Database Management System

Unit-2

Dharmendra Kumar(Associate Professor)
Department of Computer Science and Engineering
United College of Engineering and Research, Prayagraj

RELATIONAL DATA MODEL

1 Introduction

Relational data model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

2 Some concepts related with relational data model

Relation: A relation is a table with columns and rows.

Field: A column in a table is called the field of a relation.

Tuple: It is nothing but a single row of a table, which contains a single record.

Relation schema: A relation schema represents the name of the relation with its attributes. If $A_1, A_2, ..., A_n$ are attributes then R= $(A_1, A_2, ..., A_n)$ is a relation schema.
Example: A relation schema student with their attributes are like the followings:-
student=(rollNo, name,branch,contactNo).

Relation Instance: Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.

Attribute domain: The set of all possible values of a relation is said to be domain of an attribute.

Degree: The total number of attributes exist in the relation is called the degree of the relation.

Cardinality: Total number of rows present in the table.

Atomic values: A value is said to be atomic if it is not divisible.

Note: Domain of an attribute is said to be atomic if all its possible values are atomic i.e. not divisible.

3 Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.

- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

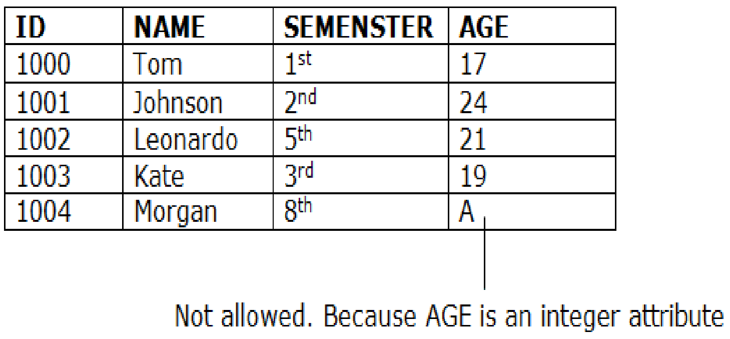
3.1 Types of Integrity Constraint

1. Domain Constraints
2. Entity Integrity Constraints
3. Referential Integrity Constraints
4. Key Constraints

3.1.1 Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example: Consider the following table



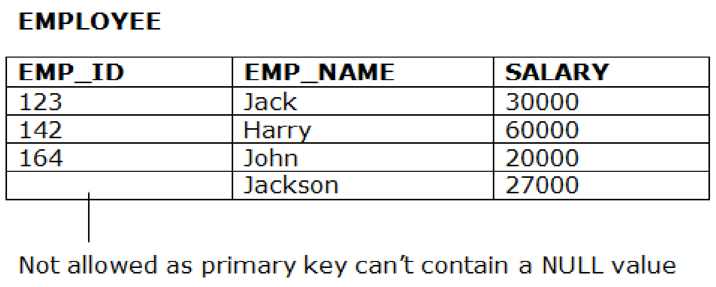
ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

3.1.2 Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example: Consider the following table



EMPLOYEE

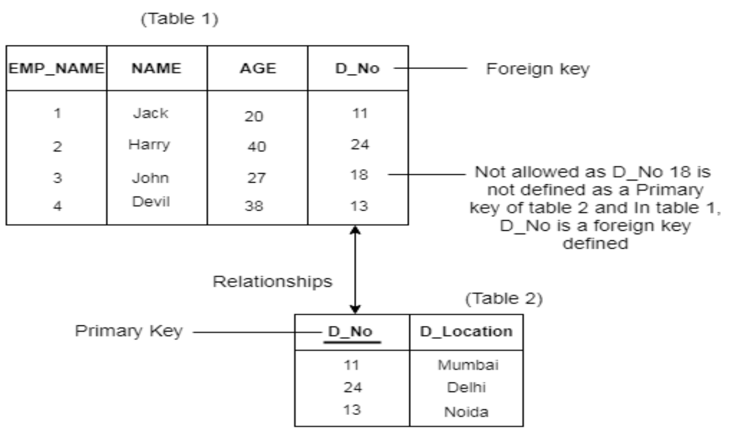
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3.1.3 Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example: Consider the following table



(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

(Table 2)

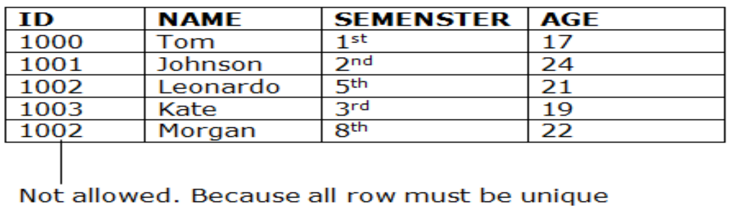
D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
Primary Key

3.1.4 Key constraints

- Keys in the entity set is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example: Consider the following table



ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

4 Foreign key

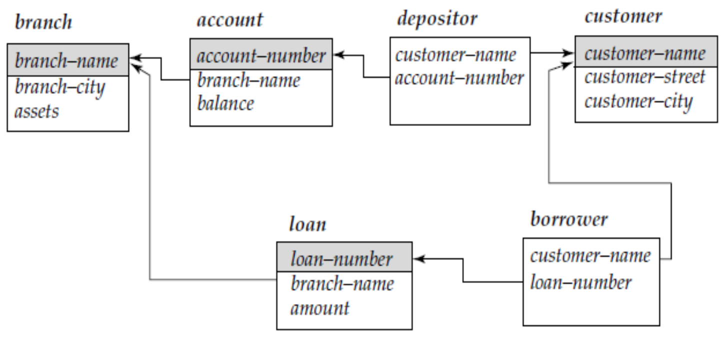
- Consider R and S are two tables. An attribute A of table R is said to be foreign key of R if A is the primary key in S.
- A foreign key is a column (or combination of columns) in a table whose values must match values of a column in some other table.
- FOREIGN KEY constraints enforce referential integrity, which essentially says that if column value A refers to column value B, then column value B must exist.

Example: Consider two tables Employee(ID, Name, Dept-ID) and Department(Dept-ID, Dept-name).
Here, attribute Dept-ID in Employee table is a foreign key because Dept-ID in Department table is a primary key.

5 Schema Diagram

- A database schema, along with primary key and foreign key dependencies, can be depicted pictorially by schema diagrams.
- Each relation appears as a box, with the attributes listed inside it and the relation name above it. If there are primary key attributes, a horizontal line crosses the box, with the primary key attributes listed above the line. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

Example: Following figure shows the schema diagram for our banking enterprise.



6 Query Languages

A query language is a language in which a user requests information from the database. Query languages can be categorized as either procedural or non-procedural.

In a procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired result.

In a non-procedural language, the user describes the desired information without giving a specific procedure for obtaining that information.

In this chapter, we will study following three languages:-

1. Relational algebra
2. Tuple relational calculus
3. Domain relational calculus

In these languages, relational algebra is a procedural but tuple and domain relational calculus are non-procedural languages.

Consider the following banking database. We will write all the queries for this database.

7 Relational Algebra

- The relational algebra is a procedural query language.
- It consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- The fundamental operations in the relational algebra are select, project, union, set difference, Cartesian product, and rename. In addition to the fundamental operations, there are several other operations—namely, set intersection, natural join, division, and assignment.

7.1 Fundamental Operations

- The select, project, and rename operations are called unary operations, because they operate on one relation.
- The other three operations operate on pairs of relations and are, therefore, called binary operations.

7.1.1 The Select Operation

The select operation selects tuples that satisfy a given predicate. We use the lowercase Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ. The argument relation is in parentheses after the σ. That is,
$\sigma_P(r)$
Here, r is a name of a relation and P is a predicate.

