at site 1, an Informix DBMS at site 2, and an IMS DBMS as site 3. TP Monitors normally control transactions using the X/Open Distributed Transaction Processing (DTP)

standard. A DBMS that supports this standard can function as a resource manager under the control of a TP Monitor acting as a transaction manager.

- *Load balancing:* The TP Monitor can balance client requests across multiple DBMSs on one or more computers by directing client service calls to the least loaded server. In addition, it can dynamically bring in additional DBMSs as required to provide the necessary performance.
- *Funneling:* In environments with a large number of users, it may sometimes be difficult for all users to be logged on simultaneously to the DBMS. In many cases, we would find that users generally do not need continuous access to the DBMS. Instead of each user connecting to the DBMS, the TP Monitor can establish connections with the DBMSs as and when required, and can funnel user requests through these connections. This allows a larger number of users to access the available DBMSs with a potentially much smaller number of connections, which in turn would mean less resource usage.
- *Increased reliability:* The TP Monitor acts as a *transaction manager*, performing the necessary actions to maintain the consistency of the database, with the DBMS acting as a *resource manager*. If the DBMS fails, the TP Monitor may be able to resubmit the transaction to another DBMS or can hold the transaction until the DBMS becomes available again.

3.6 *What is a Web service?*

Web services allow applications to integrate with other applications across the Internet and may be a key technology that supports B2B (Business to Business) interaction. Unlike other Web-based applications, Web services have no user interface and are not aimed at Web browsers. Web services instead share business logic, data, and processes through a programmatic interface across a network. In this way, it is the applications that interface and not the users. Developers can then add the Web service to a Web page (or an executable program) to offer specific functionality to users.

3.7 *What technologies and standards are used to develop Web services and how do they relate to each other?*

Key to the Web services approach is the use of widely accepted technologies and standards, such as:

- XML (eXtensible Markup Language).
- SOAP (Simple Object Access Protocol) is a communication protocol for exchanging structured information over the Internet and uses a message format based on XML. It is both platform- and language-independent.
- WSDL (Web Services Description Language) protocol, again based on XML, is used to describe and locate a Web service.
- UDDI (Universal Discovery, Description, and Integration) protocol is a platform-independent, XML-based registry for businesses to list themselves on the Internet. It was designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the Web services listed in its directory.

Figure 3.9 illustrates the relationship between these technologies. From the database perspective, Web services can be used both from within the database (to invoke an external Web service as a *consumer)* and the Web service itself can access its own database (as a *provider)* to maintain the data required to provide the requested service.

3.8 What is a service-oriented architecture?

The SOA approach attempts to design loosely coupled and autonomous *services* that can be combined to provide flexible composite business processes and applications. SOA principles provide a unique design approach for building Web services for SOA:

- *loose coupling:* services must be designed to interact on a loosely coupled basis;
- *reusability:* logic that can potentially be reused is designed as a separate service;
- *contract:* services adhere to a communications contract that defines the information exchange and any additional service description information, specified by one or more service description documents;
- *abstraction:* beyond what is described in the service contract, services hide logic from the outside world;
- *composability:* services may compose other services, so that logic can be represented at different levels of granularity thereby promoting reusability and the creation of abstraction layers;
- *autonomy:* services have control over the logic they encapsulate and are not dependent upon other services to execute this governance;
- *stateless:* services should not be required to manage state information, as this can affect their ability to remain loosely-coupled;
- *discoverability:* services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

3.9 Provide an example of a service-oriented architecture.

An example of SOA is shown in Figure 3.10(b).

3.10 What is Cloud computing?

Cloud computing: A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

See Section 3.5.

3.11 Discuss the five essential characteristics of cloud computing.

The essential characteristics are:

- *On-demand self-service:* consumers can obtain, configure and deploy cloud services themselves using cloud service catalogues, without requiring the assistance of anyone from the cloud provider.
- *Broad network access:* it is network based, and accessible from anywhere, from any standardized platform (e.g. desktop computers, laptops, mobile devices).
- *Resource pooling.* The cloud provider's computing resources are pooled to serve multiple consumers, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Examples of resources include storage, processing, memory, and network bandwidth.
- *Rapid elasticity.* Resource pooling avoids the capital expenditure required for the establishment of network and computing infrastructure. By outsourcing to a cloud, consumers can cater for the spikes in demand for their services by using the cloud provider's computing capacity and the risk of outages and service interruptions are significantly reduced. Moreover, capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly based on demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be called on in any quantity at any time.
- *Measured service.* Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and charged for.

See Section 3.5.

3.12 *Discuss the three main service models of cloud computing.*

- *Software as a Service (SaaS)*: software and associated data are centrally hosted on the cloud.
- *Platform as a Service (PaaS)*: a computing platform that allows the creation of web applications quickly and easily and without the complexity of buying and maintaining the software and infrastructure underneath it. Sometimes, PaaS is used to extend the capabilities of applications developed as SaaS. While earlier application development required hardware, an operating system, a database, middleware, Web servers, and other software, with the PaaS model only the knowledge to integrate them is required. The rest is taken care of by the PaaS provider.
- *Infrastructure as a Service (IaaS):* delivers servers, storage, network and operating systems – typically a platform virtualization environment – to consumers as an on-demand service, in a single bundle and billed according to usage. A popular use of IaaS is in hosting websites, where the in-house infrastructure is not burdened with this task but left free to manage the business.

3.13 *Compare and contrast the four main deployment models for the cloud.*

The four main deployment models for the cloud are:

- *Private cloud:* cloud infrastructure operated solely for a single organization, whether managed internally by the organization, a third party, or some combination of them, and it may be hosted internally or externally.
- *Community cloud:* cloud infrastructure is shared for exclusive use by a specific community of organizations that have common concerns (e.g., security requirements, compliance, jurisdiction). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may be hosted internally or externally.
- *Public cloud:* cloud infrastructure is made available to the general public by a service provider. These services are free or offered on a pay-per-use model. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of these. It exists on the premises of the cloud provider.
- *Hybrid cloud*: cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology, offering the benefits of multiple deployment models.

3.14 *What are the difference between Data as a service (DaaS) and Database as a service (DBaaS)?*

- **DaaS:** offers the ability to define data in the cloud and subsequently query that data on demand. Unlike traditional database solutions, DaaS does not implement typical DBMS interfaces such as SQL (see Chapter 6). Instead, the data is accessed via a common set of APIs. DaaS enables any organization with valuable data to create new revenue lines based on data they already own. Examples of DaaS are Urban Mapping, a geography data service, provides data for customers to embed into their own websites and applications; Xignite makes financial data available to customers; and Hoovers, a Dun & Bradstreet company, provides business data on various organizations.

- **DBaaS:** offers full database functionality to application developers. In DBaaS, a management layer is responsible for the continuous monitoring and configuring of the database to achieve optimized scaling, high availability, multi-tenancy (that is, serving multiple client organizations), and effective resource allocation in the cloud, thereby sparing the developer from ongoing database administration tasks.

See Section 3.5.2.

3.15 *Discuss the different architectural models for Database-as-a-service.*

- Separate servers
- Shared Server, Separate Database Server Process
- Shared Database Server, Separate Databases
- Shared Database, Separate Schema
- Shared Database, Shared Schema

See Section 3.5.2.

3.16 Describe the main components in a DBMS.

The major software components in a DBMS environment are depicted in Figure 3.20. The main components in a DBMS are:

- *Query processor.* This is a major DBMS component that transforms queries into a series of low-level instructions directed to the database manager.
- *Database manager (DM).* The DM interfaces with user-submitted application programs and queries. The DM accepts queries and examines the external and conceptual schemas to determine what conceptual records are required to satisfy the request. The DM then places a call to the file manager to perform the request.
- *File manager*. The file manager manipulates the underlying storage files and manages the allocation of storage space on disk. It establishes and maintains the list of structures and indexes defined in the internal schema. If hashed files are used, it calls on the hashing functions to generate record addresses. However, the file manager does not directly manage the physical input and output of data. Rather, it passes the requests on to the appropriate access methods, which either read data from or write data into the system buffer (or *cache*).
- *DML preprocessor*. This module converts DML statements embedded in an application program into standard function calls in the host language. The DML preprocessor must interact with the query processor to generate the appropriate code.
- *DDL compiler*. The DDL compiler converts DDL statements into a set of tables containing metadata. These tables are then stored in the system catalog while control information is stored in data file headers.
- *Catalog manager*. The catalog manager manages access to and maintains the system catalog. The system catalog is accessed by most DBMS components.

3.17 Describe the internal architecture of Oracle.

Oracle is based on the client–server architecture. The Oracle server consists of the database (the raw data, including log and control files) and the instance (the processes and system memory on the server that provide access to the database). An instance can connect to only one database. The database consists of a logical structure, such as the database schema, and a physical structure, containing the files that make up an Oracle database.

Chapter 4 – The Relational Model – Answers to Review Questions

4.1 *Discuss each of the following concepts in the context of the relational data model:*

(a)	*Relation*	A table with columns and rows.
(b)	*Attribute*	A named column of a relation.
(c)	*Domain*	The set of allowable values for one or more attributes.
(d)	*Tuple*	A row of a relation.
(e)	*Intension*	The structure of a relation together with a specification of the domains and any other restrictions on possible values.
	Extension	An instance of the tuples of a relation.
(f)	*Degree*	The number of attributes in a relation.
	Cardinality	The number of tuples in a relation.

Each term defined in Section 4.2.1.

4.2 *Describe the relationship between mathematical relations and relations in the relational data model?*

Let $D_1, D_2, \ldots, D_n$ be n sets. Their Cartesian product is defined as:

$$D_1 \otimes D_2 \otimes \ldots \otimes D_n = \{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}$$

Any set of n-tuples from this Cartesian product is a relation on the n sets. Now let $A_1, A_2, \ldots, A_n$ be attributes with domains $D_1, D_2, \ldots, D_n$. Then the set $\{A_1{:}D_1, A_2{:}D_2, \ldots, A_n{:}D_n\}$ is a relation schema. A relation R defined by a relation schema S is a set of mappings from the attribute names to their corresponding domains. Thus, relation R is a set of n-tuples:

$(A_1{:}d_1, A_2{:}d_2, \ldots, A_n{:}d_n)$ such that $d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n$

Each element in the n-tuple consists of an attribute and a value for that attribute.

Discussed fully in Sections 4.2.2 and 4.2.3.

4.3 *Describe the differences between a relation and a relation schema. What is a relational database schema?*

A relation schema is a named relation defined by a set of attribute and domain name pairs. A relational database schema is a set of relation schemas, each with a distinct name. Discussed in Section 4.2.3.

4.4 *Discuss the properties of a relation.*

A relation has the following properties:

- has a name that is distinct from all other relation names in the relational schema;
- each cell contains exactly one atomic (single) value;
- each attribute has a distinct name;
- the values of an attribute are all from the same domain;

- each tuple is distinct; there are no duplicate tuples;
- the order of attributes has no significance;
- the order of tuples has no significance, theoretically. (However, in practice, the order may affect the efficiency of accessing tuples.)

Discussed fully in Section 4.2.4.

4.5 *Discuss the differences between the candidate keys and the primary key of a relation. Explain what is meant by a foreign key. How do foreign keys of relations relate to candidate keys? Give examples to illustrate your answer.*

The primary key is the candidate key that is selected to identify tuples uniquely within a relation. A foreign key is an attribute or set of attributes within one relation that matches the candidate key of some (possibly the same) relation. Discussed in Section 4.2.5.

4.6 *Define the two principal integrity rules for the relational model. Discuss why it is desirable to enforce these rules.*

Two rules are Entity Integrity (Section 4.3.2) and Referential Integrity (Section 4.3.3).

4.7 *What is a view? Discuss the difference between a view and a base relation.*

View is the dynamic result of one or more relational operations operating on the base relations to produce another relation. Base relation exists as a set of data in the database. A view does not contain any data, rather a view is defined as a query on one or more base relations and a query on the view is translated into a query on the associated base relations. See Section 4.4.

Chapter 5 – Relational Algebra and Relational Calculus – Answers to Review Questions and Exercises

5.1 *What is the difference between a procedural and non-procedural language? How would you classify the relational algebra and relational calculus?*

Procedural language: a language that allows user to tell the system what data is needed and exactly ***how*** to retrieve the data.
Non-procedural language: a language that allows user to state ***what*** data is needed rather than how it is to be retrieved.

Informally, we may describe the relational algebra as a (high-level) procedural language: it can be used to tell the DBMS how to build a new relation from one or more relations in the database. Again, informally, we may describe the relational calculus as a non-procedural language: it can be used to formulate the definition of a relation in terms of one or more database relations.

5.2 *Explain the following terms:*

- *relationally complete;*

  A language that can be used to produce any relation that can be derived using the relational calculus is said to be relationally **complete**.

- *closure of relational operations.*

  The relational algebra is a theoretical language with operations that work on one or more relations to define another relation without changing the original relation(s). Thus, both the operands and the results are relations, and so the output from one operation can become the input to another operation. This allows expressions to be nested in the relational algebra, just as we can nest arithmetic operations. This property is called **closure**: relations are closed under the algebra, just as numbers are closed under arithmetic operations.

5.3 *Define the five basic relational algebra operations. Define the Join, Intersection, and Division operations in terms of these five basic operations.*

Five basic operations are:

- Selection and Projection (Unary)
- Cartesian Product, Union, and Set Difference (Binary).

There is also the Join, Intersection, and Division operations:

- Can rewrite θ-Join in terms of the basic selection and Cartesian product operations:

  $R \, 3_F \, S = \sigma_F (R \times S)$

- Can express the intersection operation in terms of the set difference operation:

$R \cap S = R - (R - S)$

- Can express the division operation in terms of the basic operations:

$T_1 = \Pi_C(R)$
$T_2 = \Pi_C((S \times T_1) - R)$
$T = T_1 - T_2$

5.4 *Discuss the differences between the five Join operations: Theta join, Equijoin, Natural join, Outer join, and Semijoin. Give examples to illustrate your answer.*

Theta join	$R \bowtie_F S$	Produces a relation that contains tuples satisfying the predicate F from the Cartesian product of R and S.
Equijoin	$R \bowtie_F S$	Produces a relation that contains tuples satisfying the predicate F (which only contains equality comparisons) from the Cartesian product of R and S.
Natural join	$R \bowtie S$	An Equijoin of the two relations R and S over all common attributes x. One occurrence of each common attribute is eliminated.
(Left) Outer join	$R \ltimes S$	A join in which tuples from R that do not have matching values in the common attributes of S are also included in the result relation.
Semijoin	$R \ltimes_F S$	Produces a relation that contains the tuples of R that participate in the join of R with S.

5.6 *Define the structure of a (well-formed) formula in both the tuple relational calculus and domain relational calculus.*

Tuple relational calculus

A (well-formed) formula is made out of one or more *atoms*, where an atom has one of the following forms:

- $R(S_i)$, where S_i is a tuple variable and R is a relation.
- $S_i.a_1 \ \theta \ S_j.a_2$, where S_i and S_j are tuple variables, a_1 is an attribute of the relation over which S_i ranges, a_2 is an attribute of the relation over which S_j ranges, and θ is one of the comparison operators ($<, \leq, >, \geq, =, \neq$); the attributes a_1 and a_2 must have domains whose members can be compared by θ.
- $S_i.a_1 \ \theta \ c$, where S_i is a tuple variable, a_1 is an attribute of the relation over which S_i ranges, c is a constant from the domain of attribute a_1, and θ is one of the comparison operators

We recursively build up formulae from atoms using the following rules:

- an atom is a formula;
- if F_1 and F_2 are formulae, so are their conjunction $F_1 \wedge F_2$, their disjunction $F_1 \vee F_2$, and the negation $\sim F_1$;

- if F is a formula with free variable X, then $(\exists X)(F)$ and $(\forall X)(F)$ are also formulae.

5.7 *Explain how a relational calculus expression can be unsafe? Illustrate your answer with an example. Discuss how to ensure that a relational calculus expression is safe.*

See end of Section 5.2.1.

For the following exercises, use the Hotel schema defined at the start of the Exercises at the end of Chapter 4.

5.8 *Describe the relations that would be produced by the following relational algebra operations:*

a) $\Pi_{\text{hotelNo}}(\sigma_{\text{price} > 50}(\text{Room}))$

This will produce a relation with a single attribute (hotelNo) giving the number of those hotels with a room price greater than £50.

b) $\sigma_{\text{Hotel.hotelNo} = \text{Room.hotelNo}}(\text{Hotel} \times \text{Room})$

This will produce a join of the Hotel and Room relations containing all the attributes of both Hotel and Room (there will be two copies of the hotelNo attribute). Essentially this will produce a relation containing all rooms at all hotels.

c) $\Pi_{\text{hotelName}}(\text{Hotel} \bowtie_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\sigma_{\text{price} > 50}(\text{Room})))$

This will produce a join of Hotel and those tuples of Room with a price greater than £50. Essentially this will produce a relation containing all hotel names with a room price above £50.

d) $\text{Guest} \ltimes (\sigma_{\text{dateTo} \geq \text{'1-Jan-2007'}}(\text{Booking}))$

This will produce a (left outer) join of Guest and those tuples of Booking with an end date (dateTo) greater than or equal to 1-Jan-2007. All guests who don't have a booking with such a date will still be included in the join. Essentially this will produce a relation containing all guests and show the details of any bookings they have beyond 1-Jan-2002.

e) $\text{Hotel} \rhd_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\sigma_{\text{price} > 50}(\text{Room}))$

This will produce a (semi) join of Hotel and those tuples of Room with a price greater than £50. Only those Hotel attributes will be listed. Essentially this will produce a relation containing all the details of all hotels with a room price above £50.

f) $\Pi_{\text{guestName, hotelNo}}(\text{Booking} \bowtie_{\text{Booking.guestNo} = \text{Guest.guestNo}} \text{Guest}) \div \Pi_{\text{hotelNo}}(\sigma_{\text{city} = \text{'London'}}(\text{Hotel}))$

This will produce a relation containing the names of all guests who have booked all hotels in London.

5.10 Describe the relations that would be produced by the following tuple relational calculus expressions:

(a) {H.hotelName | Hotel(H) ∧ H.city = 'London'}

This will produce a relation containing the names of all hotels in London.

(b) {H.hotelName | Hotel(H) ∧ (∃R) (Room(R) ∧ H.hotelNo = R.hotelNo ∧ R.price > 50)}

This will produce a relation containing the names of all hotels that have a room price above £50.

(c) {H.hotelName | Hotel(H) ∧ (∃B) (∃G) (Booking(B) ∧ Guest(G) ∧ H.hotelNo = B.hotelNo ∧ B.guestNo = G.guestNo ∧ G.guestName = 'John Smith')}

This will produce a relation containing the names of all hotels that have a booking for a guest called John Smith.

(d) {H.hotelName, G.guestName, B1.dateFrom, B2.dateFrom | Hotel(H) ∧ Guest(G) ∧ Booking(B1) ∧ Booking(B2) ∧ H.hotelNo = B1.hotelNo ∧ G.guestNo = B1.guestNo ∧ B2.hotelNo = B1.hotelNo ∧ B2.guestNo = B1.guestNo ∧ B2.dateFrom <> B1.dateFrom}

This will produce a relation containing the names of guests who have more than one booking at the same hotel, along with the hotel number and the dates of the bookings.

