SOLUTIONS TO REVIEW QUESTIONS

AND EXERCISES

FOR PART 2 – THE RELATIONAL MODEL AND LANGUAGES (CHAPTERS 4 – 9)

Solutions to Review Questions and Exercises

Chapter 4 The Relational Model

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:

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

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.

Exercises

The following tables form part of a database held in a relational DBMS:-

Hotel	(hotelNo, hotelName, city)
Room	(roomNo, hotelNo, type, price)
Booking	(hotelNo, guestNo, dateFrom, dateTo, roomNo)
Guest	(guestNo, guestName, guestAddress)

where Hotel contains hotel details and hotelNo is the primary key;
Room contains room details for each hotel and (roomNo, hotelNo) forms the primary key;
Booking contains details of the bookings and (hotelNo, guestNo, dateFrom) forms the primary key;
and Guest contains guest details and guestNo is the primary key.

4.8 *Identify the foreign keys in this schema. Explain how the entity and referential integrity rules apply to these relations.*

For each relation, the primary key must not contain any nulls.

Room is related to Hotel through the attribute hotelNo. Therefore, the hotelNo in Room should either be null or contain the number of an existing hotel in the Hotel relation. In this case study, it would probably be unacceptable to have a hotelNo in Room with a null value.

Booking is related to Hotel through the attribute hotelNo. Therefore, the hotelNo in Booking should either be null or contain the number of an existing hotel in the Hotel relation. However, because hotelNo is also part of the primary key, a null value for this attribute would be unacceptable. Similarly for guestNo. Booking is also related to Room through the attribute roomNo.

4.9 *Produce some sample tables for these relations that observe the relational integrity rules. Suggest some general constraints that would be appropriate for this schema.*

Student should provide some sample tables, observing entity and referential integrity. In particular, ensure the uniqueness for the composite primary keys of the Room and Booking tables.

Some general constraints may be:

- There can be no two bookings for the same room in the same hotel on the same day.
- For the Booking relation, dateFrom must be before dateTo.
- Room price must be greater than 0 and less than £200.

4.10 *Analyze the RDBMSs that you are currently using. Determine the support the system provides for primary keys, alternate keys, foreign keys, relational integrity, and views.*

This is a small student project, the result of which is dependent on the system analyzed.

4.11 *Implement the above schema in one of the RDBMSs you currently use. Implement, where possible, the primary, alternate, and foreign keys, and appropriate relational integrity constraints.*

This is a small student project, the result of which is dependent on the RDBMS used. Ensure that keys have been implemented, and that relationships have been implemented if the RDBMS supports this.

Chapter 5 Relational Algebra and Relational Calculus

Review Questions

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:

  $T_1 = \Pi_C(R)$
  $T_2 = \Pi_C((S \text{ x } T_1) - R)$
  $T = T_1 - T_2$

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	$R ⟕ S$	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	$R \rhd_F S$	Produces a relation that contains the tuples of R that participate in the join of R with S.

5.5 *Compare and contrast the tuple relational calculus with domain relational calculus. In particular, discuss the distinction between tuple and domain variables.*

In the tuple relational calculus, we use variables that range over tuples in a relation. In the domain relational calculus, we also use variables but in this case the variables take their values from *domains* of attributes rather than tuples of relations.

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.

Domain relational calculus

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

- $R(d_1, d_2, \ldots, d_n)$, where R is a relation of degree n and each d_i is a domain variable.
- $d_i \; \theta d_j$, where d_i and d_j are domain variables and θ is one of the comparison operators ($<$, $\leq$, $>$, $\geq$, $=$, $\neq$); the domains d_i and d_j must have members that can be compared by θ.
- $d_i \; \theta c$, where d_i is a domain variable, c is a constant from the domain of d_i, and θ is one of the comparison operators.

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.

Exercises

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

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) $Guest ⟕ (\sigma_{dateTo \geq \text{‘1-Jan-2007’}} (Booking))$

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

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

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

5.9 *Provide the equivalent tuple relational calculus and domain relational calculus expressions for each of the relational algebra queries given in Exercise 4.8.*

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

TRC: $\{R.hotelNo \mid Room(R) \wedge R.price > 50\}$

DRC: $\{hotelNo \mid (\exists rNo, typ, prce) (Room (rNo, hotelNo, typ, prce) \wedge prce > 50)\}$

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

TRC: $\{H, R \mid Hotel(H) \wedge (\exists R) (Room(R) \wedge (H.hotelNo = R.hotelNo))\}$

DRC: $\{hNo, hName, cty, rNo, hNo1, typ, prce \mid (Hotel(hNo, hName, cty) \wedge Room(rNo, hNo1, typ, prce) \wedge (hNo = hNo1))\}$

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

TRC: $\{\text{H.hotelName} \mid \text{Hotel(H)} \land (\exists \text{R})\ (\text{Room(R)} \land (\text{H.hotelNo} = \text{R.hotelNo}) \land$
$(\text{R.price} > 50))\}$

DRC: $\{\text{hotelName} \mid (\exists \text{hNo, cty, rNo, hNo1, typ, prce})$
$(\text{Hotel(hNo, hotelName, cty)} \land \text{Room(rNo, hNo1, typ, prce)} \land (\text{hNo} = \text{hNo1}) \land$
$(\text{prce} > 50))\}$

d) $\text{Guest} ⟕ (\sigma_{\text{dateTo} \geq \text{‘1-Jan-2007’}}\ (\text{Booking}))$

TRC: $\{\text{G.guestNo, G.guestName, G.guestAddress, B.hotelNo, B.dateFrom,}$
$\text{B.dateTo, B.roomNo} \mid \text{Guest(G)} \lor (\exists \text{B})\ (\text{Booking(B)} \land$
$(\text{G.guestNo} = \text{B.guestNo}) \land (\text{B.dateTo} > \text{‘1-Jan-2007’}))\}$

DRC: $\{\text{guestNo, guestName, guestAddress, hotelNo, dateFrom, dateTo, roomNo} \mid$
$(\exists \text{gNo1})\ (\text{Guest(guestNo, guestName, guestAddress)} \lor$
$(\text{Booking(hotelNo, gNo1, dateFrom, dateTo, roomNo)} \land$
$(\text{guestNo} = \text{gNo1}) \land (\text{dateTo} \geq \text{‘1-Jan-2007’})))\}$

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

TRC: $\{\text{H.hotelNo, H.hotelName, H.city} \mid \text{Hotel(H)} \land (\exists \text{R})\ (\text{Room(R)} \land$
$(\text{H.hotelNo} = \text{R.hotelNo}) \land (\text{R.price} > 50))\}$