Example: Select those tuples of the loan relation where the branch is "Perryridge".
Solution: $\sigma_{branch-name="Perryridge"}(loan)$
Example: Find all tuples in which the amount lent is more than \$1200.
Solution: $\sigma_{amount>1200}(loan)$

Note: In general, we allow comparisons using $=, \neq, <, \leq, >, \geq$ in the selection predicate. Furthermore, we can combine several predicates into a larger predicate by using the connectives and ($\wedge$), or ($\vee$), and not ().

Example: Find those tuples pertaining to loans of more than \$1200 made by the "Perryridge" branch.
Solution: $\sigma_{(branch-name="Perryridge")\wedge(amount>1200)}(loan)$

7.1.2 The Project Operation

The project operation is used to select columns of a table. It is denoted by Π. We list those attributes that we wish to appear in the result as a subscript to Π. The argument relation follows in parentheses.
$\Pi_{A,B,C}(r)$
Here, r is a name of a relation and A, B, C are the attributes corresponding selected column.

Example: List all loan numbers and the amount of the loan.
Solution: $\Pi_{loan-number,amount}(loan)$
Example: Find those customers who live in Harrison.
Solution: $\Pi_{customer-name}(\sigma_{city="Harrison"}(Customer))$

7.1.3 The Union Operation

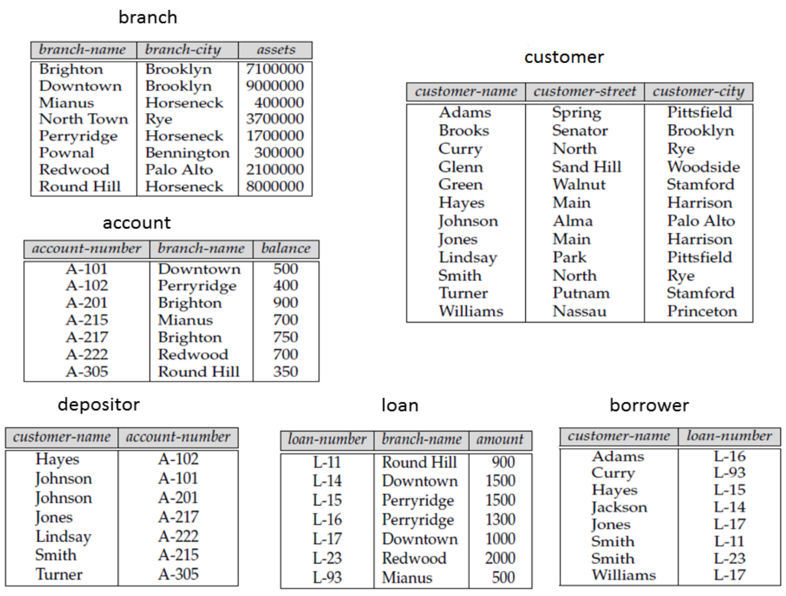
Union operation will be applied if we have to find elements which are belong into either of one relation.
For a union operation r $\cup$ s to be valid, we require that two conditions hold:

1. The relations r and s must be of the same arity. That is, they must have the same number of attributes.
2. The domains of the i^{th} attribute of r and the i^{th} attribute of s must be the same, for all i.

Example: Find the names of all bank customers who have either an account or a loan or both.
Solution: $\Pi_{customer}(depositer) \cup \Pi_{customer}(borrower)$

7.1.4 The Set Difference Operation

The set difference operation, denoted by -, allows us to find tuples that are in one relation but are not in another. The expression r - s produces a relation containing those tuples in r but not in s.



branch

branch-name	branch-city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

customer

customer-name	customer-street	customer-city
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

account

account-number	branch-name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

depositor

customer-name	account-number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

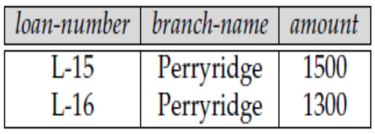
loan

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

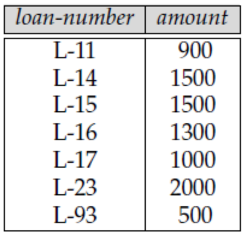
borrower

customer-name	loan-number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

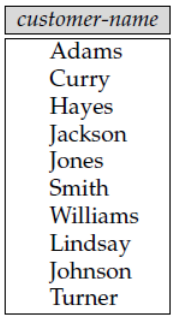
Figure 1: Banking database



loan-number	branch-name	amount
L-15	Perryridge	1500
L-16	Perryridge	1300



loan-number	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500



customer-name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

Example: Find all customers of the bank who have an account but not a loan.
Solution: $\Pi_{customer}(depositer) - \Pi_{customer}(borrower)$
As with the union operation, we must ensure that set differences are taken between compatible relations. Therefore, for a set difference operation r - s to be valid, we require that the relations r and s be of the same arity, and that the domains of the i^{th} attribute of r and the i^{th} attribute of s be the same.

7.1.5 The Cartesian-Product Operation

The Cartesian-product operation, denoted by a cross (), allows us to combine information from any two relations.We write the Cartesian product of relations r_1 and r_2 as $r_1 \ \ r_2$.
Example: Find the names of all customers who have a loan at the Perryridge branch.
Solution: $\Pi_{customer-name}(\sigma_{(borrower.loan-no=loan.loan-no)\wedge(branch-name="Perryridge")}(borrower \times loan)$
Example: Consider the following tables S_1, S_2 and R_1:
Compute the following operations. (a) $S_1 \cup S_2$ (b) $S_1 \cap S_2$ (c) $S_1 - S_2$ (d) $S_1 \times R_1$

Solution: Result of all the operations are shown as the following:-

7.1.6 The Rename Operation

Unlike relations in the database, the results of relational-algebra expressions do not have a name that we can use to refer to them. It is useful to be able to give them names; the rename operator, denoted by the lowercase Greek letter rho (ρ).
Given a relational-algebra expression E, the expression

$$\rho_x(E)$$

returns the result of expression E under the name x.
A second form of the rename operation is as follows. Assume that a relational algebra expression E has arity n. Then, the expression

$$\rho_{x(A_1,A_2,...,A_n)}(E)$$

returns the result of expression E under the name x, and with the attributes renamed to $A_1, A_2, ..., A_n$.

Example: Find the largest account balance in the bank.
Solution: Our strategy is to (1) compute first a temporary relation consisting of those balances that are not the largest and (2) take the set difference between the relation $\Pi_{balance}(account)$ and the temporary relation just computed, to obtain the result.

$$\Pi_{balance}(account) \text{ - } \Pi_{account.balance}(\sigma_{account.balance<d.balance}(account \times \rho_d(account)))$$

Example: Find the names of all customers who live on the same street and in the same city as Smith.
Solution: In this query, first we find Smith's street and city. Second, we match Smith's street and city with other customer street and city. If match found then we select that customer.
$\Pi_{customer-name}(\sigma_{customer.customer-street=smith.street\wedge customer.customer-city=smith.city}(customer \times \rho_{smith(street,city)}(\Pi_{customer-street,customer-city}(\sigma_{customer-name="Smith"}(customer)))))$

customer-name
Johnson
Lindsay
Turner

customer-name	*borrower. loan-number*	*loan. loan-number*	*branch-name*	*amount*
Adams	L-16	L-11	Round Hill	900
Adams	L-16	L-14	Downtown	1500
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Adams	L-16	L-17	Downtown	1000
Adams	L-16	L-23	Redwood	2000
Adams	L-16	L-93	Mianus	500
Curry	L-93	L-11	Round Hill	900
Curry	L-93	L-14	Downtown	1500
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Curry	L-93	L-17	Downtown	1000
Curry	L-93	L-23	Redwood	2000
Curry	L-93	L-93	Mianus	500
Hayes	L-15	L-11		900
Hayes	L-15	L-14		1500
Hayes	L-15	L-15		1500
Hayes	L-15	L-16		1300
Hayes	L-15	L-17		1000
Hayes	L-15	L-23		2000
Hayes	L-15	L-93		500
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
Smith	L-23	L-11	Round Hill	900
Smith	L-23	L-14	Downtown	1500
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Smith	L-23	L-17	Downtown	1000
Smith	L-23	L-23	Redwood	2000
Smith	L-23	L-93	Mianus	500
Williams	L-17	L-11	Round Hill	900
Williams	L-17	L-14	Downtown	1500
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300
Williams	L-17	L-17	Downtown	1000
Williams	L-17	L-23	Redwood	2000
Williams	L-17	L-93	Mianus	500

borrower x loan

customer-name	borrower. loan-number	loan. loan-number	branch-name	amount
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Hayes	L-15	L-15	Perryridge	1500
Hayes	L-15	L-16	Perryridge	1300
Jackson	L-14	L-15	Perryridge	1500
Jackson	L-14	L-16	Perryridge	1300
Jones	L-17	L-15	Perryridge	1500
Jones	L-17	L-16	Perryridge	1300
Smith	L-11	L-15	Perryridge	1500
Smith	L-11	L-16	Perryridge	1300
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300