5.12 Generate the relational algebra, tuple relational calculus, ~~and domain relational calculus~~ expressions for the following queries:

(a) *List all hotels.*

RA: Hotel

TRC: {H | Hotel(H)}

(b) *List all single rooms with a price below £20 per night.*

RA: $\sigma_{type='S' \wedge price < 20}(Room)$

TRC: {R | Room(R) ∧ R.type = 'S' ∧ R.price < 20}

(c) *List the names and cities of all guests.*

RA: $\Pi_{guestName, guestAddress}(Guest)$

TRC: {G.guestName, G.guestAddress | Guest(G)}

(d) *List the price and type of all rooms at the Grosvenor Hotel.*

RA: $\Pi_{price, type}(Room \rhd\lhd_{hotelNo} (\sigma_{hotelName = 'Grosvenor\ Hotel'}(Hotel)))$

TRC: {R.price, R.type | Room(R) ∧ (∃H) (Hotel(H) ∧ (□R.hotelNo = H.hotelNo) ∧
(H.hotelName = 'Grosvenor Hotel'))}

(e) *List all guests currently staying at the Grosvenor Hotel.*

RA: $Guest \rhd\lhd_{guestNo} (\sigma_{dateFrom \leq '01\text{-}01\text{-}15' \wedge dateTo \geq '01\text{-}01\text{-}15'} ($
$Booking \rhd\lhd_{hotelNo} (\sigma_{hotelName = 'Grosvenor\ Hotel'}(Hotel))))$

(substitute '01-01-15' for today's date).

TRC: {G |Guest(G) ∧ ((∃B)(∃H) (Booking(B) ∧ Hotel(H) ∧ (B.dateFrom ≤ '01-01-15') ∧
(B.dateTo ≥ '01-01-15') ∧ (B.guestNo = G.guestNo) ∧
(B.hotelNo = H.hotelNo) ∧ (H.hotelName = 'Grosvenor Hotel'))))}

(f) *List the details of all rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied.*

RA: $(Room \rhd\lhd_{hotelNo} (\sigma_{hotelName = 'Grosvenor\ Hotel'}(Hotel))) \supset\lhd$ // Outer Join
$\Pi_{guestName, hotelNo, roomNo}($
$(Guest \rhd\lhd_{guestNo} (\sigma_{dateFrom \leq '01\text{-}01\text{-}15' \wedge dateTo \geq '01\text{-}01\text{-}15'} ($
$Booking \rhd\lhd_{hotelNo} (\sigma_{hotelName='Grosvenor\ Hotel'}(Hotel))))$

(substitute '01-01-15' for today's date).

TRC: {R, G.guestName | (Room(R) ∧ (∃H)(Hotel(H) ∧
(R.hotelNo = H.hotelNo) ∧ (H.hotelName = 'Grosvenor Hotel'))) ∨
(Guest(G) ∧ ((∃B)(∃H) (Booking(B) ∧ Hotel(H) ∧
(G.guestNo = B.guestNo) ∧ (B.hotelNo = H.hotelNo) ∧
(H.hotelName = 'Grosvenor Hotel') ∧
(B.dateFrom ≤ '01-01-15' ∧ B.dateTo ≥ '01-01-15'))))}

(g) *List the guest details (guestNo, guestName, and guestAddress) of all guests staying at the Grosvenor Hotel.*

RA: $\Pi_{guestNo,\ guestName,\ guestAddress}(Guest \rhd\lhd_{guestNo} (\sigma_{dateFrom \leq \text{'01-01-15'} \wedge dateTo \geq \text{'01-01-15'}} ($

$Booking \rhd\lhd_{hotelNo} (\sigma_{hotelName=\text{'Grosvenor Hotel'}}(Hotel)))))$

(substitute '01-01-15' for today's date).

TRC: $\{G \mid Guest(G) \wedge ((\exists B)(\exists H)(Booking(B) \wedge Hotel(H) \wedge (B.guestNo = G.guestNo) \wedge$

$(B.hotelNo = H.hotelNo) \wedge (H.hotelName = \text{'Grosvenor Hotel'}) \wedge$

$(B.dateFrom \leq \text{'01-01-15'} \wedge B.dateTo \geq \text{'01-01-15'})))\}$

Chapter 6 – SQL: Data Manipulation – Answers to Review Questions and Exercises

6.1 *What are the two major components of SQL and what function do they serve?*

A data definition language (DDL) for defining the database structure.
A data manipulation language (DML) for retrieving and updating data.

6.2 *What are the advantages and disadvantages of SQL?*

Advantages

- Satisfies ideals for database language
- (Relatively) Easy to learn
- Portability
- SQL standard exists
- Both interactive and embedded access
- Can be used by specialist and non-specialist.

Disadvantages

- Impedance mismatch - mixing programming paradigms with embedded access
- Lack of orthogonality - many different ways to express some queries
- Language is becoming enormous (SQL-92 is 6 times larger than predecessor)
- Handling of nulls in aggregate functions
- Result tables are not strictly relational - can contain duplicate tuples, imposes an ordering on both columns and rows.

6.3 *Explain the function of each of the clauses in the SELECT statement. What restrictions are imposed on these clauses?*

FROM	Specifies the table or tables to be used.
WHERE	Filters the rows subject to some condition.
GROUP BY	Forms groups of rows with the same column value.
HAVING	Filters the groups subject to some condition.
SELECT	Specifies which columns are to appear in the output.
ORDER BY	Specifies the order of the output.

If the SELECT list includes an aggregate function and no GROUP BY clause is being used to group data together, then no item in the SELECT list can include any reference to a column unless that column is the argument to an aggregate function.

When GROUP BY is used, each item in the SELECT list must be **single-valued per group**. Further, the SELECT clause may only contain:

- Column names.
- Aggregate functions.
- Constants.
- An expression involving combinations of the above.

All column names in the SELECT list must appear in the GROUP BY clause unless the name is used only in an aggregate function.

6.4 *What restrictions apply to the use of the aggregate functions within the SELECT statement? How do nulls affect the aggregate functions?*

An aggregate function can be used only in the SELECT list and in the HAVING clause.

Apart from COUNT(*), each function eliminates nulls first and operates only on the remaining non-null values. COUNT(*) counts all the rows of a table, regardless of whether nulls or duplicate values occur.

6.5 *Explain how the GROUP BY clause works. What is the difference between the WHERE and HAVING clauses?*

SQL first applies the WHERE clause. Then it conceptually arranges the table based on the grouping column(s). Next, applies the HAVING clause and finally orders the result according to the ORDER BY clause.

WHERE filters rows subject to some condition; HAVING filters groups subject to some condition.

6.6 *What is the difference between a subquery and a join? Under what circumstances would you not be able to use a subquery?*

With a subquery, the columns specified in the SELECT list are restricted to one table. Thus, cannot use a subquery if the SELECT list contains columns from more than one table.

For the Exercises 6.7 – 6.28, use the Hotel schema defined at the start of the Exercises at the end of Chapter 3.

Simple Queries

6.7 *List full details of all hotels.*

SELECT * FROM Hotel;

6.8 *List full details of all hotels in London.*

SELECT * FROM Hotel WHERE city = ‘London’;

6.9 *List the names and addresses of all guests in London, alphabetically ordered by name.*

SELECT guestName, guestAddress FROM Guest WHERE address LIKE ‘%London%’ ORDER BY guestName;

Strictly speaking, this would also find rows with an address like: '10 London Avenue, New York'.

6.10 *List all double or family rooms with a price below £40.00 per night, in ascending order of price.*

```
SELECT * FROM Room WHERE price < 40 AND type IN ('D', 'F')
ORDER BY price;
```

(Note, ASC is the default setting).

6.11 *List the bookings for which no dateTo has been specified.*

```
SELECT * FROM Booking WHERE dateTo IS NULL;
```

Aggregate Functions

6.12 *How many hotels are there?*

```
SELECT COUNT(*) FROM Hotel;
```


6.13 *What is the average price of a room?*

```
SELECT AVG(price) FROM Room;
```


6.14 *What is the total revenue per night from all double rooms?*

```
SELECT SUM(price) FROM Room WHERE type = 'D';
```


6.15 *How many different guests have made bookings for August?*

```
SELECT COUNT(DISTINCT guestNo) FROM Booking
WHERE (dateFrom <= DATE'2004-08-01' AND dateTo >= DATE'2004-08-01') OR
      (dateFrom >= DATE'2004-08-01' AND dateFrom <= DATE'2004-08-31');
```

Subqueries and Joins

6.16 *List the price and type of all rooms at the Grosvenor Hotel.*

```
SELECT price, type FROM Room
WHERE hotelNo =
            (SELECT hotelNo FROM Hotel
            WHERE hotelName = 'Grosvenor Hotel');
```

6.17 *List all guests currently staying at the Grosvenor Hotel.*

```
SELECT * FROM Guest
WHERE guestNo =
```

```
                (SELECT guestNo FROM Booking
                WHERE dateFrom <= CURRENT_DATE AND
                        dateTo >= CURRENT_DATE AND
                        hotelNo =
                                (SELECT hotelNo FROM Hotel
                                WHERE hotelName = 'Grosvenor Hotel'));
```

6.18 *List the details of all rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied.*

```
SELECT r.* FROM Room r LEFT JOIN
        (SELECT g.guestName, h.hotelNo, b.roomNo FROM Guest g, Booking b, Hotel h
        WHERE g.guestNo = b.guestNo AND b.hotelNo = h.hotelNo AND
          hotelName= 'Grosvenor Hotel' AND
          dateFrom <= CURRENT_DATE AND
          dateTo >= CURRENT_DATE) AS XXX
ON r.hotelNo = XXX.hotelNo AND r.roomNo = XXX.roomNo;
```

6.19 *What is the total income from bookings for the Grosvenor Hotel today?*


```
SELECT SUM(price) FROM Booking b, Room r, Hotel h
WHERE (dateFrom <= CURRENT_DATE AND
        dateTo >= CURRENT_DATE) AND
        r.hotelNo = h.hotelNo AND r.roomNo = b.roomNo AND
        hotelName = 'Grosvenor Hotel';
```


6.20 *List the rooms that are currently unoccupied at the Grosvenor Hotel.*

```
SELECT * FROM Room r
WHERE roomNo NOT IN
        (SELECT roomNo FROM Booking b, Hotel h
        WHERE (dateFrom <= CURRENT_DATE AND
                dateTo >= CURRENT_DATE) AND
                b.hotelNo = h.hotelNo AND hotelName = 'Grosvenor Hotel');
```

6.21 *What is the lost income from unoccupied rooms at the Grosvenor Hotel?*

```
SELECT SUM(price) FROM Room r
WHERE roomNo NOT IN
        (SELECT roomNo FROM Booking b, Hotel h
        WHERE (dateFrom <= CURRENT_DATE AND
                dateTo >= CURRENT_DATE) AND
                b.hotelNo = h.hotelNo AND hotelName = 'Grosvenor Hotel');
```

Grouping

6.22 *List the number of rooms in each hotel.*

```
SELECT hotelNo, COUNT(roomNo) AS count FROM Room
GROUP BY hotelNo;
```

6.23 *List the number of rooms in each hotel in London.*

```
SELECT hotelNo, COUNT(roomNo) AS count FROM Room r, Hotel h
WHERE r.hotelNo = h.hotelNo AND city = 'London'
GROUP BY hotelNo;
```

6.24 *What is the average number of bookings for each hotel in August?*

```
SELECT AVG(X)
FROM ( SELECT hotelNo, COUNT(hotelNo) AS X
        FROM Booking b
        WHERE (dateFrom <= DATE'2004-08-01' AND
                dateTo >= DATE'2004-08-01') OR
                (dateFrom >= DATE'2004-08-01' AND
                dateFrom <= DATE'2004-08-31')
        GROUP BY hotelNo);
```

Yes - this is legal in SQL-92!

6.25 *What is the most commonly booked room type for each hotel in London?*


```
SELECT MAX(X)
FROM ( SELECT type, COUNT(type) AS X
        FROM Booking b, Hotel h, Room r
        WHERE r.roomNo = b.roomNo AND b.hotelNo = h.hotelNo AND
                city = 'London'
        GROUP BY type);
```

6.26 *What is the lost income from unoccupied rooms at each hotel today?*

```
SELECT hotelNo, SUM(price) FROM Room r
WHERE roomNo NOT IN
        (SELECT roomNo FROM Booking b, Hotel h
        WHERE (dateFrom <= CURRENT_DATE AND
                dateTo >= CURRENT_DATE) AND
                b.hotelNo = h.hotelNo)
GROUP BY hotelNo;
```

Populating Tables

6.27 *Insert records into each of these tables.*

```
INSERT INTO Hotel
VALUES ('H111', 'Grosvenor Hotel', 'London');

INSERT INTO Room
VALUES ('1', 'H111', 'S', 72.00);
INSERT INTO Guest
VALUES ('G111', 'John Smith', 'London');
INSERT INTO Booking
VALUES ('H111', 'G111', DATE'2005-01-01', DATE'2005-01-02', '1');
```

6.28 *Update the price of all rooms by 5%.*

```
UPDATE Room SET price = price*1.05;
```

COMP 3611 Module 3 Unit 1 SQL DML Exercises - SOLUTIONS

This document is a placeholder for the SQL DML Exercises using the Just Books demo database.

Part A – Simple Queries

1. Display all the data from the **books** table

```
SELECT *
FROM books;
```


2. Display just the titles for all the books.

```
SELECT title
FROM books;
```


3. Display the titles and publication date (make this the heading) for all the books.

```
SELECT title, pubdate "Publication Date"
FROM books;
```

4. Display each customer number along with the city and state.

```
SELECT customer#, city, state
FROM customers;
```

5. Display the list of categories from the books table (with no duplicates).

```
SELECT DISTINCT category                or UNIQUE
FROM books;
```

6. Display the customer ids that have placed orders (with no duplicates).

```
SELECT DISTINCT customer#          or UNIQUE
FROM orders;
```

7. Display all the author names in the format “lastname, firstname”.

```
SELECT lname || ', ' || fname
FROM author;
```

8. Display each of the order items along with the calculated total paid.

```
SELECT order#, item#, isbn, quantity, paideach, quantity*paideach "Item Total"
FROM orderitems;
```


9. Display for each book the title and the profit margin i.e. (retail price – cost) / cost, as a percentage with a nice heading.

```
SELECT title, (retail-cost)/cost*100 "Profit %"
FROM books;
```

Part B – Insert, Update, Delete DML

1. Add a new order with order# 1021, customer 1009, and order date July 20 2019.

```
INSERT INTO orders (order#, customer#, orderdate)
 VALUES (1021, 1009, '20-JUL-19');
```

2. Change order# 1017 shipping zip code to 33222.

```
UPDATE orders
 SET shipzip = '33222'
 WHERE order# = 1017;
```


3. Make the above changes permanent.

```
COMMIT
```


4. Attempt to add a new order with order# 1022, customer 2000, and order date Aug 6 2019. What occurs? Why?


```
INSERT INTO orders (order#, customer#, orderdate)
 VALUES (1022, 2000, '06-AUG-19');
```

**Foreign key error due to customer 2000 not existing in the CUSTOMERS table

5. Attempt to add a new order with order# 1023, customer 1009, omitting any order date. What occurs? Why?

```
INSERT INTO orders (order#, customer#)
 VALUES (1023, 1009);
```

**Constraint error due to Orderdate having a NOT NULL constraint

6. Return the database to the previous stable state.

```
ROLLBACK;
```

7. Attempt to delete order# 1005. What occurs? Why?

```
DELETE FROM orders
 WHERE order# = 1005;
```

Note: Deleting the master order record from the ORDERS table raises an error because child records exist in the ORDERITEMS table. To eliminate an order, the child records and then the master record must be deleted.

```
DELETE FROM orderitems
 WHERE order# = 1005;
DELETE FROM orders
 WHERE order# = 1005;
```


8. Discard these changes.

```
ROLLBACK;
```

Part C – Queries with selection and ordering

1. Display the customer names who reside in New Jersey.

```
SELECT lastname, firstname, state
FROM customers
WHERE state = 'NJ';
```

2. Display the order# and shipping date for any orders shipped after April 1 2019.

```
SELECT order#, shipdate
FROM orders
WHERE shipdate > '01-APR-19';
```


3. Display the title and category for any books not in the fitness category.


```
SELECT title, category
FROM books
WHERE category <> 'FITNESS';
```


4. Display the customer#, last name, and state for any customers in Georgia or New Jersey.

```
SELECT customer#, lastname, state
FROM customers
WHERE state = 'GA' OR state = 'NJ'
ORDER BY lastname;

SELECT customer#, lastname, state
FROM customers
WHERE state IN ('GA', 'NJ')
ORDER BY lastname;
```

5. Display the order# and order date for all orders placed before April 2 2019.

```
SELECT order#, orderdate
FROM orders
WHERE orderdate < '02-APR-19';

SELECT order#, orderdate
FROM orders
WHERE orderdate <= '01-APR-09';
```

6. Display the name of any author whose last name contains the letters “IN” in alphabetic order

```
SELECT lname, fname
FROM author
WHERE lname LIKE '%IN%'
ORDER BY lname, fname;
```


7. Display the last name and the referral for any customers where a referral was made.

```
SELECT lastname, referred
FROM customers
WHERE referred IS NOT NULL;
```

8. Display the titles for any books in the cooking or children categories.

```
SELECT title, category
FROM books
WHERE category LIKE '%C%N%';

SELECT title, category
FROM books
```

```
WHERE category = 'CHILDREN' or category = 'COOKING';
```

```
SELECT title, category
FROM books
WHERE category IN ('CHILDREN', 'COOKING');
```

9. Display the titles and publication date for any computer category books published in 2015.

```
SELECT title, pubdate
FROM books
WHERE category = 'COMPUTER'
  AND pubdate BETWEEN '01-JAN-15' AND '31-DEC-15';
```



```
SELECT title, pubdate
FROM books
WHERE category = 'COMPUTER'
  AND pubdate >= '01-JAN-15' AND pubdate <= '31-DEC-15';
```

```
SELECT title, pubdate
FROM books
WHERE category = 'COMPUTER' AND pubdate LIKE '%15';
```

Part D – Queries with joins

1. Display the title with publisher contact and phone for all books.

```
SELECT b.title, p.contact, p.phone
 FROM books b, publisher p
 WHERE b.pubid = p.pubid;


SELECT b.title, p.contact, p.phone
 FROM books b JOIN publisher p
  USING (pubid);
```


2. Display the customer name and order# for all orders who have not yet been shipped by order date.


```
SELECT c.firstname, c.lastname, o.order#
 FROM customers c, orders o
 WHERE c.customer# = o.customer#
  AND o.shipdate IS NULL
 ORDER BY o.orderdate;
```



```
SELECT c.firstname, c.lastname, o.order#
 FROM customers c JOIN orders o
    USING (customer#)
 WHERE o.shipdate IS NULL
 ORDER BY o.orderdate;
```

3. Display the customer name and id who have an address in Florida (FL) and have purchased a COMPUTER category book.

```
SELECT DISTINCT c.lastname, c.customer#
 FROM books b, orders o, orderitems i, customers c
```

```
WHERE c.customer# = o.customer#
 AND o.order# = i.order#
 AND i.isbn = b.isbn
 AND c.state = 'FL'
 AND b.category = 'COMPUTER';


SELECT DISTINCT c.lastname, customer#
FROM books b JOIN orderitems USING (isbn)
 JOIN orders USING (order#)
 JOIN customers c USING (customer#)
WHERE c.state = 'FL'
 AND b.category = 'COMPUTER';
```


4. Display the titles that have been ordered by customer Jake Lucas.

```
SELECT DISTINCT b.title
FROM customers c, orders o, orderitems i, books b
WHERE c.customer# = o.customer#
 AND o.order# = i.order#
 AND i.isbn = b.isbn
 AND c.firstname = 'JAKE'
 AND c.lastname = 'LUCAS';


SELECT DISTINCT b.title
FROM customers c JOIN orders USING (customer#)
  JOIN orderitems USING (order#)
  JOIN books b USING (isbn)
WHERE c.firstname = 'JAKE'
 AND c.lastname = 'LUCAS';
```

5. Modify the above query to include the profit made (paid – cost). Display the results in order days and profit made order.

```
SELECT b.title, i.paideach-cost
 FROM customers c, orders o, orderitems i, books b
 WHERE c.customer# = o.customer#
  AND o.order# = i.order#
  AND i.isbn = b.isbn
  AND c.firstname = 'JAKE'
  AND c.lastname = 'LUCAS'
 ORDER BY o.orderdate, i.paideach-b.cost DESC;
```



```
SELECT b.title, i.paideach-b.cost
 FROM customers c JOIN orders o USING (customer#)
```



```
   JOIN orderitems  i USING (order#)
   JOIN books b USING (isbn)
```



```
 WHERE c.firstname = 'JAKE'
  AND c.lastname = 'LUCAS'
 ORDER BY o.orderdate, i.paideach-b.cost DESC;
```