DRC: $\{\text{hotelNo, hotelName, city} \mid (\exists \text{rNo, hNo1, typ, prce})$
$(\text{Hotel(hotelNo, hotelName, city)} \land \text{Room(rNo, hNo1, typ, prce)} \land$
$(\text{hotelNo} = \text{hNo1}) \land (\text{prce} > 50))\}$

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

TRC: $\{\text{G.guestName} \mid \text{Guest(G)} \land (\sim (\exists \text{H})\ (\text{Hotel(H)} \land$
$(\text{H.city} = \text{‘London’}) \land (\sim(\exists \text{B})\ (\text{Booking(B)} \land$
$\text{G.guestNo} = \text{B.guestNo} \land \text{H.hotelNo} = \text{B.hotelNo}))))\}$

DRC: $\{\text{guestName} \mid (\exists \text{gNo, gName, gAddress, hNo, gNo1, dFrom, dTo, rNo,}$
$\text{hName, cty, hNo1, typ, prce})\ (\sim(\text{Hotel(hNo, hName, cty)} \land$
$(\text{cty} = \text{‘London’}) \land \text{Guest(gNo, gName, gAddress)} \land$
$\text{Booking(hNo1, gNo1, dFrom, dTo, rNo)} \land$
$(\text{gNo} = \text{gNo1}) \land (\text{hNo} = \text{hNo1})))\}$

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

(a) $\{H.hotelName \mid Hotel(H) \wedge H.city = \text{'London'}\}$

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

(b) $\{H.hotelName \mid Hotel(H) \wedge (\exists R)\ (Room(R) \wedge H.hotelNo = R.hotelNo \wedge R.price > 50)\}$

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

(c) $\{H.hotelName \mid Hotel(H) \wedge (\exists B)\ (\exists G)\ (Booking(B) \wedge Guest(G) \wedge H.hotelNo = B.hotelNo \wedge B.guestNo = G.guestNo \wedge G.guestName = \text{'John Smith'})\}$

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

(d) $\{H.hotelName, G.guestName, B1.dateFrom, B2.dateFrom \mid Hotel(H) \wedge Guest(G) \wedge Booking(B1) \wedge Booking(B2) \wedge H.hotelNo = B1.hotelNo \wedge G.guestNo = B1.guestNo \wedge B2.hotelNo = B1.hotelNo \wedge B2.guestNo = B1.guestNo \wedge B2.dateFrom \neq B1.dateFrom\}$

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.

5.11 *Provide the equivalent domain relational calculus and relational algebra expressions for each of the tuple relational calculus expressions given in Exercise 4.10.*

(a) $\{H.hotelName \mid Hotel(H) \wedge H.city = \text{'London'}\}$

DRC: $\{hotelName \mid (\exists hNo, cty)\ (Hotel(hNo, hotelName, cty) \wedge cty = \text{'London'})\}$

RA: $\Pi_{hotelName}\ (\sigma_{city = \text{'London'}}\ (Hotel)\ )$

(b) $\{H.hotelName \mid Hotel(H) \wedge (\exists R)\ (Room(R) \wedge H.hotelNo = R.hotelNo \wedge R.price > 50)\}$

DRC: $\{hotelName \mid (\exists hNo, cty, rNo, hNo1, typ, prce)\ (Hotel(hNo, hotelName, cty) \wedge Room(rNo, hNo1, typ, prce) \wedge (hNo = hNo1) \wedge (prce > 50))\ \}$

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

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

DRC: {hotelName | (∃hNo, cty, gNo, gName, gAddress, hNo1, gNo1, dFrom, dTo, rNo) (Hotel(hNo, hotelName, cty) ∧
Guest(gNo, gName, gAddress) ∧
Booking(hNo1, gNo1, dFrom, dTo, rNo) ∧ (gNo = gNo1) ∧
(hNo = hNo1) ∧ (gName = 'John Smith'))}

RA: $\Pi_{\text{hotelName}}$ ($\sigma_{\text{guestName = 'John Smith'}}$ (Guest) $\bowtie_{\text{Guest.guestNo = guestNo}}$ (
Booking $\bowtie_{\text{.Booking.hotelNo = Hotel.hotelNo}}$ Hotel))

(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}

DRC: {hotelName, guestName, dateFrom1, dateFrom2 | (∃hNo, cty,
gNo, gAddress, hNo1, gNo1, dTo1, rNo1, hNo2, gNo2, dTo2, rNo2)
(Hotel(hNo, hotelName, cty) ∧
Guest(gNo, guestName, gAddress) ∧
Booking(hNo1, gNo1, dateFrom1, dTo1, rNo1) ∧
Booking(hNo2, gNo2, dateFrom2, dTo2, rNo2) ∧
(hNo = hNo1) ∧ (gNo = gNo1) ∧ (hNo2 = hNo1) ∧ (gNo2 = gNo1) ∧
(dateFrom1 ≠ dateFrom2))}

RA: Booking2(hotelNo, guestNo, dateFrom2, dateTo2, roomNo2) ←
$\Pi_{\text{hotelNo, guestNo, dateFrom, dateTo, roomNo}}$ (Booking)

$\Pi_{\text{hotelName, guestName, dateFrom, dateFrom2}}$ (Hotel $\bowtie_{\text{Hotel.hotelNo = hotelNo}}$
(Guest $\bowtie_{\text{Guest.guestNo = guestNo}}$ (Booking $\bowtie_{\text{Booking.hotelNo = Booking2.hotelNo ∧ Booking.guestNo = Booking2.guestNo ∧ dateFrom ≠ dateFrom2}}$ Booking2)))

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: $\{H \mid \text{Hotel}(H)\}$

DRC: $\{\text{hotelNo, hotelName, city} \mid \text{Hotel(hotelNo, hotelName, city)}\}$

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

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

TRC: $\{R \mid \text{Room}(R) \wedge R.\text{type} = \text{‘S’} \wedge R.\text{price} < 20\}$

DRC: $\{\text{roomNo, hotelNo, type, price} \mid (\text{Room(roomNo, hotelNo, type, price)} \wedge$
$\text{type} = \text{‘S’} \wedge \text{price} < 20)\}$

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

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

TRC: $\{G.\text{guestName}, G.\text{guestAddress} \mid \text{Guest}(G)\}$

DRC: $\{\text{guestName, guestAddress} \mid (\exists \text{guestNo})$
$(\text{Guest(guestNo, guestName, guestAddress)})\}$

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

RA: $\Pi_{\text{price, type}}(\text{Room} \bowtie_{\text{hotelNo}} (\sigma_{\text{hotelName} = \text{‘Grosvenor Hotel’}}(\text{Hotel})))$