Result of $\sigma_{branch\text{-}name = \text{“Perryridge”}}(borrower \times loan)$.

customer-name
Adams
Hayes

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Instance S_1 of sailors

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Instance S_2 of sailors

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Instance R_1 of Reserves

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0
28	yuppy	9	35.0
44	guppy	5	35.0

$S_1 \cup S_2$

sid	*sname*	*rating*	*age*
31	Lubber	8	55.5
58	Rusty	10	35.0

$S_1 \cap S_2$

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0

$S_1 - S_2$

(*sid*)	*sname*	*rating*	*age*	(*sid*)	*bid*	*day*
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

$S_1 \times S_2$

balance
500
400
700
750
350

Figure 2: Result of $\Pi_{account.balance}(\sigma_{account.balance<d.balance}(account \times \rho_d(account)))$

balance
900

Figure 3: Largest balance

customer-name
Curry
Smith

7.2 Additional Operations

7.2.1 Intersection Operation

Example: Find all customers who have both a loan and an account.
Solution: $\Pi_{customer}(depositer) \cap \Pi_{customer}(borrower)$
Note: $r \cap s$ = r-(r-s)

7.2.2 Join

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

7.2.3 Types of join operations

7.2.4 Theta (θ) Join or Condition Join

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol θ.
$R_1 \bowtie_\theta R_2$ R_1 and R_2 are relations having attributes $(A_1, A_2, .., A_n)$ and $(B_1, B_2, .., B_n)$ such that the attributes don't have anything in common, that is $R_1 \cap R_2 = \phi$.
Theta join can use all kinds of comparison operators.

7.2.5 Equijoin

When Theta join uses only equality comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Example: Theta join and Equijoin operations are shown as following:-

7.2.6 Natural-Join

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.
Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Example: Natural join operation is shown as following:-

Example: Find the names of all customers who have a loan at the bank, along with the loan number and the loan amount.
Solution: Query without using natural join is
$\Pi_{customer-name,loan-number,amount}(\sigma_{borrower.loan-number=loan.loan-number}(borrower \times loan))$

customer-name
Hayes
Jones
Smith

(*sid*)	*sname*	*rating*	*age*	(*sid*)	*bid*	*day*
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid<R1.sid} R1$$

sid	*sname*	*rating*	*age*	*bid*	*day*
22	Dustin	7	45.0	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{R.sid=S.sid} R1$$

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

- r $\bowtie$ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Equialent query using natural join is
$\Pi_{customer-name,loan-number,amount}(borrower \bowtie loan)$
Example: Find the names of all branches with customers who have an account in the bank and who live in Harrison.
Solution: $\Pi_{branch-name}(\sigma_{customer-city="Harrison"}(customer \bowtie depositor \bowtie account))$
Note: If there is no common attributes between two relations, then natural-join and Cartesian product is equal.

7.2.7 The Division Operation

The division operation, denoted by , is suited to queries that include the phrase "for all." We are describing division operation through an example. Consider two relation instances A and B in which A has (exactly) two fields x and y and B has just one field y, with the same domain as in A. We define the division operation A/B as the set of all x values (in the form of unary tuples) such that for every y value in (a tuple of) B, there is a tuple $< x, y >$ in A.

Another way to understand division is as follows. For each x value in (the first column of) A, consider the set of y values that appear in (the second field of) tuples of A with that x value. If this set contains (all y values in) B, then the x value is in the result of A/B.

Example: Division operation is explain in the following figure:-

Example: Find all customers who have an account at all the branches located in Brooklyn.
Solution: In this query, we will apply the division operator. For this we have to find numerator and denominator of the query. If numerator is N and denominator is D then final query will be N$\div$ D.
In this query, denominator is all the branches located in "Brooklyn". Query for this is
D = $\Pi_{branch-name}(\sigma_{branch-city="Brooklyn"}(branch))$

And numerator is all the customers who have an account with their branch name. Query for this is
N = $\Pi_{customer-name,branch-name}(depositor \bowtie account)$

Therefore, the final query is
N $\div$ D.
Note: Let r(R) and s(S) be given, with S $\subseteq$ R:
$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$

7.2.8 Assignment Operation

It is denoted by $\leftarrow$. If E is a relational algebra query expression, then we can assigned it as like the following:-
r $\leftarrow$ E

7.3 Example:

Consider the following database which consists of three tables. Write the following queries

customer-name	*loan-number*	*amount*
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-23	2000
Smith	L-11	900
Williams	L-17	1000

branch-name
Brighton
Perryridge

A

sno	*pno*
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

branch-name
Brighton
Downtown

Result of $\Pi_{branch\text{-}name}(\sigma_{branch\text{-}city\,=\,\text{“Brooklyn”}}\ (branch)$.

customer-name	*branch-name*
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

Result of $\Pi_{customer\text{-}name,\ branch\text{-}name}\ (depositor \bowtie\ account)$

Customer-name
Johnson

Result of final query

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Instance S_3 of sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Instance R_2 of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Instance B_1 of Boats Reserves

in relational algebra:

1. Find the names of sailors who have reserved boat 103.
   Solution: $\Pi_{sname}((\sigma_{bid=103}(Reserves)) \bowtie Sailors)$

2. Find the names of sailors who have reserved a red boat.
   Solution: $\Pi_{sname}((\sigma_{color="red"}(Boats)) \bowtie Reserves \bowtie Sailors)$

3. Find the colors of boats reserved by Lubber.
   Solution: $\Pi_{color}((\sigma_{sname="Lubber"}(Sailors)) \bowtie Reserves \bowtie Boats)$

4. Find the names of sailors who have reserved at least one boat.
   Solution: $\Pi_{sname}(Sailors \bowtie Reserves)$

5. Find the names of sailors who have reserved a red or a green boat.
   Solution: $temp \leftarrow \Pi_{sname,color}(Sailors \bowtie Reserves \bowtie Boat)$
   $\Pi_{sname}(\sigma_{color="red"}(temp)) \cup \Pi_{sname}(\sigma_{color="green"}(temp))$

6. Find the names of sailors who have reserved a red and a green boat.
   Solution: $temp \leftarrow \Pi_{sname,color}(Sailors \bowtie Reserves \bowtie Boat)$
   $\Pi_{sname}(\sigma_{color="red"}(temp)) \cap \Pi_{sname}(\sigma_{color="green"}(temp))$

7. Find the names of sailors who have reserved at least two boats.
   Solution: $temp \leftarrow \Pi_{sid,sname,bid}(Sailors \bowtie Reserves)$
   $\Pi_{sname}(\sigma_{temp1.sid=r.sid \wedge temp1.bid \neq r.bid}(temp \times \rho_r(temp)))$