6. Display the titles for books written by an author with last name Adams.

```
SELECT b.title
 FROM books b, bookauthor ba, author a
 WHERE b.isbn = ba.isbn
  AND ba.authorid = a.authorid
  AND a.lname = 'ADAMS';


SELECT b.title
```

```
FROM books b JOIN bookauthor USING (isbn)
 JOIN author a USING (authorid)
WHERE a.lname = 'ADAMS';
```

7. Display the author name and book title for all books that have been ordered by customer Becca Nelson.

```
SELECT a.lname, a.fname, b.title
 FROM books b, orders o, orderitems i, customers c, bookauthor t, author a
 WHERE c.customer# = o.customer#
  AND o.order# = i.order#
  AND i.isbn = b.isbn
  AND b.isbn = t.isbn
  AND t.authorid = a.authorid
  AND c.firstname = 'BECCA'
  AND c.lastname = 'NELSON';
```



```
SELECT a.lname, a.fname, b.title
FROM customers c JOIN orders USING (customer#)
 JOIN orderitems USING (order#)
 JOIN books b USING (isbn)
 JOIN bookauthor USING (isbn)
 JOIN author a USING (authorid)
WHERE c.firstname = 'BECCA'
 AND c.lastname = 'NELSON';
```

8. Display the title for all books, along with order# and state that they may have been ordered from.

```
SELECT b.title, o.order#, c.state
```

```
FROM books b, orders o, orderitems i, customers c
WHERE c.customer# (+) = o.customer#
 AND o.order# (+) = i.order#
 AND i.isbn (+) = b.isbn;

SELECT b.title, order#, c.state
FROM books b LEFT OUTER JOIN orderitems i USING (isbn)
 LEFT OUTER JOIN orders USING (order#)
 LEFT OUTER JOIN customers c USING (customer#);
```

Part E – Queries using functions

1. Display all customer names in proper case (e.g. Joe Smith).

```
SELECT INITCAP(firstname), INITCAP(lastname)
FROM customers;
```

2. Display all customer names with the information whether they were referred or not (use the actual words REFERRED and NOT REFERRED).

```
SELECT firstname, lastname, NVL2(referred, 'REFERRED', 'NOT
REFERRED')
FROM customers;
```


3. Display each book title with the paid (paid – cost) for order 1002. Format the amount as dollars and cents.


```
SELECT title, TO_CHAR(quantity*(paideach-cost), '$999.99')
FROM books JOIN orderitems USING (isbn)
WHERE order# = 1002;
```


4. Display each book title with the percentage of retail over cost.

```
SELECT title, ROUND((retail-cost)/cost *100, 0)||'%'
FROM books;
```

5. Display the current day of the week name and time nicely formatted.

```
SELECT TO_CHAR(CURRENT_DATE, 'DAY, HH:MI:SS')
FROM dual;
```

6. Display each book title along with its cost. Pad the cost with leading asterisks (*) to a width of 12.

```
SELECT title, LPAD(cost, 12, '*')
FROM books;
```

7. Display the different sizes of ISBN found in the books.

```
SELECT DISTINCT LENGTH(isbn)
FROM books;
```

8. Display each book title along with its publication date, today's date, and how the book is in months as a column called AGE.


```
SELECT title, pubdate, SYSDATE,
  TRUNC(MONTHS_BETWEEN(SYSDATE, pubdate),0) Age
FROM books;
```


9. Display the date of the next Wednesday from today.

```
SELECT NEXT_DAY(SYSDATE, 'WEDNESDAY')
FROM dual;
```

Part F – Queries with summary operations

1. Display how many books are in the COOKING category

```
SELECT COUNT(*)
FROM books
WHERE category = 'COOKING';
```

2. Display how many books have a retail price over $30.00.

```
SELECT COUNT(*)
FROM books
WHERE retail > 30;
```


3. Display the most recent publication date of the books.

```
SELECT MAX(pubdate)
FROM books;
```


4. Display the total margin (retail – cost) for all orders by customer 1017.

```
SELECT SUM((retail-cost)*quantity)
FROM books, orders, orderitems
WHERE books.isbn = orderitems.isbn
  AND orderitems.order# = orders.order#
  AND customer# = 1017;
```

5. Display the smallest retail price for COMPUTER books.

```
SELECT MIN(retail)
FROM books
WHERE category = 'COMPUTER';
```

6. What is the average order margin ((retail – cost) * quantity) for each order.

```
SELECT AVG(SUM((retail-cost)*quantity))
 FROM books, orders, orderitems
 WHERE books.isbn = orderitems.isbn
   AND orderitems.order# = orders.order#
 GROUP BY orders.order#;
```

7. Display the how many orders each customer has placed.

```
SELECT customer#, COUNT(*)
FROM orders
GROUP BY customer#;
```


8. Display the highest retail price for the books by author Lisa White.


```
SELECT MAX(retail)
 FROM books JOIN bookauthor USING (isbn)
   JOIN author USING(authorid)
 WHERE lname = 'WHITE'
   AND fname = 'LISA';
```

Part G – Sub-Queries

1. Display the title and retail price for all books whose retail price is smaller than the average for all books.

```
SELECT title, retail
FROM books
WHERE retail <
                (SELECT AVG(retail)
                 FROM books);
```

2. Display the order# for all orders shipped in the same state as order 1014.

```
SELECT order#
FROM orders
WHERE shipstate = (SELECT shipstate
                   FROM orders
                   WHERE order# = 1014);
```


3. Display the author name(s) having the highest sales

```
SELECT lname, fname
FROM bookauthor JOIN author USING (authorid)
WHERE isbn IN
          (SELECT isbn
           FROM orderitems
           GROUP BY isbn
           HAVING SUM(quantity) =
                      (SELECT MAX(COUNT(*))
                       FROM orderitems
                       GROUP BY isbn) );
```

Note: Could have more than one book matching the highest sales

4. Display the city and state for orders with the highest delay between ordering and shipping dates.

```
SELECT shipcity, shipstate
FROM orders
WHERE shipdate-orderdate =
            (SELECT MAX(shipdate-orderdate)
            FROM orders);
```

5. Display the customer ids that are ordering the cheapest selling books.

```
SELECT customer#
FROM customers JOIN orders USING (customer#)
    JOIN orderitems USING (order#)
    JOIN books USING (isbn)
WHERE retail =
        (SELECT MIN(retail)
         FROM books);
```


6. Display the book titles who have the same publisher as book 'THE WOK WAY TO COOK'.

```
SELECT title
FROM books
WHERE pubid =
        (SELECT pubid
         FROM books
         WHERE title = 'THE WOK WAY TO COOK');
```

Chapter 7 – SQL: Data Definition – Answers to Review Questions and Exercises

7.1 *Describe the eight base data types in SQL.*

The eight base types are: Boolean, character, bit (removed from SQL:2003), exact numeric, approximate numeric, datetime, interval, large object. See Section 7.1.2.

7.2 *Discuss the functionality and importance of the Integrity Enhancement Feature (IEF).*

Required data:	NOT NULL of CREATE/ALTER TABLE.
Domain constraint:	CHECK clause of CREATE/ALTER TABLE and CREATE DOMAIN.
Entity integrity:	PRIMARY KEY (and UNIQUE) clause of CREATE/ALTER TABLE.
Referential integrity:	FOREIGN KEY clause of CREATE/ALTER TABLE.
General constraints:	CHECK and UNIQUE clauses of CREATE/ALTER TABLE and (CREATE) ASSERTION.

See Section 7.2.

7.3 *Discuss each of the clauses of the CREATE TABLE statement.*

The clauses are (see Section 7.3.2):

- column definition;
- PRIMARY KEY
- FOREIGN KEY
- CHECK constraints

7.4 *Discuss the advantages and disadvantages of views.*

See Section 7.4.7.

7.5 *Describe how the process of view resolution works.*

Described in Section 7.4.3.

7.6 *What restrictions are necessary to ensure that a view is updatable?*

ISO standard specifies the views that must be updatable in a system that conforms to the standard. Definition given in SQL standard is that a view is updatable if and only if:

- DISTINCT is not specified; that is, duplicate rows must not be eliminated from the query results.
- Every element in the SELECT list of the defining query is a column name (rather than a constant, expression, or aggregate function) and no column appears more than once.
- The FROM clause specifies only one table; that is, the view must have a single source table for which the user has the required privileges. If the source table is itself a view, then that

view must satisfy these conditions. This, therefore, excludes any views based on a join, union (UNION), intersection (INTERSECT), or difference (EXCEPT).

- The WHERE clause does not include any nested SELECTs that reference the table in the FROM clause.
- There is no GROUP BY or HAVING clause in the defining query.

In addition, every row that is added through the view must not violate the integrity constraints of the base table (Section 7.4.5).

7.7 *What is a materialized view and what are the advantages of a maintaining a materialized view rather than using the view resolution process?*

Materialized view is a temporary table that is stored in the database to represent a view, which is maintained as the base table(s) are updated.

Advantages - may be faster than trying to perform view resolution.
- may also be useful for integrity checking and query optimisation.

See Section 7.4.8.

7.8 *Describe the difference between discretionary and mandatory access control. What type of control mechanism does SQL support.*

Discretionary – each user is given appropriate access rights (or *privileges*) on specific database objects.

Mandatory – each database object is assigned a certain *classification level* (e.g. Top Secret, Secret, Confidential, Unclassified) and each *subject* (e.g. user, application) is given a designated *clearance level* (Top Secret > Secret > Confidential > Unclassified).

SQL security mechanism is based on discretionary access control.

7.9 *Discuss how the Access Control mechanism of SQL works.*

Each user has an **authorization identifier** (allocated by DBA).
Each object has an **owner**. Initially, only owner has access to an object but the owner can pass privileges to carry out certain actions on to other users via the GRANT statement and take away given privileges using REVOKE.

Answer the following questions using the relational schema from the Exercises at the end of Chapter 4.

7.10 *Create the Hotel table using the integrity enhancement features of SQL.*

```
CREATE DOMAIN HotelNumber AS CHAR(4);

CREATE TABLE Hotel(
        hotelNo         HotelNumber             NOT NULL,
```

```
    hotelName       VARCHAR(20)         NOT NULL,
    city            VARCHAR(50)         NOT NULL,
    PRIMARY KEY (hotelNo));
```

7.11 *Now create the Room, Booking, and Guest tables using the integrity enhancement features of SQL with the following constraints:*

(a) Type must be one of Single, Double, or Family.
(b) Price must be between £10 and £100.
(c) roomNo must be between 1 and 100.
(d) dateFrom and dateTo must be greater than today's date.
(e) The same room cannot be double booked.
(f) The same guest cannot have overlapping bookings.

```
CREATE DOMAIN RoomType AS CHAR(1)
    CHECK(VALUE IN ('S', 'F', 'D'));
CREATE DOMAIN HotelNumbers AS HotelNumber
    CHECK(VALUE IN (SELECT hotelNo FROM Hotel));
CREATE DOMAIN RoomPrice AS DECIMAL(5, 2)
    CHECK(VALUE BETWEEN 10 AND 100);
CREATE DOMAIN RoomNumber AS VARCHAR(4)
    CHECK(VALUE BETWEEN '1' AND '100');

CREATE TABLE Room(
    roomNo          RoomNumber          NOT NULL,
    hotelNo         HotelNumbers        NOT NULL,
    type            RoomType            NOT NULL DEFAULT 'S'
    price           RoomPrice           NOT NULL,
    PRIMARY KEY (roomNo, hotelNo),
    FOREIGN KEY (hotelNo) REFERENCES Hotel
        ON DELETE CASCADE ON UPDATE CASCADE);

CREATE DOMAIN GuestNumber AS CHAR(4);

CREATE TABLE Guest(
    guestNo             GuestNumber         NOT NULL,
    guestName       VARCHAR(20)         NOT NULL,
    guestAddress    VARCHAR(50)         NOT NULL);

CREATE DOMAIN GuestNumbers AS GuestNumber
    CHECK(VALUE IN (SELECT guestNo FROM Guest));
CREATE DOMAIN BookingDate AS DATETIME
    CHECK(VALUE > CURRENT_DATE);

CREATE TABLE Booking(
    hotelNo         HotelNumbers        NOT NULL,
```

```
        guestNo         GuestNumbers        NOT NULL,
        dateFrom        BookingDate         NOT NULL,
        dateTo          BookingDate         NULL,
        roomNo          RoomNumber          NOT NULL,
        PRIMARY KEY (hotelNo, guestNo, dateFrom),
        FOREIGN KEY (hotelNo) REFERENCES Hotel
                ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (guestNo) REFERENCES Guest
                ON DELETE NO ACTION ON UPDATE CASCADE,
        FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
                ON DELETE NO ACTION ON UPDATE CASCADE,
        CONSTRAINT RoomBooked
        CHECK (NOT EXISTS ( SELECT *
                            FROM Booking b
                            WHERE b.dateTo > Booking.dateFrom AND
                            b.dateFrom < Booking.dateTo AND
                            b.roomNo = Booking.roomNo AND
                            b.hotelNo = Booking.hotelNo)),
        CONSTRAINT GuestBooked
        CHECK (NOT EXISTS ( SELECT *
                            FROM Booking b
                            WHERE b.dateTo > Booking.dateFrom AND
                            b.dateFrom < Booking.dateTo AND
                            b.guestNo = Booking.guestNo)));
```


7.12 *Create a separate table with the same structure as the Booking table to hold archive records. Using the INSERT statement, copy the records from the Booking table to the archive table relating to bookings before 1st January 2013. Delete all bookings before 1st January 2013 from the Booking table.*

```
CREATE TABLE BookingOld(    hotelNo         CHAR(4)         NOT NULL,
                            guestNo         CHAR(4)         NOT NULL,
                            dateFrom        DATETIME        NOT NULL,
                            dateTo          DATETIME        NULL,
                            roomNo          VARCHAR(4)      NOT NULL);

INSERT INTO BookingOld
        (SELECT * FROM Booking
        WHERE dateTo < DATE'2013-01-01');
DELETE FROM Booking
WHERE dateTo < DATE'2013-01-01';
```

7.13 *Create a view containing the hotel name and the names of the guests staying at the hotel.*

```
CREATE VIEW HotelData(hotelName, guestName)
AS      SELECT h.hotelName, g.guestName
```

```
        FROM Hotel h, Guest g, Booking b
        WHERE h.hotelNo = b.hotelNo AND g.guestNo = b.guestNo AND
                b.dateFrom <= CURRENT_DATE AND
                b.dateTo >= CURRENT_DATE;
```

7.14 *Create a view containing the account for each guest at the Grosvenor Hotel.*

```
CREATE VIEW BookingOutToday
AS      SELECT g.guestNo,g.guestName,g.guestAddress,r.price*(b.dateTo-b.dateFrom)
        FROM Guest g, Booking b, Hotel h, Room r
        WHERE g.guestNo = b.guestNo AND r.roomNo = b.roomNo AND
                b.hotelNo = h.hotelNo AND h.hotelName = 'Grosvenor Hotel' AND
                b.dateTo = CURRENT_DATE;
```

7.15 *Give the users Manager and Deputy full access to these views, with the privilege to pass the access on to other users.*

```
GRANT ALL PRIVILEGES ON HotelData
TO Manager, Director WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON BookingOutToday
TO Manager, Director WITH GRANT OPTION;
```


7.16 *Give the user Accounts SELECT access to these views. Now revoke the access from this user.*

```
GRANT SELECT ON HotelData TO Accounts;
GRANT SELECT ON BookingOutToday TO Accounts;

REVOKE SELECT ON HotelData FROM Accounts;
REVOKE SELECT ON BookingOutToday FROM Accounts;
```


7.17 *Consider the following view defined on the Hotel schema:*

```
CREATE VIEW HotelBookingCount (hotelNo, bookingCount)
AS      SELECT h.hotelNo, COUNT(*)
        FROM Hotel h, Room r, Booking b
        WHERE h.hotelNo = r.hotelNo AND r.roomNo = b.roomNo
        GROUP BY h.hotelNo;
```

For each of the following queries, state whether the query is valid and for the valid ones should how each of the queries would be mapped onto a query on the underling base tables.

(a)
```
        SELECT *
        FROM HotelBookingCount;

        SELECT h.hotelNo, COUNT(*)
```

```
FROM Hotel h, Room r, Booking b
WHERE h.hotelNo = r.hotelNo AND r.roomNo = b.roomNo
GROUP BY h.hotelNo;
```

(b)
```
SELECT hotelNo
FROM HotelBookingCount
WHERE hotelNo = 'H001';

SELECT h.hotelNo
FROM Hotel h, Room r, Booking b
WHERE h.hotelNo = r.hotelNo AND r.roomNo = b.roomNo AND
      h.hotelNo = 'H001'
GROUP BY h.hotelNo;
```

(c)
```
SELECT MIN(bookingCount)
FROM HotelBookingCount;
```

Invalid – bookingCount is based on an aggregate function, so cannot be used within another aggregate function.

(d)
```
SELECT COUNT(*)
FROM HotelBookingCount;
```


Invalid for reason given above.

(e)
```
SELECT hotelNo
FROM HotelBookingCount
WHERE bookingCount > 1000;
```

Invalid – bookingCount is based on an aggregate function, so cannot be used in the WHERE clause.

(f)
```
SELECT hotelNo
FROM HotelBookingCount
ORDER BY bookingCount;

SELECT h.hotelNo, COUNT(*) AS bookingCount
FROM Hotel h, Room r, Booking b
WHERE h.hotelNo = r.hotelNo AND r.roomNo = b.roomNo
GROUP BY h.hotelNo
ORDER BY bookingCount;
```

COMP 3611 Module 3 Unit 2 SQL DDL Exercises – SOLUTIONS

The SQL DDL exercises below use the Just Books demo database.