TRC: $\{R.\text{price}, R.\text{type} \mid \text{Room}(R) \wedge (\exists H) (\text{Hotel}(H) \wedge (\square R.\text{hotelNo} = H.\text{hotelNo}) \wedge$
$(H.\text{hotelName} = \text{‘Grosvenor Hotel’}))\}$

DRC: $\{\text{price, type} \mid (\exists \text{roomNo, hotelNo, hotelNo1, hotelName, city})$
$(\text{Room(roomNo, hotelNo, type, price)} \wedge \text{Hotel(hotelNo1, hotelName, city)} \wedge$
$(\text{hotelNo} = \text{hotelNo1}) \wedge (\text{hotelName} = \text{‘Grosvenor Hotel’}))\}$

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

RA: $\text{Guest} \bowtie_{\text{guestNo}} (\sigma_{\text{dateFrom} \leq \text{'01-01-15'} \wedge \text{dateTo} \geq \text{'01-01-15'}} ($

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

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

TRC: $\{G \mid \text{Guest}(G) \wedge ((\exists B)(\exists H)\ (\text{Booking}(B) \wedge \text{Hotel}(H) \wedge (B.\text{dateFrom} \leq \text{'01-01-15'}) \wedge$

$(B.\text{dateTo} \geq \text{'01-01-15'}) \wedge (B.\text{guestNo} = G.\text{guestNo}) \wedge$

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

DRC: $\{\text{guestNo, guestName, guestAddress} \mid (\exists \text{hotelNo, guestNo1, dateFrom, dateTo,}$

$\text{hotelNo1, hotelName, city})$

$(\text{Guest}(\text{guestNo, guestName, guestAddress}) \wedge$

$\text{Booking}(\text{hotelNo, guestNo1, dateFrom, dateTo}) \wedge$

$\text{Hotel}(\text{hotelNo1, hotelName, city}) \wedge (\text{guestNo} = \text{guestNo1}) \wedge$

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

$(\text{hotelNo} = \text{hotelNo1}) \wedge (\text{hotelName} = \text{'Grosvenor Hotel'}))\}$

(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: $(\text{Room} \bowtie_{\text{hotelNo}} (\sigma_{\text{hotelName} = \text{'Grosvenor Hotel'}}(\text{Hotel}))) ⟕$ // Outer Join

$\Pi_{\text{guestName, hotelNo, roomNo}}($

$(\text{Guest} \bowtie_{\text{guestNo}} (\sigma_{\text{dateFrom} \leq \text{'01-01-15'} \wedge \text{dateTo} \geq \text{'01-01-15'}} ($

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

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

TRC: $\{R, G.\text{guestName} \mid (\text{Room}(R) \wedge (\exists H)(\text{Hotel}(H) \wedge$

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

$(\text{Guest}(G) \wedge ((\exists B)(\exists H)\ (\text{Booking}(B) \wedge \text{Hotel}(H) \wedge$

$(G.\text{guestNo} = B.\text{guestNo}) \wedge (B.\text{hotelNo} = H.\text{hotelNo}) \wedge$

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

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

DRC: $\{roomNo, hotelNo, type, price, guestName \mid$
$(\exists hNo, hName, city, hNo1, gNo1, dFrom, dTo, rNo)$
$(Room(roomNo, hotelNo, type, price) \wedge Hotel(hNo1, hName, city) \wedge$
$(hotelNo = hNo1) \wedge (hName = \text{'Grosvenor Hotel'})) \vee$
$(Guest(guestNo, guestName, guestAddress) \wedge Hotel(hNo, hName, city) \wedge$
$Booking(hNo1, gNo1, dFrom, dTo, rNo) \wedge$
$(guestNo = gNo1) \wedge (hNo1 = hNo) \wedge (hName = \text{'Grosvenor Hotel'}) \wedge$
$(dFrom \leq \text{'01-01-15'} \wedge dTo \geq \text{'01-01-15'})))\}$

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

RA: $\Pi_{guestNo,\ guestName,\ guestAddress}(Guest \bowtie_{guestNo} (\sigma_{dateFrom \leq \text{'01-01-15'} \wedge dateTo \geq \text{'01-01-15'}} ($
$Booking \bowtie_{hotelNo} (\sigma_{hotelName=\text{'Grosvenor Hotel'}}(Hotel)))))$
(substitute '01-01-15' for today's date).

TRC: $\{G \mid Guest(G) \wedge ((\exists B) (\exists H) (Booking(B) \wedge Hotel(H) \wedge (B.guestNo = G.guestNo) \wedge$
$(B.hotelNo = H.hotelNo) \wedge (H.hotelName = \text{'Grosvenor Hotel'}) \wedge$
$(B.dateFrom \leq \text{'01-01-15'} \wedge B.dateTo \geq \text{'01-01-15'})))\}$

DRC: $\{guestNo, guestName, guestAddress \mid$
$((\exists hNo, gNo, dFrom, dTo, rNo, hNo1, hName, city)$
$(Guest(guestNo, guestName, guestAddress) \wedge$
$Booking(hNo, gNo, dFrom, dTo, rNo) \wedge Hotel(hNo1, hName, city) \wedge$
$(guestNo = gNo) \wedge (hNo = hNo1) \wedge (hName = \text{'Grosvenor Hotel'}) \wedge$
$(dFrom \leq \text{'01-01-15'} \wedge dTo \geq \text{'01-01-15'})))\}$

5.13 *Using relational algebra, create a view of all rooms in the Grosvenor Hotel, excluding price details. What would be the advantages of this view?*

$\Pi_{roomNo,\ hotelNo,\ type}(Room \bowtie_{hotelNo} (\sigma_{hotelName=\text{'Grosvenor Hotel'}} (Hotel)))$

Security - hides the price details from people who should not see it.
Reduced complexity - a query against this view is simpler than a query against the two underlying base relations.