8. Find the sids of sailors with age over 20 who have not reserved a red boat.
   Solution: $\Pi_{sid}(\sigma_{age>20}(Sailors)) - \Pi_{sid}((\sigma_{color="red"}(Boats)) \bowtie Reserves \bowtie Sailors)$

9. Find the names of sailors who have reserved all boats.
   Solution: $N \leftarrow \Pi_{sname,bid}(Sailors \bowtie Reserves)$
   $D \leftarrow \Pi_{bid}(Boat)$
   Therefore, final query is $N \div D$

10. Find the names of sailors who have reserved all boats called Interlake.
    Solution: $N \leftarrow \Pi_{sname,bid}(Sailors \bowtie Reserves)$
    $D \leftarrow \Pi_{bid}(\sigma_{bname="Interlake"}(Boat))$
    Therefore, final query is $N \div D$

7.4 Extended Relational-Algebra Operations

7.4.1 Generalized Projection

The generalized-projection operation extends the projection operation by allowing arithmetic functions to be used in the projection list. The generalized projection operation has the form

$$\Pi_{F_1,F_2,...,F_n}(E)$$

Where E is any relational-algebra expression, and each of $F_1, F_2, ..., F_n$ is an arithmetic expression involving constants and attributes in the schema of E. As a special case, the arithmetic expression may be simply an attribute or a constant.

7.4.2 Aggregate Functions

Aggregate functions take a collection of values and return a single value as a result. For example, the aggregate function **sum** takes a collection of values and returns the sum of the values. Following aggregate functions are used.

1. sum
2. avg
3. count
4. min
5. max

Example: Find the sum of all account balances.
Solution: Query for this will be

$$\mathcal{G}_{sum(balance)}(account)$$

The symbol $\mathcal{G}$ is the letter G in calligraphic font; read it as "calligraphic G." The relational-algebra operation $\mathcal{G}$ signifies that aggregation is to be applied, and its subscript specifies the aggregate operation to be applied.

Example: Find the sum of all account balances of each branch.
Solution: Query for this will be

$${}_{branch-name}\mathcal{G}_{sum(balance)}(account)$$

Example: Find the number of depositors.
Solution: Query for this will be

$$\mathcal{G}_{count(customer-name)}(depositor)$$

7.4.3 Outer Join

The outer-join operation is an extension of the join operation to deal with missing information. There are actually three forms of the operation: left outer join, denoted ; right outer join, denoted ; and full outer join, denoted . All three forms of outer join compute the join, and add extra tuples to the result of the join.

The left outer join ( ) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join.

The right outer join ( ) is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join.

The full outer join( ) does both of those operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right

relation that did not match any from the left relation, and adding them to the result of the join.
Example: Consider the following two relations Employee and FT-works:-
Natural join and left outer join are the followings:-
Right outer join and Full outer join are the followings:-

7.5 Modification of the Database

7.5.1 Deletion

We can delete only whole tuples; we cannot delete values on only particular attributes. In relational algebra a deletion is expressed by
$r \leftarrow r - E$
Where r is a relation and E is a relational-algebra query.

Example: Delete all of Smith's account records.
Solution: $depositor \leftarrow depositor - \sigma_{customer-name}(depositor)$

Example: Delete all loans with amount in the range 0 to 50.
Solution: $loan \leftarrow loan - \sigma_{amount \geq 0 \ \wedge \ amount \leq 50}(loan)$

Example: Delete all accounts at branches located in Needham.
Solution:
$account \leftarrow account - r$
Where r is
$r \leftarrow \Pi_{account-number,branch-name,balance}(\sigma_{branch-city="Needham"}(branch \bowtie account))$

7.5.2 Insertion

To insert data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. Obviously, the attribute values for inserted tuples must be members of the attribute's domain. Similarly, tuples inserted must be of the correct arity. The relational algebra expresses an insertion by
$r \leftarrow r \cup E$
where r is a relation and E is a relational-algebra expression.

Example: Suppose that we wish to insert the fact that Smith has \$1200 in account A-973 at the Perryridge branch.
Solution:
$depositor \leftarrow depositor \cup ("Smith", "A-973")$
$account \leftarrow account \cup ("A-973", "Perryridge", 1200)$

employee-name	*street*	*city*
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

Employee table

employee-name	*branch-name*	*salary*
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

FT-works table

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500

Employee ⋈ FT-works

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*

Employee ⟕ FT-works

7.5.3 Updating

To update value of a particular row into a relation, we write the following type of query:-

$r \leftarrow \Pi_{F_1,F_2,\ldots,F_n}(r)$

where each F_i is either the i^{th} attribute of r, if the i^{th} attribute is not updated, or, if the attribute is to be updated, F_i is an expression, involving only constants and the attributes of r, that gives the new value for the attribute.
If we want to select some tuples from r and to update only them, we can use the following expression; here, P denotes the selection condition that chooses which tuples to update:

$r \leftarrow \Pi_{F_1,F_2,\ldots,F_n}(\sigma_P(r)) \cup (r - \sigma_P(r))$

Example: Suppose that interest payments are being made, and that all balances are to be increased by 5 percent.
Solution: $account \leftarrow \Pi_{account-number,branch-name,balance*1.05}(account)$

Example: Suppose that accounts with balances over $10,000 receive 6 percent interest, whereas all others receive 5 percent.
Solution: $account \leftarrow \Pi_{account-number,branch-name,balance*1.06}(\sigma_{balance>10000}(account)) \cup \Pi_{account-number,branch-name,balance*1.05}(\sigma_{balance\leq 10000}(account))$

7.6 Views

Any relation that is not part of the logical model, but is made visible to a user as a virtual relation, is called a view.
We define a view by using the create view statement. To define a view, we must give the view a name, and must state the query that computes the view. The form of the create view statement is

create view v as ¡query expression¿ where¡query expression¿is any legal relational-algebra query expression. The view name is represented by v.

Example: Consider the view consisting of branches and their customers. We wish this view to be called all-customer. We define this view as follows:

create view all-customer as $\Pi_{branch-name,customer-name}(depositor \bowtie account)$
$\cup \Pi_{branch-name,customer-name}(borrower \bowtie loan)$

Once we have defined a view, we can use the view name to refer to the virtual relation that the view generates. Using the view all-customer, we can find all customers of the Perryridge branch by writing

$Pi_{customer-name}(\sigma_{branch-name=\text{\Perryridge''}}(all - customer))$

8 The Tuple Relational Calculus

A query in the tuple relational calculus is expressed as

{ t ! P(t)}

that is, it is the set of all tuples t such that predicate P is true for t. We use t[A] to denote the value of tuple t on attribute A, and we use t ∈ r to denote that tuple t is in relation r.