Part A: Tables

1. Define an **employees** table with the following columns: **emp_id** is 5 digits, **lastname** is up to 15 letters, **firstname** is up to 10 letters, **job_class** up to 4 letters, **comm** is one letter.

```
CREATE TABLE employees
(emp_id NUMBER(5),
lastname VARCHAR2(15),
firstname VARCHAR2(10),
job_class VARCHAR2(4),
comm CHAR(1));
```


2. Change the above table to include **empdate** (the date they started employment with a default of today's date) and **enddate** (the date they ended employment).

```
ALTER TABLE employees
ADD (empdate DATE DEFAULT SYSDATE,
 enddate DATE);
```

3. Change the **job_class** column to be at most 2 letters.

```
ALTER TABLE employees
MODIFY job_class VARCHAR2(2);
```

4. Remove the **enddate** column.

```
ALTER TABLE employees
DROP column enddate;
```

5. Create a temporary table called **book_pricing** that consists of the data in the **isbn**, **cost**, **retail**, and **category** columns from the books table. Display the results.

```
CREATE TABLE book_pricing (id, cost, retail, category)
 AS (SELECT isbn, cost, retail, category
 FROM books);
SELECT *
 FROM book_pricing;
```


6. Purge all the date from the **book_pricing** table (i.e. do not use DELETE)


```
TRUNCATE TABLE book_pricing;
```


7. Drop the **book_pricing** table and verify it no longer exists.

```
DROP TABLE book_pricing;
DESCRIBE book_pricing;
```

Part B: Keys and Other Constraints

1. Modify the **employees** table script for A.1 above to include the following:
   - **emp_id** is the primary key
   - **lastname** and **firstname** must not be null
   - **comm** can only be Y or N and the default value is Y

```
DROP TABLE employees;
CREATE TABLE employees
 (emp_id NUMBER(5),
 lastname VARCHAR2(15) NOT NULL,
 firstname VARCHAR2(10) NOT NULL,
 job_class VARCHAR2(4),
 comm CHAR(1) DEFAULT 'Y',
 CONSTRAINT employees_pk PRIMARY KEY (emp_id,
 CONSTRAINT employees_comm_ck CHECK (comm IN('Y','N')));
```


2. Modify the **employees** table to include **base_salary** with the condition that it must be a positive number.

```
ALTER TABLE employees
 ADD base_salary NUMBER(7,2)
 CONSTRAINT employees_basesalary_ck CHECK (base_salary > 0);
```

3. Create a book_stores table with columns
   - store_id: 3 digits, primary key
   - store_name: 30 letters, unique
   - mgr_id: 5 digits

```
CREATE TABLE book_stores
 (store_id NUMBER(3),
 name VARCHAR2(30) NOT NULL,
 mgr_id NUMBER(5),
```

```
CONSTRAINT book_stores_storeid_pk PRIMARY KEY (store_id),
CONSTRAINT book_stores_name_uk UNIQUE (name) );
```

4. Add the constraint to the **book_stores** table that **mgr_id** must be a valid **emp_id** from the **employees** table.

```
ALTER TABLE book_stores
 ADD CONSTRAINT book_stores_mgr_id_fk FOREIGN KEY (mgr_id)
  REFERENCES employees (emp_id);
```

Part C: Sequences and Indexes

1. Define a sequence for the **customers** table starting with the next available id number.

```
CREATE SEQUENCE cust_seq
 START WITH 1021
 NOMAXVALUE
 NOMINVALUE
 NOCACHE
 NOCYCLE;
```


2. Insert a new row in **customers** for yourself that uses this sequence (only the last and first names are required).

```
INSERT INTO customers (customer#, lastname, firstname)
 VALUES (cust_seq.NEXTVAL, 'STUDENT', 'IMA');
```


3. Define a temporary sequence where the starting value, maximum, and increment are all 5, minimum is 0, with no cache and no cycle.

```
CREATE SEQUENCE temp_seq
 INCREMENT BY 5
 START WITH 5
 MAXVALUE 5
 MINVALUE 0
 NOCYCLE;
```

4. Try to display the next several values for this temporary sequence. What happens? Why?

```
SELECT temp_seq.NEXTVAL  FROM DUAL;        gives 5
SELECT temp_seq.NEXTVAL  FROM DUAL;        error
```

Error caused by the sequence running out of values to issue, as the maximum value of 5 was reached and the CYCLE option is set to NOCYCLE.

5. Change this temporary sequence so the maximum value is 1000. Repeat the above query.

```
ALTER SEQUENCE temp_seq
 MAXVALUE 1000;
```

6. Define the following indexes on the **customers** table:
   - a bitmap index on state
   - a composite index on the last and first names

   Open the indexes tab to see the structure of all the indexes.

```
CREATE BITMAP INDEX customers_state_idx
 ON customers (state);

CREATE INDEX customers_name_idx
 ON customers (lastname, firstname);
```


7. Remove these two new indexes. View the results in the indexes tab (remember to hit Refresh).

```
DROP INDEX customers_state_idx;

DROP INDEX customers_name_idx;
```

Part D: Views

1. Define a view consisting of the **pubid**, **contact** and **phone** from the **publisher** table. This view is not to allow updates.

```
CREATE VIEW contacts
 AS SELECT pubid, contact, phone
  FROM publisher
 WITH READ ONLY;
SELECT * FROM contacts;
```

2. Define a view called **book_reorder** consisting of the **isbn** and **title** from the **books** table and the view from above.


```
CREATE VIEW book_reorder
 AS SELECT isbn, title, pubid, contact, phone
  FROM books JOIN contacts USING (pubid);
SELECT * FROM book_reorder;
```


3. Define a view that can be used to summarize how many books each author has in the inventory.

```
CREATE OR REPLACE VIEW author_books
 AS SELECT authorid, lname, fname, isbn
   FROM author a JOIN bookauthor USING (authorid);
SELECT authorid, lname, fname, count(*)
 FROM author_books
   GROUP BY authorid, lname, fname;
```

Chapter 8 – Advanced SQL – Answers to Review Questions and Exercises

8.1 *Explain the term "impedance mismatch." Briefly describe how SQL now overcomes the impedance mismatch.*

The term impedence mismatch refers to the mixing of different programming paradigms between SQL, a declarative language, and procedural and object-orieinted programming languages. SQL overcomes the impedance mismatch through extensions such as the SQL/PSM (Persistend Stored Modules) extension and Oracle's PL/SQL (Procedural Language/SQL).

8.2 *Describe the general structure of a PL/SQL block.*

A PL/SQL block has up to three parts:

- An optional declaration part, in which variables, constants, cursors, and exceptions are defined and possibly initialized;
- A mandatory executable part, in which the variables are manipulated;
- An optional exception part, to handle any exceptions raised during execution.

8.3 *Describe the control statements in PL/SQL. Give examples to illustrate your answers.*

PL/SQL supports the usual conditional, iterative, and sequential flow-of-control mechanisms including:

Conditional IF statement


```
IF (position = 'Manager') THEN
salary := salary*1.05;
ELSE
salary := salary*1.03;
END IF;
```

Conditional CASE statement

```
CASE lowercase(x)
WHEN 'a' THEN x := 1;
WHEN 'b' THEN x := 2;
y := 0;
WHEN 'default' THEN x := 3;
 END CASE;
```

Iteration statement (LOOP)

```
x:=1;
myLoop:
LOOP
x := x+1;
IF (x > 3) THEN
EXIT myLoop; --- exit loop immediately
END LOOP myLoop;
```

```
--- control resumes here
y := 2;
```

Iteration statement (WHILE and REPEAT)

```
WHILE (condition) LOOP
<SQL statement list>
END LOOP [labelName];
```

Iteration statement (FOR)

```
DECLARE
numberOfStaff NUMBER;
SELECT COUNT(*) INTO numberOfStaff FROM PropertyForRent
WHERE staffNo = 'SG14';
myLoop1:
FOR iStaff IN 1 .. numberOfStaff LOOP
.....
END LOOP myLoop1;
```

8.4 Describe how the PL/SQL statements differ from the SQL standard. Give examples to illustrate your answers.

PL/SQL (Procedural Language/SQL) is Oracle's procedural extension to SQL.

There are two versions of PL/SQL: one is part of the Oracle server, and the other is a separate engine embedded in a number of Oracle tools. They are very similar to each other and have the same programming constructs, syntax, and logic mechanisms, although PL/SQL for Oracle tools has some extensions to suit the requirementsof the particular tool (for example, PL/SQL has extensions for Oracle Forms).

PL/SQL has concepts similar to modern programming languages, such as variable and constant declarations, control structures, exception handling, and modularization. PL/SQL is a block-structured language: blocks can be entirely separate or nested within one another. The basic units that constitute a PL/SQL program are procedures, functions, and anonymous *(unnamed)* blocks.

8.5 What are SQL cursors? Give an example of the use of an SQL cursor.

PL/SQL uses **cursors** to allow the rows of a query result to be accessed one at a time. In effect, the cursor acts as a pointer to a particular row of the query result. The cursor can be advanced by 1 to access the next row. A cursor must be *declared* and *opened* before it can be used, and it must be *closed* to deactivate it after it is no longer required. Once the cursor has been opened, the rows of the query result can be retrieved one at a time using a FETCH statement, as opposed to a SELECT statement.

An example is shown in Figure 8.3.

8.6 What are database triggers and what could they be used for?

A trigger defines an action that the database should take when some event occurs in the application. A trigger may be used to enforce some referential integrity constraints, to enforce complex constraints, or to audit changes to data.

8.7 Discuss the differences between BEFORE, AFTER, and INSTEAD OF triggers. Give examples to illustrate your answers.

The BEFORE keyword indicates that the trigger should be executed before an insert is applied. It could be used to prevent a member of staff from managing more than 100 properties at the same time.

The AFTER keyword indicates that the trigger should be executed after the database table is updated. It could be used to create an audit record.

SQL also supports INSTEAD OF triggers, which provide a transparent way of modifying views that cannot be modified directly through SQL DML statements (INSERT, UPDATE, and DELETE). These triggers are called INSTEAD OF triggers because, unlike other types of trigger, the trigger is fired *instead of* executing the original SQL statement.

8.8 Discuss the differences between row-level and statement-level triggers. Give examples to illustrate your answers.

There are two types of trigger: *row-level* triggers (FOR EACH ROW) that execute for each row of the table that is affected by the triggering event, and *statement-level* triggers (FOR EACH STATEMENT) that execute only once even if multiple rows are affected by the triggering event. An example of a row-level trigger is shown in Example 8.2. An example of a statement-level trigger to set a new sequence number for an update is seen in the middle of Figure 8.5(b).

8.9 Discuss the advantages and disadvantages of database triggers.

Advantages of triggers include:

- *Elimination of redundant code:* Instead of placing a copy of the functionality of the trigger in every client application that requires it, the trigger is stored only once in the database.
- *Simplifying modifications:* Changing a trigger requires changing it in one place only; all the applications automatically use the updated trigger. Thus, they are only coded once, tested once, and then centrally enforced for all the applications accessing the database. The triggers are usually controlled, or at least audited, by a skilled DBA. The result is that the triggers can be implemented efficiently.
- *Increased security:* Storing the triggers in the database gives them all the benefits of security provided automatically by the DBMS.
- *Improved integrity:* Triggers can be extremely useful for implementing some types of integrity constraints, as we have demonstrated earlier. Storing such triggers in the

database means that integrity constraints can be enforced consistently by the DBMS across all applications.

- *Improved processing power:* Triggers add processing power to the DBMS and to the database as a whole.
- *Good fit with the client-server architecture:* The central activation and processing of triggers fits the client-server architecture well (see Chapter 3). A single request from a client can result in the automatic performing of a whole sequence of checks and subsequent operations by the database server. In this way, performance is potentially improved as data and operations are not transferred across the network between the client and the server.

Triggers also have disadvantages, which include:

- *Performance overhead:* The management and execution of triggers have a performance overhead that have to be balanced against the advantages cited previously.
- *Cascading effects:* The action of one trigger can cause another trigger to be fired, and so on, in a cascading manner. Not only can this cause a significant change to the database, but it can also be hard to foresee this effect when designing the trigger.
- *Cannot be scheduled:* Triggers cannot be scheduled: they occur when the event that they are based on happens.
- *Less portable:* Although now covered by the SQL standard, most DBMSs implement their own dialect for triggers, which affects portability.

Exercises

8.10 Create a stored procedure for each of the queries specified in Exercises 6.7–6.11.

6.7 List full details of all hotels.

```
DECLARE
        vHotelNo        Hotel.hotelNo%TYPE;
        vHotelName      Hotel.hotelName%TYPE;
        vCity           Hote.city%TYPE;

        CURSOR hotelCursor IS
        SELECT * FROM Hotel;

BEGIN
        OPEN hotelCursor;
        LOOP

                FETCH hotelCursor
                        INTO vHotelNo, vHotelName, vCity;
                EXIT WHEN hotelCursor%NOTFOUND;

                dbms_output.put_line('Hotel Number: ' || vHotelNo);
                dbms_output.put_line('Hotel Name: ' || vHotelName);
```

```
            dbms_output.put_line('City: ' || vCity);
        END LOOP

        IF hotelCursor%ISOPEN THEN CLOSE hotelCursor END IF;

EXCEPTION
        WHEN OTHERS THEN
            dbms_output.put_line('Error detected');
            IF hotelCursor%ISOPEN THEN CLOSE hotelCursor; END IF;
END;
```

6.8 List full details of all hotels in London.

```
DECLARE
        vHotelNo        Hotel.hotelNo%TYPE;
        vHotelName      Hotel.hotelName%TYPE;
        vCity           Hote.city%TYPE;

        CURSOR hotelCursor IS
        SELECT * FROM Hotel
        WHERE city = 'London';

BEGIN
        OPEN hotelCursor;
        LOOP
            FETCH hotelCursor
                INTO vHotelNo, vHotelName, vCity;
            EXIT WHEN hotelCursor%NOTFOUND;

            dbms_output.put_line('Hotel Number: ' || vHotelNo);
            dbms_output.put_line('Hotel Name: ' || vHotelName);
            dbms_output.put_line('City: ' || vCity);
        END LOOP

        IF hotelCursor%ISOPEN THEN CLOSE hotelCursor END IF;

EXCEPTION
        WHEN OTHERS THEN
            dbms_output.put_line('Error detected');
            IF hotelCursor%ISOPEN THEN CLOSE hotelCursor; END IF;
END;
```

6.9 List the names and addresses of all guests in London, alphabetically ordered by name.

```
DECLARE
        vGuestName      Guest.guestName%TYPE;
```

```
        vGuestAddress     Guest.guestAddress%TYPE;

        CURSOR guestCursor IS
        SELECT guestName, guestAddress
        FROM Guest WHERE address LIKE '%London%'
        ORDER BY guestName;

BEGIN
        OPEN guestCursor;
        LOOP
                FETCH guestCursor
                        INTO vGuestName, vGuestAddress;
                EXIT WHEN guestCursor %NOTFOUND;

                dbms_output.put_line('Guest Name: ' || vGuestName);
                dbms_output.put_line('Guest Address: ' || vGuestAddress);
        END LOOP

        IF guestCursor %ISOPEN THEN CLOSE guestCursor END IF;
EXCEPTION
        WHEN OTHERS THEN
                dbms_output.put_line('Error detected');
                IF guestCursor %ISOPEN THEN CLOSE guestCursor; END IF;
END;
```


6.10 List all double or family rooms with a price below £40.00 per night, in ascending order of price.

```
DECLARE
        vRoomNo         Room.roomNo%TYPE;
        vHotelNo        Room.hotelNo%TYPE;
        vType           Room.roomType%TYPE;
        vPrice          Room.price%TYPE;

        CURSOR roomCursor IS
        SELECT * FROM Room
        WHERE price < 40
        AND type IN ('D', 'F')
        ORDER BY price;

BEGIN

        OPEN roomCursor;
        LOOP
                FETCH roomCursor
```

```
                INTO vRoomNo, vHotelNo, vType, vPrice;
            EXIT WHEN roomCursor %NOTFOUND;

            dbms_output.put_line('Room Number: ' || vRoomNo);
            dbms_output.put_line('Hotel Number: ' || vHotelNo);
            dbms_output.put_line('Type: ' || vType);
            dbms_output.put_line('Price: ' || vPrice);
        END LOOP

        IF roomCursor %ISOPEN THEN CLOSE roomCursor END IF;

EXCEPTION
        WHEN OTHERS THEN
            dbms_output.put_line('Error detected');
            IF roomCursor %ISOPEN THEN CLOSE roomCursor; END IF;
END;
```

6.11 List the bookings for which no dateTo has been specified.


```
DECLARE
        vHotelNo        Booking.hotelNo%TYPE;
        vGuestNo        Booking.guestNo%TYPE;
        vDateFrom       Booking.dateFrom%TYPE;
        vRoomNo         Booking.roomNo%TYPE;

        CURSOR bookingCursor IS
        SELECT * FROM Booking
        WHERE dateTo IS NULL;

BEGIN
        OPEN bookingCursor;
        LOOP
            FETCH bookingCursor
                INTO vHotelNo, vGuestNo, vDateFrom, vRoomNo;
            EXIT WHEN bookingCursor %NOTFOUND;

            dbms_output.put_line('Hotel Number: ' || vHotelNo);
            dbms_output.put_line('Guest Number: ' || vGuestNo);
            dbms_output.put_line('Date From: ' || vDateFrom);
            dbms_output.put_line('Room Number: ' || vRoomNo);
        END LOOP

        IF bookingCursor %ISOPEN THEN CLOSE bookingCursor END IF;

EXCEPTION
        WHEN OTHERS THEN
```

```
                dbms_output.put_line('Error detected');
                IF bookingCursor %ISOPEN THEN CLOSE bookingCursor; END IF;
END;
```

8.11 Create a database trigger for the following situations:

(a) The price of all double rooms must be greater than £100.

```
CREATE TRIGGER DouleRoomPrice
BEFORE INSERT ON Room
FOR EACH ROW
WHEN (new.type = 'D' AND new.price < 100)
BEGIN
        raise_application_error(-20000, 'Price for double room must be over 100);
END;
```

(b) The price of double rooms must be greater than the price of the highest single room.

```
CREATE TRIGGER RoomPrice
BEFORE INSERT ON Room
FOR EACH ROW
WHEN (new.type = 'D')
DECLARE vMaxSingleRoomPrice  NUMBER;
BEGIN
        SELECT MAX(price) INTO vMaxSingleRoomPrice
         FROM Room
        WHERE type = 'S';
        IF (new.price < vMaxSingleRoomPrice)
                raise_application_error(-20000, 'Double room price must be higher than
higest single room price '||vMaxSingleRoomPrice);
        END IF;
END;
```

(c) A booking cannot be for a hotel room that is already booked for any of the specified dates.

```
CREATE TRIGGER BookingDate
BEFORE INSERT ON Booking
FOR EACH ROW
DECLARE vBookingCount          NUMBER;

BEGIN
        SELECT COUNT(*) INTO vBookingCount
        FROM Booking
        WHERE hotelNo = new.hotelNo
        AND dateFrom <= new.dateFrom
        AND dateTo >= new.dateTo;
```

```
        IF (vBookingCount > 0)
                raise_application_error(-20000, 'Room ' || new.roomNo || ' is already
booked during these dates');
        END IF;
END;
```

(d) A guest cannot make two bookings with overlapping dates.

```
CREATE TRIGGER BookingGuest
BEFORE INSERT ON Booking
FOR EACH ROW
DECLARE vBookingCount          NUMBER;

BEGIN
        SELECT COUNT(*) INTO vBookingCount
        FROM Booking
        WHERE guestNo = new.guestNo
        AND dateFrom <= new.dateFrom
        AND dateTo >= new.dateTo;
        IF (vBookingCount > 0)
                raise_application_error(-20000, 'Guest ' || new.guestNo || ' is already
booked during these dates');
        END IF;
END;
```


(e) Maintain an audit table with the names and addresses of all guests who make bookings for hotels in London (do not store duplicate guest details).

```
CREATE TRIGGER BookingAfterInsert
AFTER INSERT ON Booking
FOR EACH ROW
DECLARE
        vGuestName      Guest.guestName%TYPE;
        vGuestAddress   Guest.guestAddress%TYPE;
BEGIN
        SELECT g.guestName INTO vGuestName, g.guestAddress into vGuestAddress
        FROM Guest g, Hotel h
        WHERE g.guestNo = new.bookingNo
        AND h.hotelNo = new.hotelNo
        AND h.city = 'London';
        IF (vGuestName IS NOT NULL AND vGuestAddress IS NOT NULL)

                INSERT INTO GuestAudit
                VALUES (vGuestName, vGuestAddress);
        END iF;
```

```
END;
```

8.12 *Create an INSTEAD OF database trigger that will allow data to be inserted into the following view:*

```
CREATE VIEW LondonHotelRoom AS
SELECT h.hotelNo, hotelName, city, roomNo, type, price
FROM Hotel h, Room r
WHERE h.hotelNo _ r.hotelNo AND city _ 'London'

CREATE TRIGGER UpdateLondonHotelRoom
INSTEAD OF INSERT ON LondonHotelRoom
FOR EACH ROW

BEGIN
        INSERT INTO Hotel (hotelNo, hotelName, city)
        VALUES (new.hotelNo, new.hotelName, 'London');
        INSERT INTO Room (roomNo, hotelNo, type, price)
        VALUES (new.roomNo, new.hotelNo, new.type, new.price);
END;
```

COMP 3611 Module 3 Unit 3 PL/SQL Exercises - SOLUTIONS

The PL/SQL exercises below use the Just Books demo database.