5.14 *List all employees.*

RA: Employee

TRC: {E | Employee(E)}

DRC: {empNo, fName, lName, address, DOB, sex, position, deptNo | Employee(empNo, fName, lName, address, DOB, sex, position, deptNo)}

5.15 *List all the details of employees who are female.*

RA: $\sigma_{\text{sex='F'}}$(Employee)

TRC: {E | Employee(E) ∧ E.sex = ‘F’}

DRC: {empNo, fName, lName, address, DOB, sex, position, deptNo | Employee(empNo, fName, lName, address, DOB, sex, position, deptNo) ∧ (sex=’F’)}

5.16 *List the names and addresses of all employees who are managers.*

RA: $\Pi_{\text{fName, lName,address}}$(Employee $\sigma_{\text{position='manager'}}$(Employee))

TRC: {E.fName, E.lName, E.address | Employee(E) ∧ (E.position = ‘Manager’)}

DRC: {fName, lName, address | Employee(empNo, fName, lName, address, DOB, sex, position, deptNo) ∧ (position=’manager’)}

5.17 *Produce a list of the names and addresses of all employees who work for the IT department.*

RA: $\Pi_{\text{fName, lName,address}}$(Employee $\bowtie_{Employee.deptNo=Dept.deptNo}(\sigma_{deptName='IT'}(Dept))$

TRC: {E.fName, E.lName, E.address | Employee(E) ∧ (∃D) (Dept(D) ∧ (E.deptNo = D.deptNo) ∧ (D.deptName = ‘IT’))}

DRC: {fName, lName, address | (∃empNo, DOB, sex, position, deptNo, deptNo1, deptName, mgrEmpNo) (Empoyee(empNo, fName, lName, address, DOB, sex, position, deptNo) ∧ Department (deptNo1, deptName, mgrEmpNo) ∧ (deptNo = deptNo1) ∧ (deptName = ‘IT’))}

5.18 *Produce a list of the names of all employees who work on the SCCS project.*

RA: $\Pi_{\text{fName, lName}}$ (Employee $\bowtie_{\text{empNo}}$ (WorksOn $_{\text{projNo}}$ ($\sigma_{\text{projName = 'SCCS'}}$(Project)))

TRC: {E.fName, E.lName | Employee(E) $\wedge$ ($\exists$W) ($\exists$P) (WorksOn(W) $\wedge$ Project(P) $\wedge$ (P.projName = 'SCCS') $\wedge$ (P.projNo = W.projNo) $\wedge$ (E.empNo = W.empNo))

DRC: {fName, lName | ($\exists$empNo, address, DOB, sex, position, deptNo, projNo, projName, deptNo1, empNo1, projNo1, dateWorked, hoursWorked) (Employee (empNo, fName, lName, address, DOB, sex, position, deptNo) $\wedge$ Project (projNo, projName, deptNo1) $\wedge$ WorksOn (empNo1, projNo1, dateWorked, hoursWorked) $\wedge$ (empNo = empNo1) $\wedge$ (projNo = projNo1) $\wedge$ (projName = 'SCCS'))}

5.19 *Produce a complete list of all managers who are due to retire this year, in alphabetical order of surname.*

Formulate the following queries in relational algebra.

5.20 *Find out how many employees are managed by "James Adam."*

$\Im_{\text{COUNT empNo}}$(Employee $\bowtie_{\text{Employee.deptNo = deptNo}}$(Department $\bowtie_{\text{Department.mgrEmpNo = Employee.empNo}}$($\sigma_{\text{fName = 'James'} \wedge \text{lName = 'Adam'}}$(Employee))))

5.21 *Produce a report of the total hours worked by each employee.*

$_{\text{empNo}}\Im_{\text{SUM hoursWorked}}$(WorksOn)

5.22 *For each project on which more than two employees worked, list the project number, project name, and the number of employees who work on that project.*

($\Pi_{\text{projName, projNo}}$(Project)) $\bowtie$

($\sigma_{\text{empCount > 2}}$($\rho_R$(projNo, empCount) $_{\text{projNo}}\Im_{\text{COUNT empNo}}$(WorksOn))))

5.23 *List the total number of employees in each department for those departments with more than 10 employees. Create an appropriate heading for the columns of the results table.*

($\sigma_{\text{empCount > 10}}$($\rho_R$(deptNo, empCount) $_{\text{deptNo}}\Im_{\text{COUNT empNo}}$(Employee)))))

The following tables form part of a Library database held in an RDBMS:

Book	(<u>ISBN</u>, title, edition, year)
BookCopy	(<u>copyNo</u>, ISBN, available)
Borrower	(<u>borrowerNo</u>, borrowerName, borrowerAddress)
BookLoan	(<u>copyNo</u>, <u>dateOut</u>, dateDue, borrowerNo)

where	Book	contains details of book titles in the library and the ISBN is the key.
	BookCopy	contains details of the individual copies of books in the library and copyNo is the key. ISBN is a foreign key identifying the book title.
	Borrower	contains details of library members who can borrow books and borrowerNo is the key.
	BookLoan	contains details of the book copies that are borrowed by library members and copyNo/dateOut forms the key. borrowerNo is a foreign key identifying the borrower.

Formulate the additional queries in relational algebra, tuple relational calculus, and domain relational calculus.

5.24 List all book titles.

RA: Π_{title}(Book)

TRC: {B.title | Book(B)}

DRC: {title | (∃ISBN, edn, yr) (Book(ISBN, title, edn, yr) }

5.25 List all borrower details.

RA: Borrower

TRC: {B | Borrower(B)}

DRC: {bNo, bName, bAddress | (Borrower(bNo, bName, bAddress) }

5.26 List all book titles published in the year 2012.

RA: $\Pi_{title}(\sigma_{year='2012'}$(Book))

TRC: {B.title | Book(B) $\wedge$ B.year='2012'}

DRC: {title | (∃ISBN, edn, yr) (Book(ISBN, title, edn, yr) $\wedge$ yr='2012'}

5.27 *List all copies of book titles that are available for borrowing.*

RA: $\Pi_{\text{copyNo, title}}(\text{Book} \bowtie_{\text{ISBN}} (\sigma_{\text{available='Y'}}(\text{BookCopy})))$

TRC: {BC.copyNo, B.title | Book(B) ∧ (∃BC) (BookCopy(BC) ∧ (B.ISBN = BC.ISBN) ∧ (BC.available='Y'))}

DRC: {copyNo, title | (∃ISBN, edn, yr, ISBN, avail) (Book(ISBN, title, edn, yr) ∧ BookCopy(copyNo, ISBN, avail) ∧ avail='Y')}

5.28 *List all copies of the book title "Lord of the Rings" that are available for borrowing.*

RA: $\Pi_{\text{copyNo}}(\sigma_{\text{title='Lord of the Rings'}}(\text{Book}) \bowtie_{\text{ISBN}} (\sigma_{\text{available='Y'}}(\text{BookCopy})))$

TRC: {BC.copyNo | BookCopy(BC) ∧ (∃B) (Book(B) ∧ (B.ISBN = BC.ISBN) ∧ (BC.available='Y') ∧ (B.title= 'Lord of the Rings'))}

DRC: {copyNo | (∃ISBN, edn, yr, ISBN, avail) (Book(ISBN, title, edn, yr) ∧ BookCopy(copyNo, ISBN, avail) ∧ avail='Y' ∧ title= 'Lord of the Rings')}