8.1 Example Queries

- Find the branch-name, loan-number, and amount for loans of over \$1200.
  Solution: $\{\ t \mid t \in loan \wedge t[amount] > 1200\ \}$

- Find the loan number for each loan of an amount greater than \$1200.
  Solution: $\{\ t \mid \exists\ s \in loan(t[loan - number] = s[loan - number] \wedge s[amount] > 1200)\ \}$

- Find the names of all customers who have a loan from the Perryridge branch.
  Solution: $\{\ t \mid \exists\ s \in borrower(t[customer - name] = s[customer - name] \wedge \exists\ s \in loan(u[loan{-}number] = s[loan{-}number] \wedge u[branch{-}name] = "Perryridge"))\}$

- Find all customers who have a loan, an account, or both at the bank.
  Solution: $\{\ t \mid \exists s \in borrower(t[customer - name] = s[customer - name]) \vee\ \exists u \in depositor(t[customer - name] = u[customer - name])\}$

- Find those customers who have both an account and a loan at the bank.
  Solution: $\{\ t \mid \exists s \in borrower(t[customer - name] = s[customer - name]) \wedge\ \exists u \in depositor(t[customer - name] = u[customer - name])\}$

- Find all customers who have an account at the bank but do not have a loan from the bank.
  Solution: $\{\ t \mid \exists s \in depositor(t[customer - name] = s[customer - name]) \wedge \exists u \in borrower(t[customer - name] = u[customer - name])\}$

- Find all customers who have an account at all branches located in Brooklyn.
  Solution: $\{ t \mid \forall u \in branch(u[branch{-}city] = "Brooklyn" \Rightarrow \exists s \in depositor(t[customer{-}name] = s[customer - name] \wedge \exists w \in account(w[account - number] = s[account - number] \wedge w[branch - name] = u[branch - name])))\}$

Note:

1. The formula P $\Rightarrow$ Q means "P implies Q"; that is, "if P is true, then Q must be true."

2. Note that P $\Rightarrow$ Q is logically equivalent to P $\vee$ Q.

9 Domain Relational Calculus

It uses domain variables that take on values from an attributes domain, rather than values for an entire tuple. The domain relational calculus, however, is closely related to the tuple relational calculus.
An expression in the domain relational calculus is of the form

$$\{< x_1, x_2, ..., x_n > \mid P(x_1, x_2, ..., x_n)\}$$

where $x_1, x_2, ..., x_n$ represent domain variables. P represents a formula composed of atoms, as was the case in the tuple relational calculus.

9.1 Example Queries

- Find the branch-name, loan-number, and amount for loans of over \$1200.
  Solution: $\{ < l, b, a > \mid < l, b, a > \in \mathit{loan} \wedge a > 1200 \}$

- Find the loan number for each loan of an amount greater than \$1200.
  Solution: $\{ < l > \mid \exists\ b, a\ (< l, b, a > \in \mathit{loan} \wedge a > 1200) \}$

- Find the names of all customers who have a loan from the Perryridge branch and find the loan amount.
  Solution: $\{ < c, a > \mid \exists\ l (< c, l > \in \mathit{borrower} \wedge \exists b\ (< l, b, a > \in \mathit{loan} \wedge b = \text{"}\mathit{Perryridge}\text{"}))\}$

- Find all customers who have a loan, an account, or both at the Perryridge branch.
  Solution: $\{< c > \mid \exists l (< c, l > \in \mathit{borrower} \wedge \exists b, a (< l, b, a > \in \mathit{loan} \wedge b = \text{"}\mathit{Perryridge}\text{"})) \vee \exists a (< c, a > \in \mathit{depositor} \wedge \exists b, n (< a, b, n > \in \mathit{account} \wedge b = \text{"}\mathit{Perryridge}\text{"}))\}$

- Find all customers who have an account at all branches located in Brooklyn.
  Solution: $\{< c > \mid \forall x, y, z (< x, y, z > \in \mathit{branch} \wedge y = \text{"}\mathit{Brooklyn}\text{"} \Rightarrow$
  $\exists a, b (< a, x, b > \in \mathit{account} \wedge < c, a > \in \mathit{depositor}))\}$

10 Exercise

1. Consider the following relational database, where the primary keys are underlined.
   employee (<u>person-name</u>, street, city)
   works (<u>person-name</u>, company-name, salary)
   company (<u>company-name</u>, city)
   manages (<u>person-name</u>, manager-name)

   Give an expression in the relational algebra to express each of the following queries:

   (a) Find the names of all employees who work for First Bank Corporation.
   (b) Find the names and cities of residence of all employees who work for First Bank Corporation.
   (c) Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than \$10,000 per annum.
   (d) Find the names of all employees in this database who live in the same city as the company for which they work.
   (e) Find the names of all employees who live in the same city and on the same street as do their managers.
   (f) Find the names of all employees in this database who do not work for First Bank Corporation.
   (g) Find the names of all employees who earn more than every employee of Small Bank Corporation.

(h) Assume the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

(i) Find the company with the most employees.

(j) Find the company with the smallest payroll.

(k) Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

Solution:

(a) $\Pi_{person-name}(\sigma_{company-name="First\ Bank\ Corporation"}(works))$

(b) $\Pi_{person-name,city}(\sigma_{company-name="First\ Bank\ Corporation"}(employee \bowtie works))$

(c) $\Pi_{person-name,street,city}(\sigma_{company-name="First\ Bank\ Corporation" \wedge salary>10000}(employee \bowtie works))$

(d) $\Pi_{person-name}(employee \bowtie works \bowtie company))$

(e) $temp \leftarrow \Pi_{manager-name,street,city}(\sigma_{employee.person-name=manages.manager-name}(employee \times manages)$
$\Pi_{person-name}(employee \bowtie manages \bowtie temp)$

(f) $\Pi_{person-name}(\sigma_{company-name \neq "FirstBankCorporation"}(works))$

(g) $\Pi_{person-name}(works) - \Pi_{works.person-name}(\sigma_{works.salary \leq r.salary}(works \times (\rho_r(\sigma_{company-name="Small\ Bank\ corporation"}(works)))))$

(h) $company \div \Pi_{city}(\sigma_{company-name="Small\ Bank\ Corporation"}(company)))$

(i) $temp \leftarrow\ {}_{company-name}\mathcal{G}_{count(person-name)\ as\ person-count}(works)$
$\Pi_{company-name}(\sigma_{temp.person-count=r.max-employee}(temp \times \rho_r(\mathcal{G}_{max(person-count)\ as\ max-employee}(temp))))$

(j) $temp \leftarrow\ {}_{company-name}\mathcal{G}_{sum(salary)\ as\ payroll}(works)$
$\Pi_{company-name}(\sigma_{temp.payroll=r.min-payroll}(temp \times \rho_r(\mathcal{G}_{min(payroll)\ as\ min-payroll}(temp))))$

(k) $temp1 \leftarrow \mathcal{G}_{avg(salary)\ as\ avg-salary}(\sigma_{company-name="First\ Bank\ Corporation"}(works)$
$temp2 \leftarrow_{company-name}\ \mathcal{G}_{avg(salary)\ as\ avg-salary}(works)$
$\Pi_{company-name}(\sigma_{temp2.avg-salary>temp1.avg-salary}(temp2 \times temp1))$

2. Consider the relational database of previous question. Give an expression in the relational algebra for each request:

(a) Modify the database so that Jones now lives in Newtown.

(b) Give all employees of First Bank Corporation a 10 percent salary raise.

(c) Give all managers in this database a 10 percent salary raise.

(d) Give all managers in this database a 10 percent salary raise, unless the salary would be greater than \$100,000. In such cases, give only a 3 percent raise.

(e) Delete all tuples in the works relation for employees of Small Bank Corporation.

Solution:

(a) $employee \leftarrow \Pi_{person-name,street,"Newtown"}(\sigma_{person-name="Jones"}(Employee)) \cup (employee - \sigma_{person_name="Jones"}(employee))$

(b) $works \leftarrow \Pi_{person-name,company-name,salary*1.1}(\sigma_{company-name="FirstBankCorporation"}(works)) \cup$ $(works - \sigma_{company-name="FirstBankCorporation"}(works)$

(c) $temp \leftarrow \Pi_{works.person-name,company-name,salary}(\sigma_{works.person-name=manages.manages-name}(works \times manages))$
$works \leftarrow (works - temp) \cup \Pi_{works.person-name,comapny-name,salary*1.1}(temp)$

(d) $temp1 \leftarrow \Pi_{works.person-name,company-name,salary}(\sigma_{works.person-name=manages.manages-name}(works \times manages))$
$temp2 \leftarrow \Pi_{works.person-name,company-name,salary*1.03}(\sigma_{salary*1.1>100000}(temp1))$
$temp2 \leftarrow temp2 \cup \Pi_{works,person-name,company-name,salary*1.1}(\sigma_{salary*1.1 \leq 100000}(temp1))$
$works \leftarrow (works - temp1) \cup temp2$