1. Write an anonymous block that examines the system date and responds to the console window whether it is a **Weekend** (i.e. Saturday or Sunday) or a **Weekday**.

```
SET SERVEROUTPUT ON;
DECLARE
  theday CHAR(3);
BEGIN
  theday := TO_CHAR(SYSDATE, 'DY');
  IF theday = 'SAT' OR theday = 'SUN' THEN
    DBMS_OUTPUT.PUT_LINE('Weekend');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Weekday');
  END IF;
END;
```


2. Write an anonymous block that uses a substitution variable to retrieve and display a publisher's information based on the pubid. Run using id 2 then id 7.

```
SET SERVEROUTPUT ON;
DECLARE
  v_pubid number(2) := &pubid;
  v_name VARCHAR2(23);
  v_contact VARCHAR2(15);
```

```
  v_phone VARCHAR2(12);
BEGIN
  SELECT name, contact, phone
    INTO v_name, v_contact, v_phone
    FROM publisher WHERE pubid = v_pubid;
    DBMS_OUTPUT.PUT_LINE('Name:    ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Contact: ' || v_contact);
    DBMS_OUTPUT.PUT_LINE('Phone:   ' || v_phone);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No such publisher');
END;
```


3. Develop and test an AGE function to calculate age in years for a specified date. Test using your date of birth.

```
CREATE OR REPLACE FUNCTION AGE(p_date DATE)
  RETURN NUMBER
IS
  v_today DATE := SYSDATE;
  v_age NUMBER;
BEGIN
  v_age := TRUNC((v_today - p_date)/365.25);
  RETURN v_age;
END;
/

SELECT AGE(TO_DATE('1983-04-05','YYYY-MM-DD')) FROM DUAL;
```

4. Create and test a trigger to insert the next customer using the value from the CUST_SEQ sequence.

```
CREATE OR REPLACE TRIGGER customer_bi
BEFORE INSERT ON customers
FOR EACH ROW
BEGIN
  :NEW.customer# := cust_seq.NEXTVAL;
END;
/

INSERT INTO customers (lastname, firstname) VALUES ('Testing', '123');
SELECT * FROM customers;
ROLLBACK;
```

Chapter 10 – Database Systems Development Lifecycle – Answers to Review Questions

10.1 *Describe the major components of an information system.*

Database, database software, application software, computer hardware including storage media, and people using and developing the system.

10.2 *Discuss the relationship between the information systems lifecycle and the database system development lifecycle.*

See Sections 10.1 and 10.2.

10.3 *Describe the main purpose(s) and activities associated with each stage of the database system development lifecycle.*

See Table 10.1 in Section 10.2.

10.4 *Discuss what a user view represents in the context of a database system.*

User view: defines what is required of a database system from the perspective of a particular job role (such as Manager or Supervisor) or enterprise application area (such as marketing, personnel, or stock control) (See Section 10.4.1).

10.5 *Discuss the main approaches for managing the design of a database system that has multiple user views.*

Three main approaches: centralized, view integration, and combination of both approaches (see Section 10.5).

10.6 *Compare and contrast the three phases of database design.*

Three phases are conceptual database design, logical database design, and physical database design:
Conceptual database design constructs a model of the information used in an enterprise, independent of all physical considerations.
Logical database design is based on a specific data model, but independent of all other physical considerations.
Physical database design constructs a description of the implementation of the database on secondary storage (see Section 10.6.3).

10.7 *What are the main purposes of data modeling and identify the criteria for an optimal data model.*

The two main purposes of data modeling are to assist in the understanding of the meaning (semantics) of the data and to facilitate communication about the information requirements (see Section 10.6.2).

10.8 *Identify the stage(s) where it is appropriate to select a DBMS and describe an approach to selecting the 'best' DBMS.*

Physical database design is tailored to a specific DBMS; therefore it is essential the DBMS is determined before the physical design phase can begin. Physical database design is described in Section 10.6.3.

10.9 *Application design involves transaction design and user interface design. Describe the purpose and main activities associated with each.*

See Section 10.8.

10.10 *Discuss why testing cannot show the absence of faults, only that software faults are present.*

See Section 10.12.

10.11 *Describe the main advantages of using the prototyping approach when building a database system.*

Should be relatively inexpensive to develop and quick to build (see Section 10.9).

Chapter 11 – Database Analysis – Answers to Review Questions

11.1 *Briefly describe what the process of fact-finding attempts to achieve for a database developer.*

Attempts to uncover facts about the business and the users of the database system including the vocabulary, problems, opportunities, constraints, requirements, and priorities.

11.2 *Describe how fact-finding is used throughout the stages of the database system development lifecycle.*

See Table 11.1 in Section 11.2.

11.3 *For each stage of the database system development lifecycle identify examples of the facts captured and the documentation produced.*

See Table 11.1 in Section 11.2.

11.4 *A database developer normally uses several fact-finding techniques during a single database project. The five most commonly used techniques are examining documentation, interviewing, observing the business in operation, conducting research, and using questionnaires. Describe each fact-finding technique and identify the advantages and disadvantages of each.*

See Section 11.3.

11.5 *Describe the purpose of defining a mission statement and mission objectives for a database system.*

Mission statement defines the major aims of the database system; the mission objectives identify the particular tasks that the database must support (see Section 11.4.2).

11.6 *What is the purpose of identifying the systems boundary for a database system?*

Ensures that all appropriate areas of the organization are supported by the database system (see Section 11.4.3).

11.7 *How does the contents of a users' requirements specification differ from a systems specification?*

Systems specification describes the any features to be included in the new database system such as networking and shared access requirements, performance requirements, and the levels of security required. On the other hand, the users' requirements specification describes in detail the data to be held in the database and how the data is to be used (see Section 10.4.4).

11.8 *Describe one method to deciding whether to use either the centralized or view integration approach, or a combination of both when developing a database system for multiple user views.*

One way is to examine the overlap in the data used between the various user views (see, for

example, Table 11.7).

Chapter 12 – Entity-Relationship Modeling – Answers to Review Questions and Exercises

12.1 *Describe what entity types represent in an ER model and provide examples of entities with a physical or conceptual existence.*

An entity type represents a group of 'objects' in the real world with the same properties (see Section 12.1). Examples of entities with physical and conceptual existence are shown in Figure 12.2.

12.2 *Describe what relationship types represent in an ER model and provide examples of unary, binary, ternary, and quaternary relationships.*

A relationship type is a set of associations between one or more participating entity types (see Section 12.2). Examples:

Unary: Staff *Supervise* Staff (also called recursive relationship)
Binary: Branch *Has* Staff
Ternary: Staff *Registers* Client at Branch
Quaternary: Solicitor *Arranges* Bid with a Buyer supported by a Financial Institution.

12.3 *Describe what attributes represent in an ER model and provide examples of simple, composite, single-value, multi-value, and derived attributes.*

An attribute represents a property of an entity or a relationship type (see Section 12.3). Examples:

Simple: position or salary attribute of Staff.
Composite: address attribute composed of street, city, and postcode attributes
Single-valued: branchNo attribute of Branch
Multi-valued: telNo attribute of Branch
Derived: duration attribute of Lease, calculated from rentStart and rentFinish attributes.

12.4 *Describe what the multiplicity constraint represents for a relationship type.*

Multiplicity represents the number (or range) of possible occurrences of an entity type that may relate to a single occurrence of an associated entity type through a particular relationship (see Section 12.6).

12.5 *What are enterprise constraints and how does multiplicity model these constraints?*

Enterprise constraints are rules that the data in the database must conform to as specified by users or database administrators of a database (see Section 4.3.4). Multiplicity constrains the way that entities are related – it is a representation of the policies (or business rules) established by the user or enterprise.

12.6 *How does multiplicity represent both the cardinality and the participation constraints on a relationship type?*

Multiplicity actually consists of two separate constraints (see Section 12.6.5):

Cardinality – which describes the maximum number of possible relationship occurrences for an entity participating in a given relationship type.
Participation – which determines whether all or only some entity occurrences participate in a relationship.

12.7 *Provide an example of a relationship type with attributes.*

The relationship Newspaper *Advertises* PropertyForRent consists of two attributes: dateAdvert (representing the date the advert took place) and cost (representing the cost of the advert).

12.8 *Describe how strong and weak entity types differ and provide an example of each.*

A strong entity type is an entity type that is not existence-dependent on some other entity type (see Section 12.4). Examples of strong entity types are Branch, Staff, and PropertyForRent.

A weak entity type is an entity type that is existence-dependent on some other entity type. An example of a weak entity type is Preference.

12.9 *Describe how fan and chasm traps can occur in an ER model and how they can be resolved.*

A fan trap occurs where a model represents a relationship between two entity types, but the pathway between certain entity occurrences is ambiguous. Resolve the fan trap by restructuring the original ER diagram to represent the correct association between these entities (see Section 12.7.1).

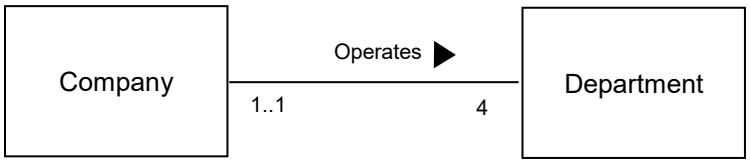
A chasm trap occurs where a model suggests the existence of a relationship between entity types, but the pathway does not exist between certain entity occurrences. A chasm trap may occur where there are one or more relationships with optional participation. Resolve the chasm trap by identifying the missing relationship (see Section 12.7.2).

Chapter 12 – Entity-Relationship Modeling – Answers to Review Questions and Exercises

Exercises

12.10 *Create an ER diagram for each of the following descriptions:*

(a) *Each company operates four departments, and each department belongs to one company.* (Note when the exact cardinality is known (in this example, 4) a value can replace the multiplicity range.

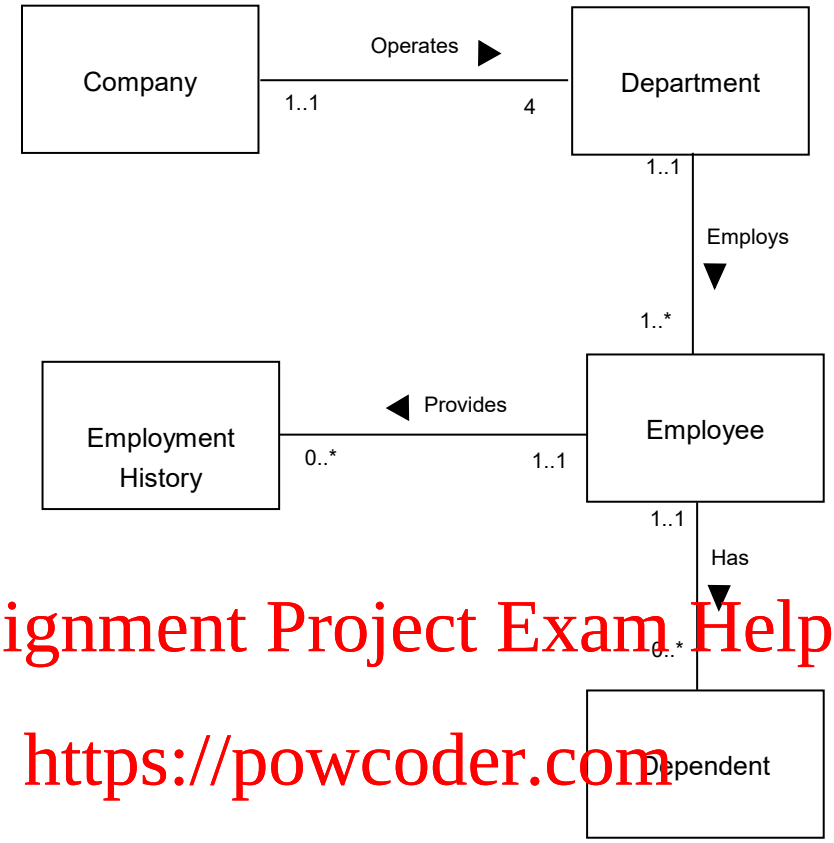


(b) *Each department in part (a) employs one or more employees, and each employee works for one department.*

(c) *Each of the employees in part (b) may or may not have one or more dependants, and each dependant belongs to one employee.*

(d) *Each employee in part (c) may or may not have an employment history.*

(e) *Represent all the ER diagrams described in (a), (b), (c), and (d) as a single ER diagram.*



12.13 *Create an ER model for each of the following descriptions:*
(a) A large organization has several parking lots, which are used by staff.
(b) Each parking lot has a unique name, location, capacity, and number of floors (where appropriate).
(c) Each parking lot has parking spaces, which are uniquely identified using a space number.
(d) Members of staff can request the sole use of a single parking space. Each member of staff has a unique number, name,telephone extension number, and vehicle license number.
(e) Represent all the ER models described in parts (a), (b), (c), and (d) as a single ER model. Provide any assumptions necessary to support your model.

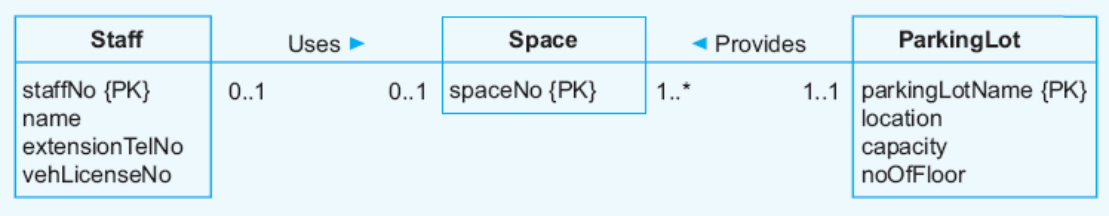


Figure 13.11

12.14 *Create an ER model to represent the data use by the library.*
The library provides books to borrowers. Each book is described by title, edition and year of publication and is uniquely identified using the ISBN. Each borrower is described by his or her name and address and is uniquely identified using a borrower number. The library provides one or more copies of each book and each copy is uniquely identified using a copy number, status indicating if the book is available for loan and the allowable loan period for a given copy. A borrower may loan one or many books and the date each book is loaned out and is returned is recorded. Loan number uniquely identifies each book loan.

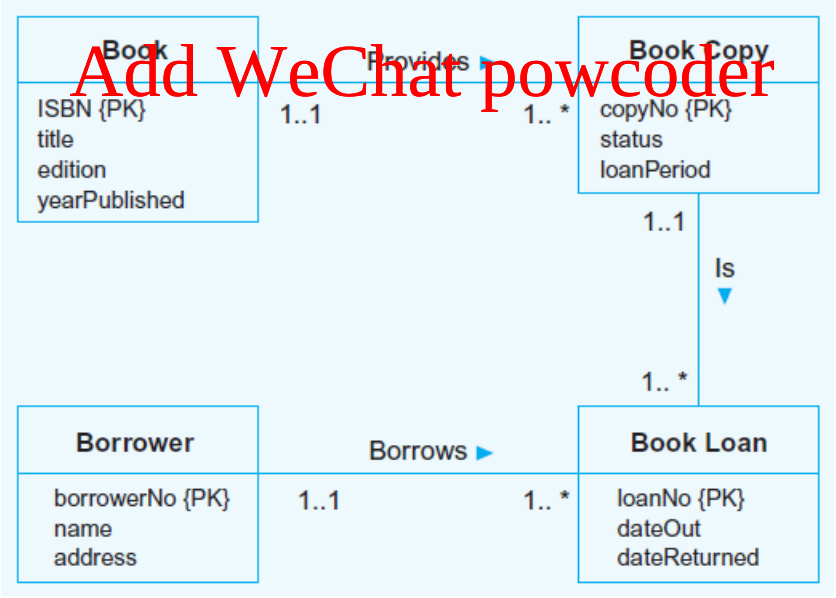


Figure 13.12

Chapter 13 – Enhanced Entity-Relationship Modeling – Answers to Review Questions and Exercises

13.1 *Describe what a superclass and a subclass represent.*

A **superclass** is an entity type that includes one or more distinct subgroupings of its occurrences, which require to be represented in a data model. A **subclass** is a distinct subgrouping of occurrences of an entity type, which require to be represented in a data model (see Section 13.1.1).

13.2 *Describe the relationship between a superclass and its subclass.*

The relationship between superclass and subclass is 1:1, called a superclass/subclass relationship (see Section 13.1.2). Each member of a subclass is also a member of the superclass.

13.3 *Describe and illustrate using an example the process of attribute inheritance.*

An entity in a subclass represents the same 'real world' object as in the superclass, and may possess subclass-specific attributes, as well as those associated with the superclass. For example, a member of the SalesPersonnel subclass *inherits* all the attributes of the Staff superclass such as staffNo, name, position, and salary together with those specifically associated with the SalesPersonnel subclass such as salesArea and carAllowance (see Section 13.1.3).

13.4 *What are the main reasons for introducing the concepts of superclasses and subclasses into an ER model?*

There are two important reasons for introducing the concepts of superclasses and subclasses into an ER model. Firstly, it avoids describing similar concepts more than once, thereby saving time for the designer and making the ER diagram more readable. Secondly, it adds more semantic information to the design in a form that is familiar to many people. For example, the assertions that 'Manager IS-A member of staff' and 'flat IS-A type of property', communicates significant semantic content in a concise form (see Section 13.1.2).

13.5 *Describe what a shared subclass represents and how does this concept relate to multiple inheritance.*

A subclass with more than one superclass is called a **shared subclass**. The subclass will inherit the attributes of all its superclasses (i.e. multiple inheritance).

13.6 *Describe and contrast the process of specialization with the process of generalization.*

Specialization is the process of maximizing the differences between members of an entity by identifying their distinguishing features. **Generalization** is the process of minimizing the differences between entities by identifying their common features. Specialization is a top-down approach whereas generalization is a bottom-up approach (see Sections 13.1.4 and 13.1.5).

13.7 *Describe the two main constraints that apply to a specialization/generalization relationship.*

Two main constraints are: participation and disjoint constraints (see Section 13.1.6).

13.8 *Describe and contrast the concepts of aggregation and composition and provide an example of each.*

Aggregation represents a 'has-a' or 'is-part-of' relationship between entity types, where one represents the 'whole' and the other 'the part'. **Composition** is a specific form of aggregation that represents an association between entities, where there is a strong ownership and coincidental lifetime between the 'whole' and the 'part' (see Sections 13.2 and 13.3).

Exercises

13.11 *Introduce specialization/generalization concepts into the ER model shown in Figure 13.11 and described in Exercise 12.13 to show the following:*

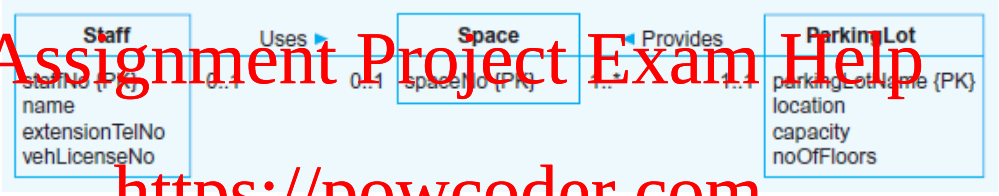


Figure 13.11

(a) The majority of parking spaces are under cover and each can be allocated for use by a member of staff for a monthly rate.
(b) Parking spaces that are not under cover are free to use and each can be allocated for use by a member of staff.
(c) Up to twenty covered parking spaces are available for use by visitors to the company. However, only members of staff are able to book out a space for the day of the visit. There is no charge for this type of booking, but the member of staff must provide the visitor's vehicle license number.

The final answer to this question is shown as Figure 17.11.