5.29 *List the names of borrowers who currently have the book title "Lord of the Rings" on loan.*

RA: $\Pi_{\text{borrowerName}}(((\sigma_{\text{title='Lord of the Rings'}}(\text{Book})) \bowtie_{\text{ISBN}} (\sigma_{\text{available='N'}}(\text{BookCopy}))) \bowtie_{\text{copyNo}} (\text{BookLoan}) \bowtie_{\text{borrowerNo}} (\text{Borrower}))$

TRC: {BW.borrowerName | Borrower(BW) ∧ (∃BL) (∃B) (∃BC) (Book(B) ∧ BookCopy(BC) ∧ BookLoan(BL) ∧ (BC.ISBN = B.ISBN) ∧ (BW.borrowerNo = BL.borrowerNo) ∧ (BL.copyNo = BC.copyNo) ∧ (BC.available='N') ∧ (B.title= 'Lord of the Rings'))}

DRC: {borrowerName | (∃ISBN, title, edn, yr, copyNo, avail, bNo, bAddress, dOut, dDue) (Book(ISBN, title, edn, yr) ∧ BookCopy(copyNo, ISBN, avail) ∧ Borrower(bNo, borrowerName, bAddress) ∧ BookLoan(copyNo, dOut, dDue, bNo) ∧ avail='N' ∧ title= 'Lord of the Rings')}

5.30 *List the names of borrowers with overdue books.*

RA: $\Pi_{\text{borrowerName}}(\text{Borrower} \bowtie_{\text{borrowerNo}} (\sigma_{\text{dateDue}>\text{‘today’s date’}}(\text{BookLoan})))$

TRC: $\{\text{BW.borrowerName} \mid \text{Borrower(BW)} \land (\exists \text{BL}) (\text{BookLoan(BL)} \land$
$(\text{BW.borrowerNo} = \text{BL.borrowerNo}) \land (\text{BL.dateDue} > \text{‘today’s date’}))\}$

DRC: $\{\text{borrowerName} \mid (\exists \text{bNo, bAddress, copyNo, dOut, dDue})$
$(\text{Borrower(bNo, borrowerName, bAddress)} \land \text{BookLoan(copyNo, dOut,}$
$\text{dDue, bNo)} \land \text{dDue}>\text{‘today’s date’})\}$

Formulate the following queries in relational algebra.

5.31 *How many copies of ISBN “0-321-52306-7” are there?*

$\Im_{\text{COUNT copyNo}}(\sigma_{\text{ISBN} = \text{‘0-321-52306-7’}} (\text{BookCopy}))$

5.32 *How many copies of ISBN “0-321-52306-7” are currently available?*

$\Im_{\text{COUNT copyNo}}(\sigma_{\text{available} = \text{‘Y’} \land \text{ISBN} = \text{‘0-321-52306-7’}} (\text{BookCopy}))$

5.33 *How many times has the book title with ISBN “0-321-52306-7” been borrowed?*

$\Im_{\text{COUNT copyNo}}((\sigma_{\text{ISBN} = \text{‘0-321-52306-7’}} (\text{BookCopy}) \bowtie_{\text{copyNo}} \text{BookLoan})$

5.34 *Produce a report of book titles that have been borrowed by “Peter Bloomfield”.*

$\Pi_{\text{title}}(\text{Book} \bowtie_{\text{ISBN}} ((\text{BookCopy} \bowtie_{\text{copyNo}} \text{BookLoan}) \bowtie_{\text{borrowerNo}}$
$(\sigma_{\text{borrowerName} = \text{‘Peter Bloomfield’}}(\text{Borrower}))))$

5.35 *For each book title with more than 3 copies, list the names of library members who have borrowed them.*

5.36 *Produce a report with the details of borrowers who currently have books overdue.*

$\Pi_{\text{borrowerName, borrowerAddress}}(\text{Borrower} \bowtie_{\text{borrowerNo}} (\sigma_{\text{dateDue}>\text{‘today’s date’}}(\text{BookLoan})))$

5.37 *Produce a report detailing how many times each book title has been borrowed.*

$_{\text{ISBN}}\Im_{\text{COUNT copyNo}}(\text{BookCopy} \bowtie_{\text{copyNo}} \text{BookLoan})$

5.38 *Analyze the RDBMSs that you are currently using. What types of relational languages does the system provide? For each of the languages provided, what are the equivalent operations for the eight relational algebra operations defined in Section 5.1?*

This is a small student project, the result of which is dependent on the system analyzed. However, it is likely that the supported languages will be based around SQL and QBE, in which case, the student should attempt to map the various SQL clauses to the algebra and calculus. See also Exercise 5.31.

Chapter 6 SQL: Data Manipulation

Review Questions

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.

Exercises

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;
```

General

6.29 *Investigate the SQL dialect on any DBMS that you are currently using. Determine the compliance of the DBMS with the ISO standard. Investigate the functionality of any extensions the DBMS supports. Are there any functions not supported?*

This is a small student project, the result of which is dependent on the dialect of SQL being used.

6.30 *Show that a query using the HAVING clause has an equivalent formulation without a HAVING clause.*

Hint: Allow the students to show that the restricted groups could have been restricted earlier with a WHERE clause.

6.31 *Show that SQL is relationally complete.*

Hint: Allow the students to show that each of the relational algebra operations can be expressed in SQL.

Case Study 2

For Exercises 6.32–6.40, use the Projects schema defined in the Exercises at the end of Chapter 5.

6.32 *List all employees in alphabetical order of surname, and then first name.*

```
SELECT * FROM Employee ORDER BY lName, fName;
```

6.33 *List all the details of employees who are female.*

SELECT * FROM Employee WHERE sex = 'female';

6.34 *List the names and addresses of all employees who are Managers.*

SELECT fName, lName, address FROM Employee WHERE position = 'Manager';

6.35 *Produce a list of the names and addresses of all employees who work for the IT department.*

SELECT fName, lName, address FROM Employee e, Department d

WHERE e.deptNo = d.deptNo AND d.deptName = 'IT';

6.36 *Produce a complete list of all managers who are due to retire this year, in alphabetical order of surname.*

6.37 *Find out how many employees are managed by 'James Adams'.*

SELECT COUNT(empNo) FROM Employee

WHERE deptNo = (SELECT deptNo FROM Employee WHERE fName = 'James' AND lName = 'Adams') AND (fName <> 'James' AND lName <> 'Adams');

6.38 *Produce a report of the total hours worked by each employee, arranged in order of department number and within department, alphabetically by employee surname.*

SELECT e.lName, e.fName, hoursWorked

FROM WorksOn w, Employee e, Department d

WHERE e.deptNo = d.deptNo AND e.empNo = w.empNo

ORDER BY d.deptNo, e.lName;

6.39 *For each project on which more than two employees worked, list the project number, project name, and the number of employees who work on that project.*

SELECT p.projNo, projName, COUNT(empNo)

FROM Project p, WorksOn w

WHERE p.projNo = w.projNo

GROUP BY p.projNo

HAVING COUNT(empNo) > 2;

6.40 *List the total number of employees in each department for those departments with more than 10 employees. Create an appropriate heading for the columns of the results table.*

SELECT COUNT(empNo) AS empCount, deptNo

FROM Employee

GROUP BY deptNo