(e) $works \leftarrow works - \sigma_{company-name="Small\ Bank\ Corporation"}(works)$

3. Let the following relation schemas be given:
R = (A, B, C) and S = (D, E, F)
Let relations r(R) and s(S) be given. Give an expression in the tuple relational calculus that is equivalent to each of the following:

(a) $\Pi_A(r)$
(b) $\sigma_{B=17}(r)$
(c) r×s
(d) $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

Solution:

(a) $\{t \mid \exists u \in r(t[A] = u[A])\}$
(b) $\{t \mid t \in r \wedge t[B] = 17\}$
(c) $\{t \mid \exists u \in r(t[A] = u[A] \wedge t[B] = u[B] \wedge t[C] = u[C] \wedge \exists w \in s(t[D] = w[D] \wedge t[E] = w[E] \wedge t[F] = w[F]))\}$
(d) $\{t \mid \exists u \in r(t[A] = u[A] \wedge \exists w \in s(t[F] = w[F] \wedge u[C] = w[D]))\}$

4. Let R = (A, B, C), and let r_1 and r_2 both be relations on schema R. Give an expression in the domain relational calculus that is equivalent to each of the following:

(a) $\Pi_A(r_1)$
(b) $\sigma_{B=17}(r_1)$
(c) $r_1 \cup r_2$
(d) $r_1 \cap r_2$
(e) $r_1 - r_2$
(f) $\Pi_{A,B}(r_1) \bowtie \Pi_{B,C}(r_2)$

Solution:

(a) $\{< a > \mid \exists b, c(< a, b, c > \in r_1)\}$
(b) $\{< a, b, c > \mid\ < a, b, c > \in r_1 \wedge b = 17)\}$
(c) $\{< a, b, c > \mid\ < a, b, c > \in r_1 \vee < a, b, c > \in r_2\}$

(d) $\{< a, b, c > \mid \ < a, b, c > \in r_1 \land < a, b, c > \in r_2\}$

(e) $\{< a, b, c > \mid \ < a, b, c > \in r_1 \land < a, b, c > \notin r_2\}$

(f) $\{< a, b, c > \mid \exists p, q(< a, b, p > \in r_1 \land < q, b, c > \in r_2)\}$

5. Let R = (A, B) and S = (A, C), and let r(R) and s(S) be relations. Write relational-algebra expressions equivalent to the following domain-relational calculus expressions:

   (a) $\{< a > \mid \exists b(< a, b > \in r \land b = 17)\}$

   (b) $\{< a, b, c > \mid < a, b > \in r \land < a, c > \in s\}$

   (c) $\{< a > \mid \exists b(< a, b > \in r) \lor \forall c(\exists d(< d, c > \in s) \Rightarrow < a, c > \in s)\}$

   (d) $\{< a > \mid \exists c(< a, c > \in s \land \exists b_1, b_2(< a, b_1 > \in r \land < c, b_2 > \in r \land b_1 > b_2))\}$

   Solution:

   (a) $\Pi_A(\sigma_{B=17}(r))$

   (b) $r \bowtie s$

   (c) $\Pi_A(r) \cup (s \div \Pi_C(s))$

   (d) $\Pi_{r.A}(\sigma_{r.B>r1.B}((r \bowtie s) \times (\rho_{r1}(r))))$

6. Given two relations R_1 and R_2, where R_1 contains N_1 tuples,R_2 contains N_2 tuples, and $N_2 > N_1 > 0$, give the minimum and maximum possible sizes (in tuples) for the result relation produced by each of the following relational algebra expressions. In each case, state any assumptions about the schemas for R_1 and R_2 that are needed to make the expression meaningful:

   (a) $R_1 \cup R_2$

   (b) $R_1 \cap R_2$

   (c) $R_1 - R_2$

   (d) $R_1 \times R_2$

   (e) $\sigma_{a=5}(R_1)$

   (f) $\Pi_a(R_1)$

   (g) $R_1 \div R_2$

   Solution:

   (a) Minimum number of tuples = N_2
   Maximum number of tuples = $N_1 + N_2$

   (b) Minimum number of tuples = 0
   Maximum number of tuples = N_1

   (c) Minimum number of tuples = 0
   Maximum number of tuples = N_1

   (d) Minimum number of tuples = $N_1 * N_2$
   Maximum number of tuples = $N_1 * N_2$

(e) Assume attribute a in R_1 is a primary key. In this case
Minimum number of tuples = 0
Maximum number of tuples = 1

Assume attribute a in R_1 is not a primary key. In this case
Minimum number of tuples = 0
Maximum number of tuples = N_1

(f) Assume attribute a in R_1 is a primary key. In this case
Minimum number of tuples = N_1
Maximum number of tuples = N_1

Assume attribute a in R_1 is not a primary key. In this case
Minimum number of tuples = 1
Maximum number of tuples = N_1

(g) Minimum number of tuples = 0
Maximum number of tuples = 0

7. Consider the following schema:
Suppliers(sid: integer, sname: string, address: string)
Parts(pid: integer, pname: string, color: string)
Catalog(sid: integer, pid: integer, cost: real)

The key fields are underlined, and the domain of each field is listed after the field name. Thus sid is the key for Suppliers, pid is the key for Parts, and sid and pid together form the key for Catalog. The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in relational algebra, tuple relational calculus, and domain relational calculus:

(a) Find the names of suppliers who supply some red part.
(b) Find the sids of suppliers who supply some red or green part.
(c) Find the sids of suppliers who supply some red part or are at 221 Packer Street.
(d) Find the sids of suppliers who supply some red part and some green part.
(e) Find the sids of suppliers who supply every part.
(f) Find the sids of suppliers who supply every red part.
(g) Find the sids of suppliers who supply every red or green part.
(h) Find the sids of suppliers who supply every red part or supply every green part.
(i) Find pairs of sids such that the supplier with the first sid charges more for some part than the supplier with the second sid.
(j) Find the pids of parts that are supplied by at least two different suppliers.
(k) Find the pids of the most expensive parts supplied by suppliers named Yosemite Sham.

Solution:

(a) **Relational algebra query is**
$\Pi_{sname}(Suppliers \bowtie Catalog \bowtie \Pi_{pid}(\sigma_{color=''red''}(Parts)))$

Tuple relational calculus query is
$\{\ t \mid \exists s \in Suppliers(t[sname] = s[sname] \land \exists u \in Catalog(s[sid] = u[sid] \land \exists w \in Parts(u[pid] = w[pid] \land w[color] = ''red'')))\}$

Domain relational calculus query is
$\{< b > \mid \exists a, c(< a, b, c >\in Suppliers \land \exists d, e(< a, d, e >\in Catalog \land \exists f, g(< d, f, g >\in Parts \land g = ''red'')))\}$

(b) **Relational algebra query is**
$\Pi_{sid}(\Pi_{pid}(\sigma_{color=''red'' \lor color=''green''(Parts)} \bowtie Catalog))$

Tuple relational calculus query is
$\{\ t \mid \exists u \in Catalog(t[sid] = u[sid] \land \exists w \in Parts(u[pid] = w[pid] \land (w[color] = ''red'' \lor w[color] = ''green''))))\}$

Domain relational calculus query is
$\{< a > \mid \exists b, c(< a, b, c >\in Catalog \land \exists d, e(< b, d, e >\in Parts \land (e = ''red'' \lor e = ''green'')))\}$

(c) **Relational algebra query is**
$\Pi_{sid}(\sigma_{color=''red''}(Catalog \bowtie Parts)) \cup \Pi_{sid}(\sigma_{address=''221\ Packer\ Street''}(Suppliers))$