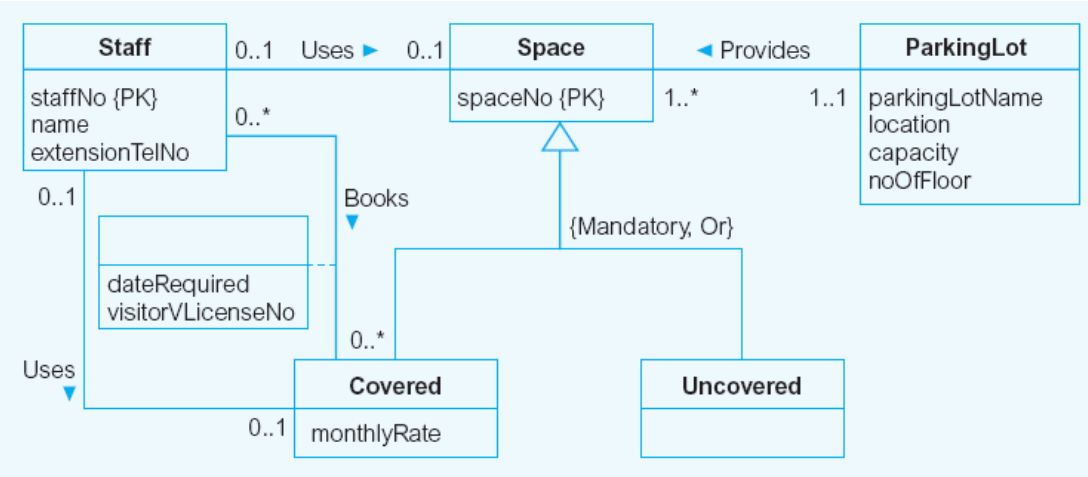


Figure 17.11

13.12 *The library case study described in Exercise 12.14 is extended to include the fact that the library has a significant stock of books that are no longer suitable for loaning out. These books can be sold for a fraction of the original cost. However, not all library books are eventually sold as many are considered too damaged to sell on, or are simply lost or stolen. Each book copy that is suitable for selling has a price and the date that the book is no longer to be loaned out. Introduce enhanced concepts into the ER model shown in Figure 13.12 and described in Exercise 12.14 to accommodate this extension to the original case study.*

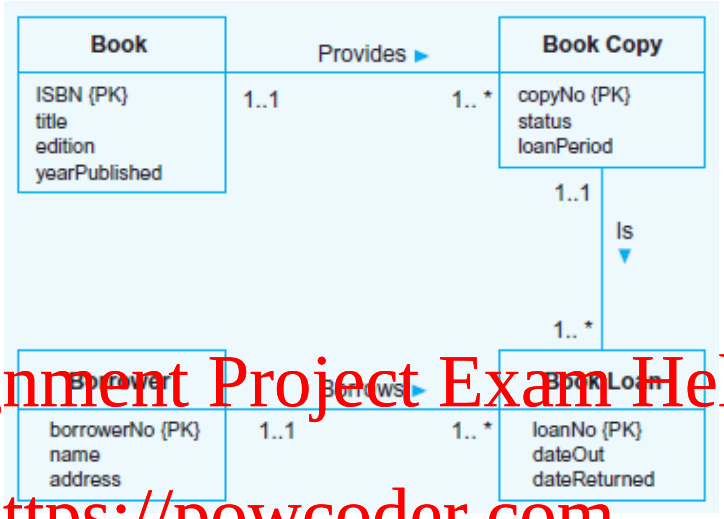


Figure 13.12

The answer to this exercise is shown as Figure 17.12.

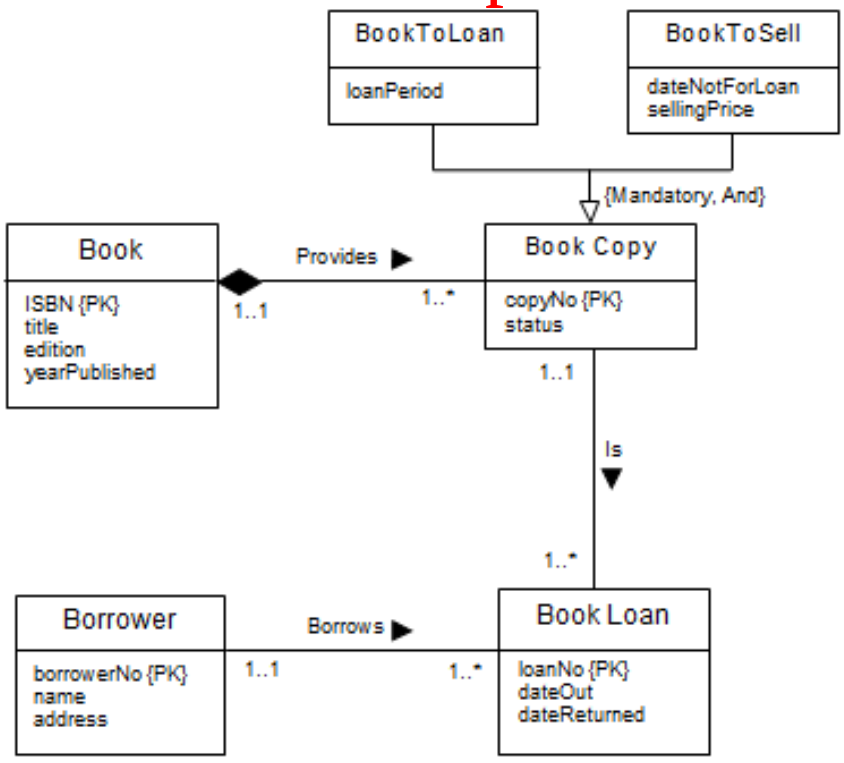


Figure 17.12

Chapter 14 – Normalization – Answers to Review Questions and Exercises

14.1 *Describe the purpose of normalizing data.*

The purpose of normalization is to identify a suitable set of relations that support the data requirements of an enterprise. The characteristics of a suitable set of relations include the following:

- the *minimal* number of attributes necessary to support the data requirements of the enterprise;
- attributes with a close logical relationship (described as functional dependency) are found in the same relation;
- *minimal* redundancy with each attribute represented only once with the important exception of attributes that form all or part of foreign keys (see Section 4.2.5), which are essential for the joining of related relations.

The benefit of using a database that has a suitable set of relations is that the database will be easier for the user to access and maintain the data, and take up minimal storage space on the computer.

14.2 *Discuss the alternative ways that normalization can be used to support database design.*

Normalization is a formal technique that can be used at any stage of database design. However in this section we highlight two main approaches for using normalization as illustrated in Figure 14.1. Approach (1) shows how normalization can be used as a bottom-up standalone database design technique while Approach (2) shows how normalization can be used as a validation technique to check the structure of relations, which may have been created using a top-down approach such as ER modeling. No matter which approach is used the goal is the same that of creating a set of well-designed relations that meet the data requirements of the enterprise. Figure 14.1 shows examples of data sources that can be used for database design. Although, the users' requirements specification (see Section 10.5) is the preferred data source, it is possible to design a database based on the information taken directly from other data sources such as forms and reports as illustrated in this chapter and the next. Figure 14.1 also shows that the same data source can be used for both approaches, however, although this is true in principle, in practice the approach taken is likely to be determined by the size, extent, and complexity of the database being described by the data source, and by the preference and expertise of the database designer. The opportunity to use normalization as a bottom-up standalone technique (Approach 1) is often limited by the level of detail that the database designer is reasonably expected to manage. However, this limitation is not applicable when normalization is used as a validation technique (Approach 2) as the database designer only focuses on part of the database such as a single relation at any one time. Therefore, no matter what the size or complexity of the database then normalization can be usefully applied.

See Figure 14.1

14.3 *Describe the types of update anomalies that may occur on a relation that has redundant data.*

A major aim of relational database design is to group attributes into relations so as to minimize information redundancy and thereby reduce the file storage space required by the base relations. Another serious difficulty using relations that have redundant information is the problem of update anomalies. These can be classified as insertion, deletion, or modification anomalies.

See Section 14.3

14.4 *Describe the concept of functional dependency.*

Functional dependency describes the relationship between attributes in a relation. For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted $A \rightarrow B$), if each value of A in R is associated with exactly one value of B in R.

Functional dependency is a property of the meaning or semantics of the attributes in a relation. The semantics indicate how the attributes relate to one another and specify the functional dependencies between attributes. When a functional dependency is present, the dependency is specified as a constraint between the attributes.

See also Section 14.3.

14.5 *What are the main characteristics of functional dependencies that are used when normalizing a relation?*

In summary, the functional dependencies that we use in normalization have the following characteristics:

- There is a *one-to-one* relationship between the attribute(s) on the left-hand side (determinant) and those on the right-hand side of a functional dependency. (Note that the relationship in the opposite direction; that is between the right to the left-hand side attributes can be a one-to-one relationship or one-to-many relationship).
- Holds for *all* time.
- The determinant has the *minimal* number of attributes necessary to maintain the dependency with the attribute(s) on the right hand-side. In other words, there must be a full functional dependency between the attribute(s) on the left and right hand-side of the dependency.

14.6 *Describe how a database designer typically identifies the set of functional dependencies associated with a relation.*

Identifying all functional dependencies between a set of attributes should be relatively simple if the meaning of each attribute and the relationships between the attributes are well understood. This type of information may be provided by the enterprise in the form of discussions with users and/or appropriate documentation such as the users' requirements specification. However, if the users are unavailable for consultation and/or the documentation is incomplete then depending on the database application, it may be necessary for the database designer to use their common sense and/or experience to provide the missing information.

As a contrast to this, consider the situation where functional dependencies are to be identified in the absence of appropriate information about the meaning of attributes and their relationships. In this case, it may be possible to identify functional dependencies if sample data is available that is a true representation of *all* possible data values that the database may hold.

14.7 *Describe the characteristics of a table in Unnormalized Form (UNF) and describe how such a table is converted to a First Normal Form (1NF) relation.*

A table in unnormalized form contains one or more repeating groups. To convert to first normal form (1NF) either remove the repeating group to a new relation along with a copy of the original key attribute(s), or remove the repeating group by entering appropriate data in the empty columns of rows containing the repeating data (see Section 14.5).

14.8 *What is the minimal normal form that a relation must satisfy? Provide a definition for this normal form.*

Minimal normal form is 1NF: a relation in which the intersection of each row and column contains one and only one value (see Section 14.5).

14.9 *Describe the two approaches to converting an Unnormalized Normal Form (UNF) table to First Normal Form (1NF) relation(s).*

There are two common approaches to removing repeating groups from unnormalized tables:

(a) *By entering appropriate data in the empty columns of rows containing the repeating data.* In other words, we fill in the blanks by duplicating the nonrepeating data, where required. This approach is commonly referred to as 'flattening' the table.

(b) *By placing the repeating data, along with a copy of the original key attribute(s), in a separate relation.* Sometimes the unnormalized table may contain more than one repeating group, or repeating groups within repeating groups. In such cases, this approach is applied repeatedly until no repeating groups remain. A set of relations is in 1NF if they contain no repeating groups.

For both approaches, the resulting tables are now referred to as 1NF relations containing atomic (or single) values at the intersection of each row and column. Although both approaches are correct, while approach (a) introduces more redundancy into the original UNF table as part of the 'flattening' process, approach (b) creates two or more relations with less redundancy than in the original UNF table. In other words, approach (b) moves the original UNF table further along the normalization process than approach (a). However, no matter which initial approach is taken the original UNF table will be normalized into the same set of 3NF relations.

14.10 *Describe the concept of 'full functional dependency' and describe how this concept relates to 2NF. Provide an example to illustrate your answer.*

Full functional dependency Indicates that if A and B are attributes of a relation, B is fully functionally dependent on A if B is functionally dependent on A, but not on any proper subset of A.

Second Normal Form (2NF) is a relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on the primary key.

14.11 *Describe the concept of transitive dependency and describe how this concept relates to 3NF. Provide an example to illustrate your answer.*

Transitive dependency A condition where A, B, and C are attributes of a relation such that if A $\rightarrow$ B and B $\rightarrow$ C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C)

Third Normal Form (3NF) is a relation that is in first and second normal form in which no non-primary-key attribute is transitively dependent on the primary key.

14.12 *Discuss how the definitions of 2NF and 3NF based on primary keys differ from the general definitions of 2NF and 3NF. Provide an example to illustrate your answer.*

The above definitions for second (2NF) and third normal form (3NF) disallow partial or transitive dependencies on the *primary key* of relations to avoid update anomalies. However, these definitions do not take into account other candidate keys of a relation, if any exist. The more general definitions for 2NF and 3NF take account of the candidate keys of a relation. Note that this requirement does not alter the definition for 1NF as this normal form is independent of keys and functional dependencies. For the general definitions, we define that a primary-key attribute is part of any candidate key and that partial, full, and transitive dependencies are with respect to all candidate keys of a relation.

Second normal form (2NF) A relation that is in First Normal Form and every non-candidate-key attribute is fully functionally dependent on *any candidate key*.

Third normal form (3NF) A relation that is in First and Second Normal Form and in which no non-candidate-key attribute is transitively dependent on *any candidate key*.

When using the general definitions of 2NF and 3NF we must be aware of partial and transitive dependencies on all candidate keys and not just the primary key. This can make the process of normalization more complex, however the general definitions place additional constraints on the relations and may identify hidden redundancy in relations that could be missed.

Exercises

14.13 *Continue the process of normalizing the* Client *and* PropertyRentalOwner *1NF relations shown in Figure 14.13 to 3NF relations. At the end of this process check that the resultant 3NF relations are the same as those produce from the alternative* ClientRental *1NF relation shown in Figure 14.16.*

The benefits of using the approach that creates two or more relations from the UNF table is that some or all of the resulting tables may already be in 3NF. This is true for the Client table, which is in 3NF. However, the PropertyRentalOwner table is only in 1NF due to the presence of a partial dependency and a transitive dependency.

The PropertyRentalOwner relation is converted to 2NF with the removal of a partial dependency. The result is creation of a new relation called PropertyOwner (2NF) and a relation called Rental, which is in 3NF.

The PropertyOwner (2NF) relation is converted to 3NF with the removal of a transitive dependency. The result is the creation of a new relation called PropertyForRent (3NF) and Owner (3NF).

In summary, the process of normalizing the PropertyRentalOwner (1NF) relation resulted in the creation of the Rental (3NF), PropertyForRent (3NF), and Owner (3NF) relations, which together with the Client relation is the same four relations created from the ClientRental 1NF relation shown in Figure 13.16.

14.14 *Examine the Patient Medication Form for the Wellmeadows Hospital case study shown in Figure 14.18.*

(a) Identify the functional dependencies represented by the data shown in the form in Figure 14.18.

patientNo → fullName
wardNo → wardName
wardName → wardNo
drugNo → name, description, dosage, methodOfAdmin
patientNo, drugNo, startDate → unitsPerDay, finishDate

The functional dependencies for bedNo are unclear. If bedNo was a unique number for the entire hospital, then could say that bedNo → wardNo. However, from further examination of the requirements specification, we can observe that bedNo is to do with the allocation of patients on the waiting list to beds.

(b) *Describe and illustrate the process of normalizing the data shown in Figure 14.18 to First (1NF), Second (2NF), and Third (3NF).*

First Normal Form

patientNo, drugNo, startDate, fullName, wardNo, wardName, bedNo, name, description, dosage, methodOfAdmin, unitsPerDay, finishDate

Second Normal Form

patientNo, drugNo, startDate, wardNo, wardName, bedNo, unitsPerDay, finish Date

drugNo, name, description, dosage, methodOfAdmin

patientNo, fullName

Third Normal Form

patientNo, drugNo, startDate, wardNo, bedNo, unitsPerDay, finish Date

drugNo, name, description, dosage, methodOfAdmin

patientNo, fullName

wardNo, wardName

(c) *Identify the primary, alternate, and foreign keys in your 3NF relations.*

patientNo (PK), drugNo (FK), startDate, wardNo (FK), bedNo, unitsPerDay, finish Date

drugNo, name, description, dosage, methodOfAdmin

patientNo, fullName

wardNo, wardName (AK)

(Primary keys underlined.)

14.15 *The table shown in Figure 14.19 lists dentist/patient appointment data. A patient is given an appointment at a specific time and date with a dentist located at a particular surgery. On each day of patient appointments, a dentist is allocated to a specific surgery for that day.*

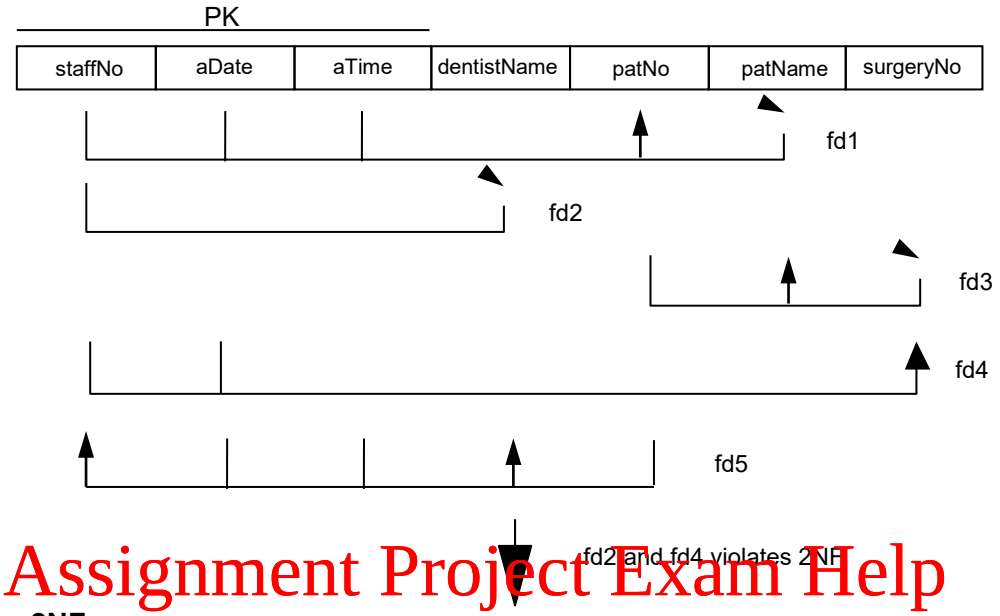
(a) *The table shown in Figure 14.19 is susceptible to update anomalies. Provide examples of insertion, deletion, and update anomalies.*

The student should provide examples of insertion, deletion and update anomalies using the data shown in the table. An example of a deletion anomaly is if we delete the details of the dentist called 'Helen Pearson', we also loose the appointment details of the patient called 'Ian MacKay'.

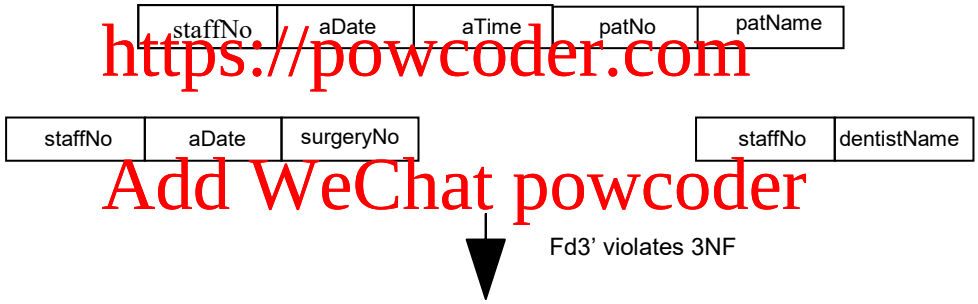
(b) *Describe and illustrate the process of normalizing the table shown in Figure 14.19 to 3NF. State any assumptions you make about the data shown in this table.*

The student should state any assumptions made about the data shown in the table. For example, we may assume that a patient is registered at only one surgery. Also, a patient may have more than one appointment on a given day.