```
HAVING COUNT(empNo) > 10;
```

Case Study 3

For Exercises 6.41–6.54, use the Library schema defined in the Exercises at the end of Chapter 5.

6.41 *List all book titles.*

```
SELECT title FROM Book;
```

6.42 *List all borrower details.*

```
SELECT * FROM Borrower;
```

6.43 *List all book titles published in the year 2012.*

```
SELECT title
FROM Book
WHERE year = '2012';
```

6.44 *List all copies of book titles that are available for borrowing.*

```
SELECT copyNo, title
FROM BookCopy bc, Book b
WHERE bc.ISBN = b.ISBN AND bc.available = 'Y';
```

6.45 *List all copies of the book title "Lord of the Rings" that are available for borrowing.*

```
SELECT copyNo
FROM BookCopy bc, Book b
WHERE bc.ISBN = b.ISBN AND bc.available = 'Y' AND title = 'Lord of the Rings';
```

6.46 *List the names of borrowers who currently have the book title "Lord of the Rings" on loan.*

```
SELECT borrowerName
FROM Borrower bw, Book b, BookCopy bc, BookLoan bl
WHERE bw.borrowerNo = bl.borrowerNo AND bl.copyNo = bc.copyNo
        AND bc.ISBN = b.ISBN AND bc.available = 'N' AND title = 'Lord of the Rings';
```

6.47 *List the names of borrowers with overdue books.*

```
SELECT borrowerName
From Borrower bw, BookLoan bl
WHERE bw.borrowerNo = bl.borrowerNo and dateDue > today’s date;
```

6.48 *How many copies of ISBN “0-321-52306-7” are there?*

```
SELECT COUNT(*)
FROM BookCopy
WHERE ISBN = ‘0-321-52306-7’;
```

6.49 *How many copies of ISBN “0-321-52306-7” are currently available?*

```
SELECT COUNT(copyNo)
FROM BookCopy
WHERE available = ‘Y’ AND ISBN = ‘0-321-52306-7’;
```

6.50 *How many times has the book with ISBN “0-321-52306-7” been borrowed?*

```
SELECT COUNT(*)
FROM BookCopy bc, BookLoan bl
WHERE bc.copyNo = bl.copyNo AND ISBN = ‘0-321-52306-7’;
```

6.51 *Produce a report of book titles that have been borrowed by “Peter Bloomfield”.*

```
SELECT DISTINCT title
FROM Borrower bw, Book b, BookCopy bc, BookLoan bl
WHERE bw.borrowerNo = bl.borrowerNo AND bl.copyNo = bc.copyNo
        AND bc.ISBN = b.ISBN AND borrowerName = ‘Peter Bloomfield’;
```

6.52 *For each book title with more than 3 copies, list the names of library members who have borrowed them.*

```
SELECT title, borrowerName
FROM Borrower bw, Book b, BookCopy bc, BookLoan bl
WHERE bw.borrowerNo = bl.borrowerNo AND bl.copyNo = bc.copyNo
        AND bc.ISBN = b.ISBN AND EXISTS
        (SELECT ISBN, COUNT(bc1.copyNo)
        FROM BookCopy bc1
        WHERE bc1.ISBN = bc.ISBN
        GROUP BY bc1.ISBN
        HAVING COUNT(bc1.copyNo) > 3);
```

6.53 *Produce a report with the details of borrowers who currently have books overdue.*

```
SELECT borrowerName, borrowerAddress
From Borrower bw, BookLoan bl
WHERE bw.borrowerNo = bl.borrowerNo and dateDue > today’s date;
```

6.54 *Produce a report detailing how many times each book title has been borrowed.*

```
SELECT ISBN, COUNT(*)
FROM BookCopy bc, BookLoan bl
WHERE bc.copyNo = bl.copyNo
GROUP BY ISBN;
```

Chapter 7 SQL: Data Definition

Review Questions

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.

Exercises

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;
```

7.19 *Assume that we also have a table for suppliers:*

Supplier (supplierNo, partNo, price)

and a view SupplierParts, which contains the distinct part numbers that are supplied by at least one supplier:

```
CREATE VIEW SupplierParts (partNo)
AS      SELECT DISTINCT partNo
        FROM Supplier s, Part p
        WHERE s.partNo = p.partNo;
```

Discuss how you would maintain this as a materialized view and under what circumstances you would be able to maintain the view without having to access the underlying base tables Part and Supplier.

7.20 *Investigate the SQL dialect on any DBMS that you are currently using. Determine the system's compliance with the DDL statements in the ISO standard. Investigate the functionality of any extensions the DBMS supports. Are there any functions not supported?*

This is a small student project, the result of which is dependent on the dialect of SQL being used.

7.21 *Create the DreamHome rental database schema defined in Section 4.2.6 and insert the tuples shown in Figure 4.3.*

This is a small student project, the result of which is dependent on the DBMS being used.

7.22 *Using the schema you have created above, run the SQL queries given in the examples from Chapter 6.*

This is a small student project, the result of which is dependent on the DBMS being used.

7.23 *Create the schema for the Hotel schema given at the start of the exercises for Chapter 4 and insert some sample tuples. Now run the SQL queries that you produced for Exercises 6.7 – 6.28.*

This is a small student project, the result of which is dependent on the DBMS being used.