Tuple relational calculus query is
$\{\ t \mid \exists u \in Catalog(t[sid] = u[sid] \land \exists w \in Parts(u[pid] = w[pid] \land w[color] = ''red''))) \lor \exists s \in Suppliers(t[sid] = s[sid] \land s[address] = ''220\ Packer\ Street'')\}$

Domain relational calculus query is
$\{< a > \mid \exists b, c(< a, b, c >\in Catalog \land \exists d, e(< b, d, e >\in Parts \land e = ''red'')) \lor \exists b, c(< a, b, c >\in Suppliers \land c = ''220\ Packer\ Street'')\}$

(d) **Relational algebra query is**
$\Pi_{sid}(\sigma_{color=''red''}(Catalog \bowtie Parts)) \cap \Pi_{sid}(\sigma_{color=''green''}(Catalog \bowtie Parts))$

Tuple relational calculus query is
$\{\ t \mid \exists u \in Catalog(t[sid] = u[sid] \land \exists w \in Parts(u[pid] = w[pid] \land w[color] = ''red''))) \land \exists u \in Catalog(t[sid] = u[sid] \land \exists w \in Parts(u[pid] = w[pid] \land w[color] = ''green'')))\}$

Domain relational calculus query is
$\{< a > \mid \exists b, c(< a, b, c >\in Catalog \land \exists d, e(< b, d, e >\in Parts \land e = ''red'')) \land \exists b, c(< a, b, c >\in Catalog \land \exists d, e(< b, d, e >\in Parts \land e = ''green''))\}$

(e) **Relational algebra query is**
$\Pi_{sid,pid}(Catalog) \div \Pi_{pid}(Parts)$

Tuple relational calculus query is
$\{\ t \mid \forall s \in Parts \Rightarrow \exists u \in Catalog(t[sid] = u[sid] \wedge s[pid] = u[pid])\}$

Domain relational calculus query is
$\{\ < s >\ \mid \forall a, b, c(< a, b, c > \in Parts \Rightarrow \exists d(< s, a, d > \in Catalog))\}$

(f) **Relational algebra query is**
$\Pi_{sid,pid}(Catalog) \div \Pi_{pid}(Parts)$

Tuple relational calculus query is
$\{\ t \mid \forall s \in Parts \Rightarrow \exists u \in Catalog(t[sid] = u[sid] \wedge s[pid] = u[pid])\}$

Domain relational calculus query is
$\{\ < s >\ \mid \forall a, b, c(< a, b, c > \in Parts \Rightarrow \exists d(< s, a, d > \in Catalog))\}$

(g) **Relational algebra query is**
$\Pi_{sid,pid}(Catalog) \div \Pi_{pid}(\sigma_{color="red" \vee color="green"}(Parts))$

Tuple relational calculus query is
Domain relational calculus query is

(h) **Relational algebra query is**
$\Pi_{sid,pid}(Catalog) \div \Pi_{pid}(\sigma_{color="red"}(Parts)) \vee \Pi_{sid,pid}(Catalog) \div \Pi_{pid}(\sigma_{color="green"}(Parts))$

Tuple relational calculus query is
Domain relational calculus query is

(i) **Relational algebra query is**
$\Pi_{r.sid,s.sid}(\sigma_{r.pid=s.pid \wedge r.sid \neq s.sid \wedge r.cost > s.cost}(\rho_r(Catalog) \times \rho_s(Catalog)))$

Tuple relational calculus query is
Domain relational calculus query is

(j) **Relational algebra query is**
$\Pi_{r.pid}(\sigma_{r.pid=s.pid \wedge r.sid \neq s.sid}(\rho_r(Catalog) \times \rho_s(Catalog))$

Tuple relational calculus query is
Domain relational calculus query is

(k) **Relational algebra query is**
$r \leftarrow \Pi_{pid,cost}(\sigma_{sname="YosemiteSham"}(Suppliers) \bowtie Catalog)$
$\Pi_{pid}(r) - \Pi_{r.pid}(\sigma_{r.cost < s.cost}(r \times \rho_s(r)))$

Tuple relational calculus query is
Domain relational calculus query is

8. Consider the Supplier-Parts-Catalog schema from the previous question. State what the following queries compute:

   (a) $\Pi_{sname}(\Pi_{sid}(\sigma_{color="red"}(Parts)) \bowtie (\sigma_{cost<100}(Catalog)) \bowtie Suppliers)$

   (b) $\Pi_{sname}(\Pi_{sid}((\sigma_{color="red"}(Parts)) \bowtie (\sigma_{cost<100}(Catalog)) \bowtie Suppliers))$

   (c) $(\Pi_{sname}((\sigma_{color="red"}(Parts)) \bowtie (\sigma_{cost<100}(Catalog)) \bowtie Suppliers)) \cap$
   $(\Pi_{sname}((\sigma_{color="green"}(Parts)) \bowtie (\sigma_{cost<100}(Catalog)) \bowtie Suppliers))$

   (d) $(\Pi_{sid}((\sigma_{color="red"}(Parts)) \bowtie (\sigma_{cost<100}(Catalog)) \bowtie Suppliers)) \cap$
   $(\Pi_{sid}((\sigma_{color="green"}(Parts)) \bowtie (\sigma_{cost<100}(Catalog)) \bowtie Suppliers))$

   (e) $\Pi_{sname}((\Pi_{sid,sname}((\sigma_{color="red"}(Parts)) \bowtie (\sigma_{cost<100}(Catalog)) \bowtie Suppliers)) \cap$
   $(\Pi_{sid,sname}((\sigma_{color="green"}(Parts)) \bowtie (\sigma_{cost<100}(Catalog)) \bowtie Suppliers)))$

   Solution:

   (a) Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars.

   (b) This Relational Algebra statement does not return anything because of the sequence of projection operators. Once the sid is projected, it is the only field in the set. Therefore, projecting on sname will not return anything.

   (c) Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.

   (d) Find the Supplier ids of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.

   (e) Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.

9. Consider the following relations containing airline flight information:
   Flights(<u>flno: integer</u>, from: string, to: string, distance: integer, departs: time, arrives: time)
   Aircraft(<u>aid: integer</u>, aname: string, cruisingrange: integer)
   Certified(<u>eid: integer, aid: integer</u>)
   Employees(<u>eid: integer</u>, ename: string, salary: integer)

   Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft (otherwise, he or she would not qualify as a pilot), and only pilots are certified to fly.
   Write the following queries in relational algebra, tuple relational calculus, and domain relational calculus. Note that some of these queries may not be expressible in relational algebra (and, therefore, also not expressible in tuple and domain relational calculus)! For such queries, informally explain why they cannot be expressed.

   (a) Find the eids of pilots certified for some Boeing aircraft.

   (b) Find the names of pilots certified for some Boeing aircraft.

   (c) Find the aids of all aircraft that can be used on non-stop flights from Bonn to Madras.

(d) Identify the flights that can be piloted by every pilot whose salary is more than \$100,000. (Hint: The pilot must be certified for at least one plane with a sufficiently large cruising range.)

(e) Find the names of pilots who can operate some plane with a range greater than 3,000 miles but are not certified on any Boeing aircraft.

(f) Find the eids of employees who make the highest salary.

(g) Find the eids of employees who make the second highest salary.

(h) Find the eids of pilots who are certified for the largest number of aircraft.

(i) Find the eids of employees who are certified for exactly three aircraft.

(j) Find the total amount paid to employees as salaries.