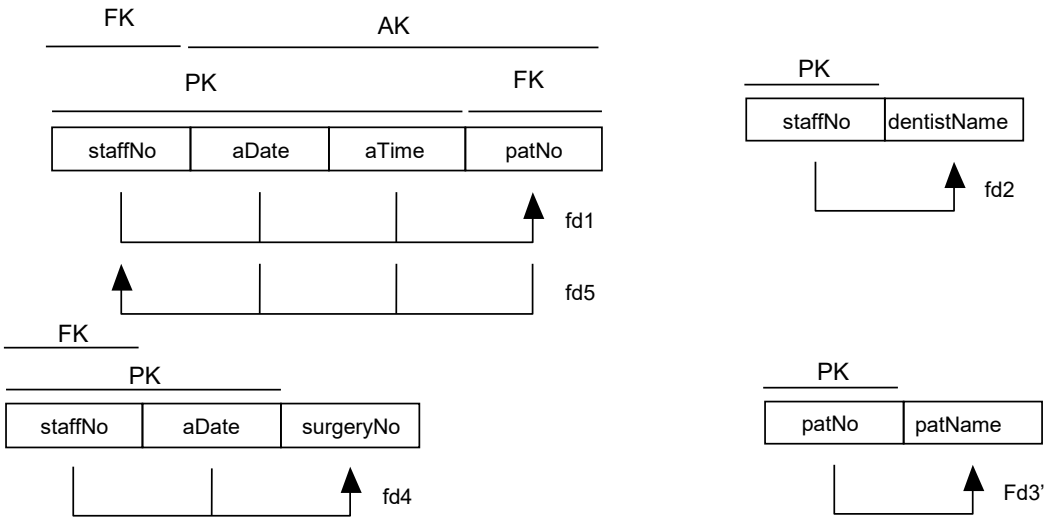
1NF



2NF



3NF



Chapter 15 – Advanced Normalization – Answers to Review Questions and Exercises

15.1 *Describe the purpose of using inference rules to identify functional dependencies for a given relation.*

Even if we restrict our attention to nontrivial functional dependencies with one-to-one (1:1) relationships that hold for all time, the complete set of functional dependencies for a given relation can still be very large. It is important to find an approach that can reduce that set to a manageable size. Ideally, we want to identify a set of functional dependencies (represented as X) for a relation that is smaller than the complete set of functional dependencies (represented as Y) for that relation and has the property that every functional dependency in Y is implied by the functional dependencies in X. Hence, if we enforce the integrity constraints defined by the functional dependencies in X, we automatically enforce the integrity constraints defined in the larger set of functional dependencies in Y. This requirement suggests that there must be functional dependencies that can be inferred from other functional dependencies. For example, functional dependencies $A \rightarrow B$ and $B \rightarrow C$ in a relation implies that the functional dependency $A \rightarrow C$ also holds in that relation. $A \rightarrow C$ is an example of a **transitive** functional dependency.

How do we begin to identify useful functional dependencies on a relation? Normally, the database designer starts by specifying functional dependencies that are semantically obvious; however, there are usually numerous other functional dependencies. In fact, the task of specifying all possible functional dependencies for 'real' database projects is more often than not, impractical. However, in this section we do consider an approach that helps identify the complete set of functional dependencies for a relation and then discuss how to achieve a minimal set of functional dependencies that can represent the complete set.

15.2 *Discuss the purpose of Armstrong's axioms.*

The set of all functional dependencies that are implied by a given set of functional dependencies X is called the **closure** of X, written X^+. We clearly need a set of rules to help compute X^+ from X. A set of inference rules, called **Armstrong's axioms**, specifies how new functional dependencies can be inferred from given ones (Armstrong, 1974). For our discussion, let A, B, and C be subsets of the attributes of the relation R. Armstrong's axioms are as follows:

(1) **Reflexivity:** If B is a subset of A, then $A \rightarrow B$

(2) **Augmentation:** If $A \rightarrow B$, then $A,C \rightarrow B,C$

(3) **Transitivity:** If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$

Note that each of these three rules can be directly proved from the definition of functional dependency. The rules are **complete** in that given a set X of functional dependencies, all functional dependencies implied by X can be derived from X using these rules. The rules are also **sound** in that no additional functional dependencies can be derived that are not implied by X. In other words, the rules can be used to derive the closure of X^+.
Several further rules can be derived from the three given above that simplify the practical task of computing X^+.

(See Section 15.1)

To begin to identify the set of functional dependencies F for a relation, typically we first identify the dependencies that are determined from the semantics of the attributes of the relation. Then, we apply Armstrong's axioms (Rules 1 to 3) to infer additional functional dependencies that are also true for that relation. A systematic way to determine these additional functional dependencies is to first determine each set of attributes A that appears on the left-hand side of some functional dependencies and then to determine the set of *all* attributes that are dependent on A. Thus, for each set of attributes A we can determine the set A^+ of attributes that are functionally determined by A based on F; (A^+ is called the **closure of A under F**).

15.3 *Discuss the purpose of Boyce-Codd Normal Form (BCNF) and describe how BCNF differs from 3NF. Provide an example to illustrate your answer.*

BCNF is based on functional dependencies that take into account all candidate keys in a relation, however BCNF also has additional constraints compared with the general definition of 3NF given above.

Boyce–Codd normal form (BCNF)
A relation is in BCNF if and only if every determinant is a candidate key.

To test whether a relation is in BCNF, we identify all the determinants and make sure that they are candidate keys.

See Sections 15.2 and 15.3.

Exercises

15.6 *On completion of Exercise 14.14 examine the 3NF relations created to represent the attributes shown in the Wellmeadows Hospital form shown in Figure 14.18. Determine whether these relations are also in BCNF. If not, transform the relations that do not conform into BCNF.*

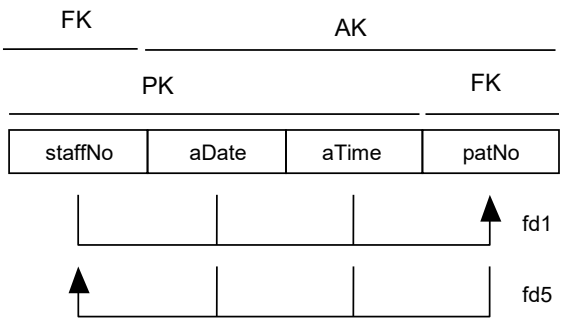
The only relations that may violate BCNF are those that have more than one candidate key. Therefore we need only re-examine the Ward relation, which has a wardNo as a PK and wardName as an alternate key. This relation contains the following functional dependencies:

wardNo → wardName (fd1)
wardName → wardNo (fd2)

The presence of fd2 does not break BCNF because wardName is a candidate key for this relation. Hence the Ward relation is in BCNF.
As the other relations shown in the answer for Exercise 14.14 have only one candidate key, they must also be in BCNF.

15.7 On completion of Exercise 14.15 examine the 3NF relations created to represent the attributes shown in the relation that displays dentist/patient appointment data in Figure 14.19. Determine whether these relations are also in BCNF. If not, transform the relations that do not conform into BCNF.

The only relations that may violate BCNF are those that have more than one candidate key. Therefore we need only re-examine the Appointment relation, which has (staffNo, aDate, aTime) as a PK and (patNo, aDate, aTime) as an alternate key. This relation contains the following functional dependencies:



The presence of fd5 does not break BCNF because (patNo, aDate, aTime) is a candidate key for this relation. Hence the Appointment relation is in BCNF.
As the other relations shown in the answer for Exercise *14.15* have only one candidate key, they must also be in BCNF.

Chapter 16 – Methodology – Conceptual Database Design – Answers to Review Questions and Exercises

16.1 *Describe the purpose of a design methodology.*

A structured approach that uses procedures, techniques, tools, and documentation aids, to support and facilitate the process of design. A design methodology consists of phases that contain steps, which guide the designer in the choice of techniques that are appropriate at each stage of the project and also helps to plan, manage, control and evaluate database development projects. Furthermore, it is a structured approach for analyzing and modeling a set of requirements for a database in a standardized and organized manner.

See Section 16.1.

16.2 *Describe the main phases involved database design.*

Conceptual database design – to build the conceptual representation of the database, which includes identification of the important entities, relationships, and attributes.
Logical database design – to translate the conceptual representation to the logical structure of the database, which includes designing the relations.
Physical database design – to decide how the logical structure is to be physically implemented (as relations) in the Target Database Management System (DBMS).

See Section 16.1.2 for complete definitions.

16.3 *Identify important factors in the success of logical database design.*

- Work interactively with the users as much as possible
- Follow a structured methodology throughout the data modeling process
- Employ a data-driven approach
- Incorporate structural and integrity considerations into the data models
- Combine conceptualization, normalization, and transaction validation techniques into the data modeling methodology
- Use diagrams to represent as much of the data models as possible
- Use a Database Design Language (DBDL) to represent additional data semantics that cannot easily be represented in a diagram
- Build a data dictionary to supplement the data model diagrams and the DBDL
- Be willing to repeat steps.

See Section 16.1.3.

16.4 *Discuss the important role played by users in the process of database design.*

The users' involvement throughout the database design phase is critical to providing the 'correct' system. In particular, the user should clarify any ambiguities in the specification that describes the required system and also review continually the development of the database design. The process of developing the database design is repeated until the user is prepared to 'sign-off' the design as being a 'true' representation of the part of the enterprise that is being modeling.

See Sections 16.1.3 and 16.2 (Step 1.9).

16.5 *Describe the main objective of conceptual database design.*

See Section 15.3 Step 1.

16.6 *Identify the main steps associated with conceptual database design.*

See start of Section 16.3 Step 1.

16.7 *How would you identify entity and relationship types from a user's requirements specification?*

One method of identifying entities is to examine the users' requirements specification. From this specification, we identify nouns or noun phrases that are mentioned (for example, staff number, staff name, property number, property address, rent, number of rooms). We also look for major objects such as people, places, or concepts of interest, excluding those nouns that are merely qualities of other objects. (See Step 1.1)

Having identified the entities, the next step is to identify all the relationships that exist between these entities. When we identify entities, one method is to look for nouns in the users' requirements specification. Again, we can use the grammar of the requirements specification to identify relationships. Typically, relationships are indicated by verbs or verbal expressions. (See Step 1.2)

16.8 *How would you identify attributes from a user's requirements specification and then associate the attributes with entity or relationship types?*

In a similar way to identifying entities, we look for nouns or noun phrases in the users' requirements specification. The attributes can be identified where the noun or noun phrase is a property, quality, identifier, or characteristic of one of these entities or relationships. By far the easiest thing to do when we have identified an entity (x) or a relationship (y) in the requirements specification is to consider *What information are we required to hold on x or y*? The answer to this question should be described in the specification. However, in some cases it may be necessary to ask the users to clarify the requirements. Unfortunately, they may give answers to this question that also contain other concepts, so that the users' responses must be carefully considered.

As there are generally many more attributes than entities and relationships, it may be useful to first produce a list of all attributes given in the users' requirements specification. As an attribute is associated with a particular entity or relationship, remove the attribute from the list. In this way, we ensure that an attribute is associated with only one entity or relationship type and, when the list is empty, that all attributes are associated with some entity or relationship type.

16.9 *Describe the purpose of specialization/generalization of entity types, and discuss why this is an optional step in conceptual database design.*

See Section 16.3 Step 1.6.

16.10 *How would you check a data model for redundancy? Give an example to illustrate your answer.*

(1) Re-examine one-to-one (1:1) relationships

In the identification of entities, we may have identified two entities that represent the same object in the enterprise. For example, we may have identified the two entities Client and Renter that are actually the same; in other words, Client is a synonym for Renter. In this case, the two entities should be merged together. If the primary keys are different, choose one of them to be the primary key and leave the other as an alternate key.

(2) Remove redundant relationships

A relationship is redundant if the same information can be obtained via other relationships. We are trying to develop a minimal data model and, as redundant relationships are unnecessary, they should be removed. It is relatively easy to identify whether there is more than one path between two entities. However, this does not necessarily imply that one of the relationships is redundant, as they may represent different associations between the entities. See discussion using an example in Step 1.7 Part (2).

(3) Consider time dimension

The time dimension is important when assessing redundancy. See discussion using an example in Step 1.7 Part (3).

16.11 *Discuss why you would want to validate a conceptual data model and describe two approaches to validating a conceptual model.*

We validate a conceptual data model to check the model to ensure that the model supports the transactions required by this view. Using the model, we attempt to perform the operations manually. If we can resolve all transactions in this way, we have checked that the conceptual data model supports the required transactions. However, if we are unable to perform a transaction manually there must be a problem with the data model, which must be resolved. In this case, it is likely that we have omitted an entity, a relationship, or an attribute from the data model.

Two possible approaches to ensuring that the local conceptual data model supports the required transactions are:

(1) describing the transactions;

(2) using transaction pathways.

See Section 16.3 Step 1.8.

Chapter 16 – Methodology – Conceptual Database Design – Answers to Review Questions and Exercises

16.12 *Identify and describe the purpose of the documentation generated during conceptual database design.*

See Section 16.3 and in particular note the documentation generated at the end of each step.

The *University Accommodation Office* case study

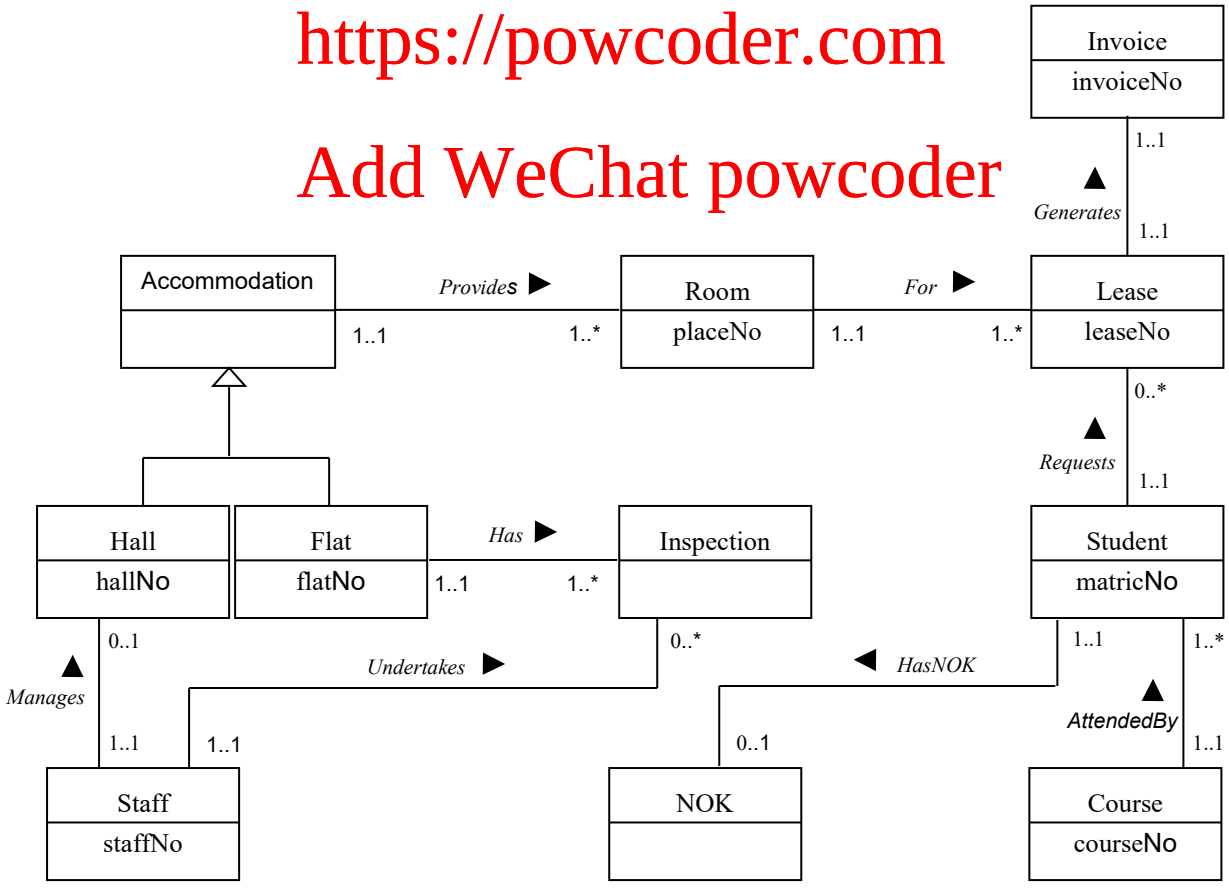
16.15 *Provide a user's requirements specification for the University Accommodation Office case study documented in Appendix B.1.*

There are the nine obvious categories given by the sub-titles in the case study:

Students	Halls of residence
Student flats	Leases
Invoices	Student apartment inspections
Residence staff	Courses
Next of Kin	

The Halls and Flats can generalize into generic Accomodations, which provide Rooms.

16.16 *Create a conceptual data model for the case study. State any assumptions necessary to support your design. Check that the local conceptual data model supports the required transactions.*



Chapter 17 – Methodology – Logical Database Design for the Relational Model – Answers to Review Questions and Exercises

17.1 *Discuss the purpose of logical database design.*

For the purpose of logical database design see Section 17.1.

17.2 *Describe the rules for deriving relations that represent:*

(a) strong entity types;
(b) weak entity types;
(c) one-to-many (1:) binary relationship types;*
(d) one-to-one (1:1) binary relationship types;
(e) one-to-one (1:1) recursive relationship types;
(f) superclass/subclass relationship types;
(g) many-to-many (:*) binary relationship types;*
(h) complex relationship types;
(i) multi-valued attributes.

See Tables 17.1 and 17.2

17.3 *Discuss how the technique of normalization can be used to validate the relations derived from the conceptual data model.*

The logical data model can be validated using the technique of nomalization and against the transactions that the model is required to support. Normalization is used to improve the model so that it satisfies various constraints that avoid unnecessary duplication of data. Normalization ensures that the resultant model is a closer model of the enterprise that it serves, it is consistent, and has minimal redundancy and maximum stability.

See also Section 17.1 Step 2.2.

17.4 *Discuss two approaches that can be used to validate that the relational schema is capable of supporting the required transactions.*

Two possible approaches to ensure that the logical data model supports the required transactions include: checking that all the information (entities, relationships and their attributes) required by each transaction is provided by the model in documenting a description of each transaction's requirements, and diagrammatically representing the pathway taken by each transaction directly on the ER diagram.

See also Section 17.1 Step 2.3.

17.5 *Describe the purpose of integrity constraints and identify the main types of constraints on a logical data model.*

There are six types of integrity constraints, namely: required data, attribute domain constraints, multiplicity, entity integrity, referential integrity, and general constraints.

See also Section 17.1 Step 2.4.

17.6 *Describe the alternative strategies that can be applied if there exists a child occurrence referencing a parent occurrence that we wish to delete.*

There are several strategies to consider when there exists a child occurrence referencing the parent occurrence that we are attempting to delete: NO ACTION, CASCADE, SET NULL, SET DEFAULT, and NO CHECK.

See Section 17.1 Step 2.4 for detailed discussion on Referential Integrity.

17.7 *Identify the tasks typically associated with merging local logical data models into a global logical model.*

See Section 17.1 Step 2.6.

The *University Accommodation Office* case study

17.10 *Create and validate a logical data model from the conceptual data model for the University Accommodation Office case study created in Exercise 16.16.*

Student (matricNo, fName, lName, street, city, postcode, DOB, sex, category, nationality, smoker, specialNeeds, comments, status, courseNo)
Primary Key matricNo
Foreign Key courseNo references Course(courseNo)

NOK (matricNo, name, street, city, postcode, contactTelNo)
Primary Key matricNo
Foreign Key matricNo references Student(matricNo)

Course (courseNo, courseTitle, courseLeader, courseLeaderTelNo,
courseLeaderRoomNo, deptName)
Primary Key courseNo
Alternate Key courseTitle

Hall (hallNo, hName, hAddress, hTelNo, mgrStaffNo)
Primary Key hallNo
Alternate Key hName
Alternate Key hTelNo
Foreign Key mgrStaffNo references Staff(staffNo)

Flat (flatNo, fAddress, noOfRooms)
Primary Key flatNo

Room (placeNo, roomNo, monthlyRent, flatNo, hallNo)
Primary Key placeNo
Foreign Key hallNo references Hall(hallNo)
Foreign Key flatNo references Flat(flatNo)

FlatInspection (flatNo, iDate, condition, comments, inspStaffNo)
Primary Key flatNo, iDate
Foreign Key flatNo references Flat(flatNo)
Foreign Key inspStaffNo references Staff(staffNo)

Lease (leaseNo, duration, matricNo, placeNo, dateEnter, dateLeave)
Primary Key leaseNo
Alternate Key placeNo, dateEnter
Alternate Key matricNo, dateEnter
Foreign Key matricNo references Student(matricNo)
Foreign Key placeNo references Room(placeNo)

AccommStaff (staffNo, fName, lName, street, city, postcode, DOB, sex, position, location)
Primary Key staffNo
Foreign Key location references Hall(hallNo)

Invoice (invoiceNo, semester, paymentDue, datePaid, paymentMethod, dateReminder1, dateReminder2, leaseNo)
Primary Key invoiceNo
Foreign Key leaseNo references Lease(leaseNo)

Chapter 18 – Methodology – Physical Database Design for the Relational Model – Answers to Review Questions and Exercises

18.1 *Explain the difference between conceptual, logical, and physical database design. Why might these tasks be carried out by different people?*

Conceptual database design is concerned with building the data model for the organization that is completely independent of any physical considerations including DBMS (logical database design assumes a particular DBMS, for example, relational, but is independent of any other physical considerations). Physical database design, however, is concerned with actually defining the data model using the DDL of a particular DBMS. Consequently, conceptual/logical database design focuses on what the data model represents, whereas physical database design focuses on how the data model is to be implemented. Different skills are required to undertake these design phases, which are often found in different people. See Section 18.1 (and 16.1.2).