Case Study 2

7.24 *Create the Projects schema using the integrity enhancement features of SQL with the following constraints:*
(a) sex must be one of the single characters 'M' or 'F'.
(b) position must be one of 'Manager', 'Team Leader', 'Analyst', or 'Software Developer'.
(c) hoursWorked must be an integer value between 0 and 40.

```
CREATE SCHMEA Projects;
CREATE DOMAIN TableKey AS CHAR(5);
CREATE DOMAIN Name AS VARCHAR(20);
CREATE DOMAIN Address AS VARCHAR(40);
CREATE DOMAIN EmpDate AS DATE;
CREATE DOMAIN Sex AS CHAR(1) CHECK(VALUE IN ('M', 'F'));
CREATE DOMAIN Position AS VARCHAR2(20) CHECK(VALUE IN (Manager', 'Team
Leader', 'Analyst', 'Software Developer'));
CREATE DOMAIN DeptNo AS CHAR(5) CHECK (VALUE IN (SELECT deptNo FROM
Department));
CREATE DOMAIN EmpNo AS CHAR(5) CHECK (VALUE IN (SELECT empNo FROM
Employee));
CREATE DOMAIN hoursWorked AS SMALLINT CHECK(VALUE BETWEEN 0 AND
40);

CREATE TABLE Employee (
        empNo           TableKey        NOT NULL,
        fName           Name    NOT NULL,
        lName           Name    NOT NULL,
        address         Address,
        DOB             EmpDate         NOT NULL,
        sex             Sex     NOT NULL,
        position        Position NOT NULL,
        deptNo          DeptNo  NOT NULL,
        PRIMARY KEY  (empNo)
        FOREIGN KEY  (deptNo) REFERENCES Department ON DELETE NO ACTION
ON UPDATE NO ACTION);

CREATE TABLE Department (
        deptNo          TableKey        NOT NULL,
        deptName        Name    NOT NULL,
        mgrEmpNo        EmpNo   NOT NULL,
        PRIMARY KEY  (deptNo),
        FOREIGN KEY  (empNo)            REFERENCES Employee ON DELETE NO
ACTION, ON UPDATE NO ACTION);

CREATE TABLE Project (
        projNo          TableKey        NOT NULL,
        projName                Name    NOT NULL,
        deptNo          DeptNo  NOT NULL,
        PRIMARY KEY  (projNo),
        FOREIGN KEY  (deptNo) REFERENCES Department ON DELETE NO ACTION
ON UPDATE NO ACTION);

CREATE TABLE WorksOn (
        empNo           EmpNo   NOT NULL,
        projNo          ProjNo  NOT NULL,
        dateWorked      EmpDate         NOT NULL,
        hoursWorked      HoursWorked      NOT NULL,
        PRIMARY KEY  (empNo, projNo, dateWorked),
        FOREIGN KEY  (empNo) REFERENCES Employee ON DELETE NO ACTION,
ON UPDATE NO ACTION,
        FOREIGN KEY  (projNo) REFERENCES Project ON DELETE NO ACTION ON
UPDATE NO ACTION);
```

7.25 *Create a view consisting of the Employee and Department tables without the address, DOB, and sex attributes.*

```
CREATE VIEW EmpDept (empNo, fName, lName, position, deptNo, deptName,
mgrEmpNo)
AS SELECT e.empNo, e.fName, e.lName, e.position, e.deptNo, d.deptName, d.mgrEmpNo
FROM Employee e, Department d
WHERE e.deptNo = d.deptNo;
```

7.26 *Create a view consisting of the attributes empNo, fName, lName, projName, and hoursWorked attributes.*

```
CREATE VIEW EmpWorksOn (empNo, fName, lName, projName, hoursWorked)
AS SELECT e.empNo, e.fName, e.lName, p.projName, w.hoursWorked
FROM Employee e, Project p, WorksOn w
WHERE e.empNo = w.empNo
AND p.projNo = w.projNo;
```

7.27 *Consider the following view defined on the Projects schema:*
CREATE VIEW EmpProject(empNo, projNo, totalHours)
AS SELECT w.empNo, w.projNo, SUM(hoursWorked)
FROM Employee e, Project p, WorksOn w
WHERE e.empNo _ w.empNo AND p.projNo _ w.projNo
GROUP BY w.empNo, w.projNo;
*(a) SELECT **
FROM EmpProject;
(b) SELECT projNo
FROM EmpProject
WHERE projNo _ 'SCCS';
(c) SELECT COUNT(projNo)
FROM EmpProject
WHERE empNo _ 'E1';
(d) SELECT empNo, totalHours
FROM EmpProject
GROUP BY empNo;

General

7.28 *Consider the following table:*

Part (<u>partNo</u>, <u>contract</u>, partCost)

which represents the cost negotiated under each contract for a part (a part may have a different price under each contract). Now consider the following view ExpensiveParts, which contains the distinct part numbers for parts that cost more than £1000:

```
CREATE VIEW ExpensiveParts (partNo)
```

```
AS      SELECT DISTINCT partNo
        FROM Part
        WHERE partCost > 1000;
```

Discuss how you would maintain this as a materialized view and under what circumstances you would be able to maintain the view without having to access the underlying base table Part.

If a row is inserted into Part with a partCost less than or equal to £1000, the view would not have to be updated. If a partNo is inserted into Part that is already in the view, no new record has to be inserted into the view (because of the DISTINCT keyword). Similarly for update. If a partNo is deleted from Part have to access the underlying base table to check if there is another partNo with same value, to determine whether row should be deleted from the view.

Chapter 8 Advanced SQL

Review Questions

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 toeach 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.

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;
```

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;
```

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;
```

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;
```

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 INSET 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;
```

8.13 *Analyze the RDBMS that you are currently using and determine the support the system provides for SQL programming constructs, database triggers, and recursive queries. Document the differences between each system and the SQL standard.*

This is a small student project, the result of which is dependent on the system analyzed.

Chapter 9 Object-Relational DBMSs

Review Questions

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.

9.11 What are the security problems associated with the introduction of user-defined methods and suggest some solutions to these problems?

If the user-defined function (UDF) causes some fatal runtime error, then if the UDF code is linked into the ORDBMS server, the error may have the consequential effect of crashing the server. Clearly, this is something that the ORDBMS has to protect against. One approach is to have all UDFs written in an interpreted language, such as SQL or Java. However, we have already seen that SQL:2011 allows an external routine, written in a high-level programming language such as ‘C’/C++, to be invoked as a UDF. In this case, an alternative approach is to run the UDF in a different address space to the ORDBMS server, and for the UDF and server to communicate using some form of interprocess communication (IPC). In this case, if the UDF causes a fatal runtime error, the only process affected is that of the UDF..

Exercises

9.12 Investigate one of the advanced database applications discussed in Section 9.1, or a similar one that handles complex, interrelated data. In particular, examine its functionality, and the data types and operations it uses. Map the data types and operations to the object-oriented concepts discussed in Appendix K.

This is a small student project, the result of which is dependent on the application investigated. However, expect the student to cover not just standard concepts such as objects, attributes, classes, superclasses, but also concepts such as overloading, complex objects, dynamic binding.

9.13 Analyze one of the relational DBMSs that you currently use. Discuss the object-oriented features provided by the system. What additional functionality do these features provide?

This is a small student project, the result of which is dependent on the system analyzed.

9.14 Analyze the RDBMSs that you are currently using. Discuss the object-oriented facilities provided by the system. What additional functionality do these facilities provide?

This is a small student project, the result of which is dependent on the system analyzed.

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)));
```

9.16 *Create SQL:2011 statements for the queries given in Chapter 6, Exercise 6.7 - 6.28.*

Depends on the solution to the previous question. For example, using the above schema, the solution to 6.16 would be:

```
SELECT price, type
FROM Room r
WHERE r–>hotelID–>hotelName = ‘Grosvenor Hotel’;
```

9.17 *Create an insert trigger that sets up a mailshot table recording the names and addresses of all guests who have stayed at the hotel during the days before and after New Year for the past two years.*

The aim of this question is to show the difficulty of creating a trigger that does not necessarily need to be tied to a modification to a table.

```
CREATE TRIGGER InsertMailshotTable
    AFTER INSERT ON Booking
    BEGIN
    INSERT INTO Mailshot
      (SELECT g.guestNo, g.guestName, g.guestAddress
      FROM Guest g, Booking b1, Booking b2
      WHERE g.guestNo = b1.guestNo AND b1.guestNo = b2.guestNo AND
            ((b1.dateFrom<=DATE’2014-12-31’ AND b1.dateTo>=DATE’2014-12-31’) OR
            (b1.dateFrom >= DATE’2014-12-31’ AND b1.dateFrom <= DATE’2015-01-02’))
            AND
            ((b1.dateFrom <= DATE’2013-12-31’ AND b1.dateTo>=DATE’2013-12-31’) OR
            (b1.dateFrom >= DATE’2013-12-31’ AND b1.dateFrom <= DATE’2014-01-02’))
            AND
            NOT EXISTS (SELECT * FROM Mailshot m
                                WHERE m.guestNo = g.guestNo);
    END;
```

9.18 *Repeat Exercise 9.16 for the multinational engineering case study in the Exercises of Chapter 24.*

Look for solution that uses the features of SQL:2011 such as types with inheritance, and object references (see also the solution to Exercise 9.22 below).

9.19 *Create an object-relational schema for the DreamHome case study documented in Appendix A. Add user-defined functions that you consider appropriate. Implement the queries listed in Appendix A using SQL:2011.*

Look for solution that uses the features of SQL:2011 such as types with inheritance, SETs, and object references (see also the solution to Exercise 9.22 below).

9.20 *Create an object-relational schema for the University Accommodation Office case study documented in Appendix B.1. Add user-defined functions that you consider appropriate.*

Look for solution that uses the features of SQL:2011 such as types with inheritance, SETs, and object references (see also the solution to Exercise 9.22 below).

9.21 *Create an object-relational schema for the EasyDrive School of Motoring case study documented in Appendix B.2. Add user-defined functions that you consider appropriate.*

Look for solution that uses the features of SQL:2011 such as types with inheritance, SETs, and object references (see also the solution to Exercise 9.22 below).

9.22 *Create an object-relational schema for the Wellmeadows case study documented in Appendix B.3. Add user-defined functions that you consider appropriate.*

Partial solution:

```
CREATE TYPE WardType AS (
        wardNo          VARCHAR(4)    NOT NULL,
        wardName        VARCHAR(20)   NOT NULL,
        location        VARCHAR(20)   NOT NULL,
        totalBeds       INTEGER,
        telExtn         VARCHAR(4)    NOT NULL,
        consultant      REF(ConsultantType),
        FUNCTION assignChargeNurse(W WardType RESULT, N NurseType)
                        RETURNS WardType
                ...
                RETURN;
        END,
);

CREATE TYPE QualificationType AS (
        qType           VARCHAR(5)    NOT NULL,
        qDate           DATE,
        institution     VARCHAR(30),
);

CREATE TYPE WorkExperienceType AS (
        orgName         VARCHAR       NOT NULL,
        sDate           DATE,
        fDate           DATE,
        position        VARCHAR(10)
);

CREATE TYPE NameType AS (
        fName           VARCHAR(15)   NOT NULL,
        lName           VARCHAR(15)   NOT NULL
);

CREATE TYPE StaffType AS (
  PRIVATE
        DOB DATE_CHECK(DOB < CURRENT_DATE),
  PUBLIC
        staffNo         VARCHAR(5)    NOT NULL,
        name            NameType,
        sAddress        VARCHAR(50)   NOT NULL,
        telNo           VARCHAR(13)   NOT NULL,
        sex             CHAR          NOT NULL,
        NIN             VARCHAR(9)    NOT NULL,
        position        VARCHAR(10)   NOT NULL,
        salary          DECIMAL(6,2)  NOT NULL,
        sScale          INTEGER         NOT NULL,
        weekHrs         INTEGER         NOT NULL,
        contType        CHAR            NOT NULL,
        typePay         CHAR            NOT NULL,
        qualification   SET(QualificationType),
        workExp         SET(WorkExperienceType)
        FUNCTION getAge (P PersonType) RETURNS INTEGER
                        RETURN          /* code to calculate age from date_of_birth */
        END,
        FUNCTION setAge (P PersonType RESULT, DOB: DATE) RETURNS PersonType
                        RETURN          /* set date_of_birth */
        END
        ) NOT FINAL;

CREATE TYPE NurseType UNDER StaffType (
        FUNCTION makeRequisition(...);
        BEGIN
                ...
        END
);

CREATE TYPE ConsultantType UNDER Staff Type (
        FUNCTION cancelAppointment(...);
        BEGIN
                ...
        END
);

CREATE TABLE Ward OF WardType(
        oid     REF(WardType) VALUES ARE SYSTEM GENERATED,
        PRIMARY KEY(wardNo));
CREATE TABLE Nurse OF NurseType(
        oid     REF(NurseType) VALUES ARE SYSTEM GENERATED,
        PRIMARY KEY(staffNo));
CREATE TABLE Consultant OF ConsultantType(
        oid     REF(ConsultantType) VALUES ARE SYSTEM GENERATED,
        PRIMARY KEY(staffNo));
```

9.23 You have been asked by the Managing Director of DreamHome to investigate and prepare a report on the applicability of an object-relational DBMS for the organization. The report should compare the technology of the RDBMS with that of the ORDBMS, and should address the advantages and disadvantages of implementing an ORDBMS within the organization, and any perceived problem areas. The report should also consider the applicability of an object-oriented DBMS, and a comparison of the two types of systems for DreamHome should be included. Finally, the report should contain a fully justified set of conclusions on the applicability of the ORDBMS for DreamHome.

A well-presented report is expected. Justification must be given for any recommendations made.