18.2 *Describe the inputs and outputs of physical database design.*

The inputs are the logical data model and the data dictionary. The outputs are the base relations, integrity rules, file organization specified, secondary indexes determined, user views and access rules. See Section 18.3.

18.3 *Describe the purpose of the main steps in the physical design methodology presented in this chapter.*

Step 3 Produces a relational database schema from the logical data model. This includes integrity rules.

Step 4 Determines the file organizations for the base relations. This takes account of the nature of the transactions to be carried out, which also determine where secondary indexes will be of use. As a result of analyzing the transactions, the design may be altered by incorporating controlled redundancy into it.

Step 5 Designs the user views for the database implementation.

Step 6 Designs the security measures for the database implementation.

See Section 18.2.

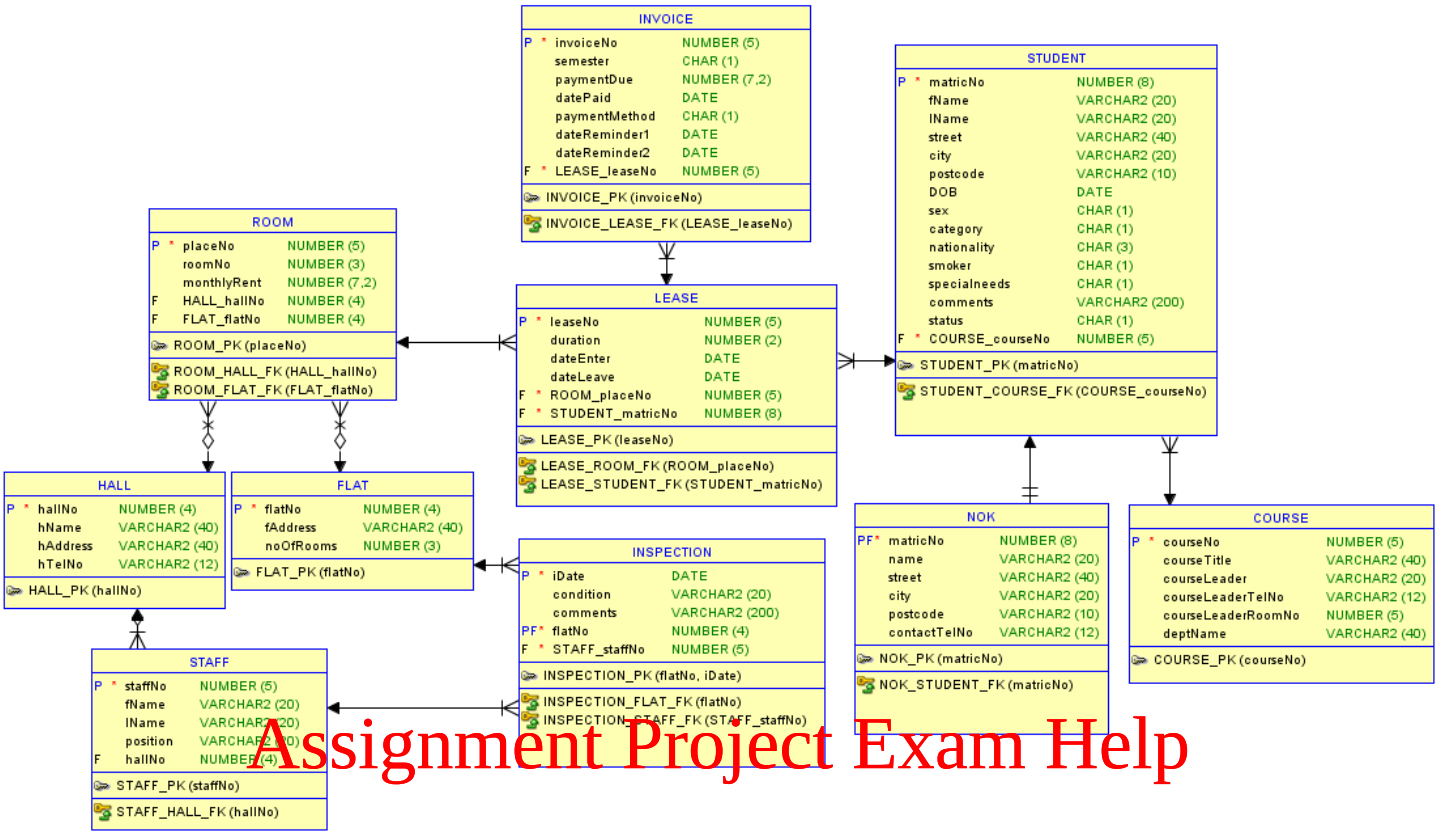
18.4 *Discuss when index may improve the efficiency of the system.*

See ‘Guidelines for choosing a wish-list of indexes’ in Step 4.3.

The *University Accommodation Office* case study

18.9 *Based on the logical data model developed in Exercise 17.10, create a physical database design for the University Accommodation Office case study (described in Appendix B.1) based on the DBMS that you have access to.*

The logical design solution is available for Exercise 17.10. All that is needed is to produce the physical design is choosing the appropriate data types. Below is a screen shot of the design produced in Oracle Datamodeler.



18.10 *Implement the University Accommodation Office database using the physical design created in 18.9.*

The SQL DDL can be generated from the Oracle Datamodeler tool directly. It will depend on the datatypes chosen, which fields are 'Mandatory' so produce a NOT NULL constraint, etc. The only other consideration would be indexes. Generally you want to enable parent to child look-ups which are usually on FK's as well as common keys. Following is a suggested list:

STUDENT	lName, fName
	courseNo
COURSE	courseTitle
NOK	name
INVOICE	leaseNo
LEASE	matricNo
	placeNo
ROOM	hallNo
	flatNo
HALL	hName
STAFF	lName, fName
	hallNo
INSPECTION	staffNo

Chapter 9 – Object-Relational DBMSs – Answers to Review Questions and Exercises

9.1 Discuss the general characteristics of advanced database applications.

Designs of this type have some common characteristics:

- Design data is characterized by a large number of types, each with a small number of instances. Conventional databases are typically the opposite.
- Designs may be very large, perhaps consisting of millions of parts, often with many interdependent subsystem designs.
- The design is not static but evolves through time. When a design change occurs, its implications must be propagated through all design representations. The dynamic nature of design may mean that some actions cannot be foreseen at the beginning.
- Updates are far-reaching because of topological or functional relationships, tolerances, and so on. One change is likely to affect a large number of design objects.
- Often, many design alternatives are being considered for each component, and the correct version for each part must be maintained. This involves some form of version control and configuration management.
- There may be hundreds of staff involved with the design, and they may work in parallel on multiple versions of a large design. Even so, the end product must be consistent and coordinated. This is sometimes referred to as *cooperative engineering*.

See Section 9.1.

9.2 Discuss why the weaknesses of the relational data model and relational DBMSs may make them unsuitable for advanced database applications.

Weaknesses discussed in Section 9.2.

9.3 Discuss the difficulties involved in mapping objects created in an object-oriented programming language to a relational database.

Discussion should centre around the loss of semantic information in mapping a hierarchical structure to a flat relational structure. Also discussion of the reverse process of recreating the original structure from flat relations, which requires additional software to be written. Both cases result in a loss of performance. See Section 9.3.

9.4 What functionality would typically be provided by an ORDBMS?

Many different answers here – see, for example, Section 9.4.
Expect standard DBMS functionality, plus object management capabilities (types, inheritance, etc), plus ability to extend query optimizer and define new index types.

9.5 What are the advantages and disadvantages of extending the relational data model?

Advantages

Apart from the advantages of resolving many of the weaknesses of the relational data model cited in Section 9.2, the main advantages of extending the relational data model come from *reuse* and *sharing*. Reuse comes from the ability to extend the DBMS server to perform standard functionality centrally, rather than have it coded in each application. If we can embed this functionality in the server, it saves having to define it in each application that needs it, and consequently allows the functionality to be shared by all applications. These advantages also give rise to increased productivity both for the developer and for the end-user.

Another obvious advantage is that the extended relational approach preserves the significant body of knowledge and experience that has gone into developing relational applications. This is a significant advantage, as many organizations would find it prohibitively expensive to change. If the new functionality is designed appropriately, this approach should allow organizations to take advantage of the new extensions in an evolutionary way without losing the benefits of current database features and functions. Thus, an ORDBMS could be introduced in an integrative fashion, as proof-of-concept projects.

Disadvantages

The ORDBMS approach has the obvious disadvantages of complexity and associated increased costs. Further, there are the proponents of the relational approach that believe the essential simplicity and purity of the relational model are lost with these types of extensions. There are also those that believe that the RDBMS is being extended for what will be a minority of applications that do not achieve optimal performance with current relational technology.

In addition, object-oriented purists are not attracted by these extensions either. They argue that the terminology of object-relational systems is revealing. Instead of discussing object models, terms like user-defined data types are used. The terminology of object-orientation abounds with terms like abstract types, class hierarchies, and object models. However, ORDBMS vendors are attempting to portray object models as extensions to the relational model with some additional complexities. This potentially misses the point of object-orientation, highlighting the large semantic gap between these two technologies. Object applications are simply not as data-centric as relational-based ones. Object-oriented models and programs deeply combine relationships and encapsulated objects to more closely mirror the 'real world'. This defines broader sets of relationships than those expressed in SQL, and involves functional programs interspersed in the object definitions. In fact, objects are fundamentally not extensions of data, but a completely different concept with far greater power to express 'real-world' relationships and behaviors.

In Chapter 6, we noted that the objectives of a database language included having the capability to be used with minimal user effort, and having a command structure and syntax that must be relatively easy to learn. Unfortunately, the size of the new SQL:2011 standard is daunting, and it would seem that these two objectives are no longer being fulfilled or even being considered by the standards bodies.

See Section 9.4 under Advantages and Disadvantages.

9.6 What are the main features of the SQL:2011 standard?

New main features are:

- type constructors for row types and reference types;
- user-defined types (distinct types and structured types) that can participate in supertype/subtype relationships;
- user-defined procedures, functions, and methods;
- type constructors for collection types (arrays and multisets);
- support for large objects;
- recursion.

See start of Section 9.5.

9.7 Discuss how reference types and object identity can be used.

Until SQL:1999, the only way to define relationships between tables was using the primary key/foreign key mechanism, which in SQL2 could be expressed using the referential table constraint clause REFERENCES, as discussed in Section 7.2.4. Since SQL:1999, **reference types** can be used to define relationships between row types and uniquely identify a row within a table. A reference type value can be stored in one table and used as a direct reference to a specific row in some base table that has been defined to be of this type (similar to the notion of a pointer type in 'C' or C++). In this respect, a reference type provides a similar functionality as the object identifier (OID) of object-oriented DBMSs, which we discuss in Chapter 27. Thus, references allow a row to be shared among multiple tables and enable users to replace complex join definitions in queries with much simpler path expressions. References also give the optimizer an alternative way to navigate data instead of using value-based joins. See Section 9.5.6.

9.8 Compare and contrast procedures, functions, and methods.

User-defined routines discussed in Section 9.5.4.

9.9 What is a trigger? Provide an example of a trigger.

A trigger is an SQL (compound) statement that is executed automatically by the DBMS as a side effect of a modification to a named table. It is similar to an SQL routine, in that it is a named SQL block with declarative, executable, and condition-handling sections. However, unlike a routine, a trigger is executed implicitly whenever the *triggering event* occurs, and a trigger does not have any arguments. The act of executing a trigger is sometimes known as *firing* the trigger. See Section 9.5.12.

9.10 Discuss the collection types available in SQL:2011.

Collections are type constructors that are used to define collections of other types. Collections are used to store multiple values in a single column of a table and can result in nested tables where a column in one table actually contains another table. The result can be a single table that represents multiple master-detail levels. Thus, collections add flexibility to the design of the physical database structure. SQL:2011 supports ARRAY and MULTISET collections. See Section 9.5.9.

Exercises

9.15 *Consider the relational schema for the Hotel case study given in the Exercises at the end of Chapter 4. Redesign this schema to take advantage of the new features of SQL:2011. Add user-defined functions that you consider appropriate.*

One possible solution as follows:

```
CREATE DOMAIN RoomType AS CHAR(1)
        CHECK(VALUE IN ('S', 'F', 'D'));
CREATE DOMAIN HotelNumbers AS HotelNumber
        CHECK(VALUE IN (SELECT hotelNo FROM Hotel))
CREATE DOMAIN RoomPrice AS DECIMAL(5, 2)
        CHECK(VALUE BETWEEN 10 AND 100);
CREATE DOMAIN RoomNumber AS VARCHAR(4)
        CHECK(VALUE BETWEEN '1' AND '100');

CREATE DOMAIN HotelNumber AS CHAR(4);
CREATE DOMAIN GuestNumber AS CHAR(4);
CREATE DOMAIN BookingDate AS DATETIME
        CHECK(VALUE > CURRENT_DATE);

CREATE TYPE HotelType AS (
        hotelNo         HotelNumber         NOT NULL,
        hotelName       VARCHAR(20)         NOT NULL,
        city            VARCHAR(50)         NOT NULL)
REF IS SYSTEM GENERATED
INSTANTIABLE
FINAL;

CREATE TABLE Hotel OF HotelType(
        REF IS hotelID SYSTEM GENERATED,
        PRIMARY KEY (hotelNo));

CREATE TABLE Room (
        roomNo          RoomNumber          NOT NULL,
        hotelID         REF(HotelType)      SCOPE Hotel
                        REFERENCES ARE CHECKED ON DELETE CASCADE,
```

```
        type            RoomType            NOT NULL DEFAULT 'S'
        price           RoomPrice           NOT NULL,
        PRIMARY KEY (roomNo, hotelID));

CREATE TYPE GuestType AS (
        guestNo         GuestNumber         NOT NULL,
        guestName       VARCHAR(20)         NOT NULL,
        guestAddress    VARCHAR(50)         NOT NULL);
REF IS SYSTEM GENERATED
INSTANTIABLE
FINAL;

CREATE TABLE Guest OF GuestType(
        REF IS guestId SYSTEM GENERATED,
        PRIMARY KEY (guestNo));

CREATE TABLE Booking (
        hotelID         REF(HotelType)      SCOPE Hotel
                        REFERENCES ARE CHECKED ON DELETE CASCADE,
        guestID         REF(GuestType)      SCOPE Guest
                        REFERENCES ARE CHECKED ON DELETE CASCADE,
        dateFrom        BookingDate         NOT NULL,
        dateTo          BookingDate         NULL,
        roomNo          RoomNumber          NOT NULL,
        PRIMARY KEY (hotelID, guestID, dateFrom),
        FOREIGN KEY(hotelID) REFERENCES Hotel
                ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (guestID) REFERENCES Guest
                ON DELETE NO ACTION ON UPDATE CASCADE,
        FOREIGN KEY (hotelID, roomNo) REFERENCES Room
                ON DELETE NO ACTION ON UPDATE CASCADE,
        CONSTRAINT RoomBooked
        CHECK (NOT EXISTS (SELECT *
                                FROM Booking b
                                WHERE b.dateTo > Booking.dateFrom AND
                                b.dateFrom < booking.dateTo AND
                                AND b.roomNo = Booking.roomNo AND
                                b.hotelID = Booking.hotelID)),
        CONSTRAINT GuestBooked
        CHECK (NOT EXISTS (SELECT *
                                FROM Booking b
                                WHERE b.dateTo > Booking.dateFrom AND
                                b.dateFrom < Booking.dateTo AND
                                AND b.guestID = Booking.guestID)));
```

Chapter 27 – Object-Oriented DBMSs – Concepts and Design – Answers to Review Questions and Exercises

27.1 *Describe the three generations of DBMSs.*

First generation is the hierarchical and network DBMSs.
Second generation is the relational DBMSs.
Third generation is the object-oriented and object-relational DBMSs.

See Section 27.1.

27.2 *Compare and contrast the different definitions of object-oriented data models.*

Discussion of definitions provided in Section 27.2.1.

27.3 *Describe the main modeling primitives of the functional data model.*

The main modeling primitives are entities and functional relationships (see Section 27.2.2).

27.4 *What is a persistent programming language and how does it differ from an OODBMS?*

A language that provides its users with the ability to (transparently) preserve data across successive executions of a program, and even allows such data to be used by different programs (see Section 27.2.3). Main difference between PPL and OODBMS is that the OODBMS tends to provide more DBMS-related services.

27.5 *Discuss the difference between the two-level storage model used by conventional DBMSs and the single-level storage model used by OODBMSs.*

See Section 27.3.

27.6 *How does this single-level storage model affect data access?*

See Section 27.3.

27.7 *Describe the main strategies that can be used to create persistent objects.*

Checkpointing, serialization, and explicit paging. Explicit paging includes reachability-based and allocation-based persistence (see Section 27.3.3).

27.8 *What is pointer swizzling? Discuss the different approaches to pointer swizzling.*

Action of converting OIDs to main memory pointers (See Section 27.3.1).

27.9 *Describe the types of transaction protocol that can be useful in design applications.*

See Sections 27.4.1 and 22.4.

27.10 Discuss why version management may be a useful facility for some applications.

There are many applications that need access to the previous state of an object. For example, the development of a particular design is often an experimental and incremental process, the scope of which changes with time. It is therefore necessary in databases that store designs to keep track of the evolution of design objects and the changes made to a design by various transactions. See Section 27.4.2.

27.11 Discuss why schema control may be a useful facility for some applications.

Engineering design is an incremental process and evolves with time. To support this process, applications require considerable flexibility in dynamically defining and modifying the database schema. For example, it should be possible to modify class definitions, the inheritance structure, and specifications of attributes and methods without requiring system shutdown. See Section 27.4.3.

27.12 Describe the different architectures for an OODBMS.

See Section 27.4.4.

27.13 List the advantages and disadvantages of an OODBMS.

See Section 27.5.

27.14 Describe how relationships can be modelled in an OODBMS.

See the discussion on Section 27.6.2.

27.15 Describe the different modelling notations in the UML.

Expect discussion of the notations for:

- *Structural diagrams*, which describe the static relationships between components, and include:
  - class diagrams,
  - object diagrams,
  - component diagrams,
  - deployment diagrams.
- *Behavioral diagrams*, which describe the dynamic relationships between components, and include:
  - use case diagrams,
  - sequence diagrams,
  - collaboration diagrams,
  - statechart diagrams,
  - activity diagrams.

Exercises

27.16 *For the DreamHome case study documented in Appendix A, suggest attributes and methods that would be appropriate for* Branch, Staff, *and* PropertyForRent *classes.*

The attributes should be similar to those documented in the textbook. The student would be expected to come up with standard get/set methods, such as GetStaffSalary, PutStaffSalary, and then methods for registering new properties for rent, new owners, new clients, appointments for viewings, and so on.