Solution:

(a) **Relational algebra query is**
$\Pi_{eid}(Certified \bowtie \sigma_{aname="Boeing"}(Aircraft))$

(b) **Relational algebra query is**
$\Pi_{ename}(Employees \bowtie Certified \bowtie \sigma_{aname="Boeing"}(Aircraft))$

(c) **Relational algebra query is**
$\Pi_{aid}(\sigma_{cruisingrange>distance}(Aircraft \times \sigma_{from="Bonn" \wedge to="Madras"}(Flights)))$

(d) **Relational algebra query is**
$\Pi_{flno}(\sigma_{cruisingrange>distance \wedge salary>100000}(Flights \bowtie Aircraft \bowtie Certified \bowtie Employee))$

(e) **Relational algebra query is**
$\Pi_{ename}(Employees \bowtie Certified \bowtie \sigma_{cruisingrange>3000 \wedge aname \neq "Boeing"}(Aircraft))$

(f) **Relational algebra query is**
The approach to take is first find all the employees who do not have the highest salary. Subtract these from the original list of employees and what is left is the highest paid employees.
$\Pi_{eid}(Employees) - \Pi_{r.eid}(\sigma_{r.salary<s.salary}(\rho_r(Employees) \times \rho_s(Employees)))$

(g) **Relational algebra query is**
$temp \leftarrow \Pi_{r.eid,r.salary}(\sigma_{r.salary<s.salary}(\rho_r(Employees) \times \rho_s(Employees)))$
$\Pi_{eid}(temp) - \Pi_{temp.eid}(\sigma_{temp.salary<s.salary}(temp \times \rho_s(temp)))$

(h) **Relational algebra query is**
$temp \leftarrow {}_{eid}\mathcal{G}_{count(aid)\ as\ count-aid}(Certified)$
$temp1 \leftarrow \mathcal{G}_{max(count-aid)\ as\ max}(temp)$
$\Pi_{eid}(\sigma_{count-aid=max}(temp \times temp1))$

(i) **Relational algebra query is**
$temp \leftarrow {}_{eid}\mathcal{G}_{count(aid)\ as\ count-aid}(Certified)$
$\Pi_{eid}(\sigma_{count-aid=3}(temp))$

(j) **Relational algebra query is**
$\mathcal{G}_{sum(salary)}(Employees)$

10. What is an unsafe query? Give an example and explain why it is important to disallow such queries.

Solution: An unsafe query is a query in relational calculus that has an infinite number of results. An example of such a query is:

$\{S\ !\ (S \in \text{Sailors})\}$

The query is for all things that are not sailors which of course is everything else. Clearly there is an infinite number of answers, and this query is unsafe. It is important to disallow unsafe queries because we want to be able to get back to users with a list of all the answers to a query after a finite amount of time.

STRUCTURED QUERY LANGUAGE (SQL)

1 Basic Structure

The basic structure of an SQL expression consists of three clauses: select, from, and where.

- The select clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.
- The from clause corresponds to the Cartesian-product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.
- The where clause corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the from clause.

A typical SQL query has the form

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
where P

Each A_i represents an attribute, and each r_i a relation. P is a predicate. The query is equivalent to the relational-algebra expression

$$\Pi_{A_1,A_2,...,A_n}(\sigma_P(r_1 \times r_2 \times .... \times r_m))$$

1.1 Schema Definition in SQL

We define an SQL relation by using the **create table** command:

create table $r(A_1\ D_1, A_2\ D_2, ..., A_n\ D_n, < integrity - constraint_1 >, ..., < integrity - constraint_k >)$ where r is the name of the relation, each A_i is the name of an attribute in the schema of relation r, and D_i is the domain type of values in the domain of attribute A_i. The allowed integrity constraints include

- **primary key** $(A_{j_1}, A_{j_2}, ..., A_{j_m})$**:** The primary key specification says that attributes $A_{j_1}, A_{j_2}, ..., A_{j_m}$ form the primary key for the relation. The primary key attributes are required to be non-null and unique; that is, no tuple can have a null value for a primary key attribute, and no two tuples in the relation can be equal on all the primary-key attributes. Although the primary key specification is optional, it is generally a good idea to specify a primary key for each relation.

- **check(P):** The check clause specifies a predicate P that must be satisfied by every tuple in the relation.

Example: Consider the following definition of tables:-

- create table customer (customer-name char(20),
  customer-street char(30),
  customer-city char(30),
  primary key (customer-name))

- create table branch
  (branch-name char(15),
  branch-city char(30),
  assets integer,
  primary key (branch-name),
  check (assets $\geq$ 0))

- create table account
  (account-number char(10),
  branch-name char(15),
  balance integer,
  primary key (account-number),
  check (balance $\geq$ 0))

- create table depositor
  (customer-name char(20),
  account-number char(10),
  primary key (customer-name, account-number))

- create table student
  (name char(15) not null,
  student-id char(10),
  degree-level char(15),
  primary key (student-id),
  check (degree-level in ('Bachelors', 'Masters', 'Doctorate')))

Note: SQL also supports an integrity constraint
unique $(A_{j_1}, A_{j_2}, ..., A_{j_m})$
The unique specification says that attributes $A_{j_1}, A_{j_2}, ..., A_{j_m}$ form a candidate key.

1.2 Some queries

Consider the following relation schemas:-

Branch-schema = (branch-name, branch-city, assets)
Customer-schema = (customer-name, customer-street, customer-city)

Loan-schema = (loan-number, branch-name, amount)
Borrower-schema = (customer-name, loan-number)
Account-schema = (account-number, branch-name, balance)
Depositor-schema = (customer-name, account-number)

- Find the names of all branches in the loan relation.
  Solution:
  ```
  select branch-name
  from loan
  ```

- Find all loan numbers for loans made at the Perryridge branch with loan amounts greater that $1200.
  Solution:
  ```
  select loan-number
  from loan
  where branch-name = 'Perryridge' and amount > 1200
  ```

- Find the loan number of those loans with loan amounts between $90,000 and $100,000.
  Solution:
  ```
  select loan-number
  from loan
  where amount ≤ 100000 and amount ≥ 90000
  ```

- For all customers who have a loan from the bank, find their names,loan numbers and loan amount.
  Solution:
  ```
  select customer-name, borrower.loan-number, amount
  from borrower, loan
  where borrower.loan-number = loan.loan-number
  ```

- Find the customer names, loan numbers, and loan amounts for all loans at the Perryridge branch.
  Solution:
  ```
  select customer-name, borrower.loan-number, amount
  from borrower, loan
  where borrower.loan-number = loan.loan-number and branch-name = 'Perryridge'
  ```

-

1.3 Rename operation

SQL provides a mechanism for renaming both relations and attributes. It uses the **as** clause, taking the form:
old-name as new-name
The **as** clause can appear in both the select and from clauses.

Example:

- select customer-name, borrower.loan-number **as** loan-id, amount
  from borrower, loan
  where borrower.loan-number = loan.loan-number

- For all customers who have a loan from the bank, find their names, loan numbers, and loan amount
  Solution:
  select customer-name, T.loan-number, S.amount
  from borrower **as** T, loan **as** S
  where T.loan-number = S.loan-number

- Find the names of all branches that have assets greater than at least one branch located in Brooklyn.
  Solution:
  select distinct T.branch-name
  from branch **as** T, branch **as**S
  where T.assets > S.assets and S.branch-city = 'Brooklyn'

1.4 String Operations

The most commonly used operation on strings is pattern matching using the operator like. We describe patterns by using two special characters:

- Percent (%): The % character matches any substring.
- Underscore (_): The _ character matches any character.

Patterns are case sensitive; that is, uppercase characters do not match lowercase characters, or vice versa. To illustrate pattern matching, we consider the following examples:

- 'Perry%' matches any string beginning with "Perry".
- '%idge%' matches any string containing "idge" as a substring, for example, 'Perryridge', 'Rock Ridge', 'Mianus Bridge', and 'Ridgeway'.
- '___' matches any string of exactly three characters.
- '___%' matches any string of at least three characters.

Example: Find the names of all customers whose street address includes the substring 'Main'.
Solution:
select customer-name
from customer
where customer-street like '%Main%'

1.5 Ordering the Display of Tuples

To display the result in the sorted order, we use the **order by** clause.
Example: To list in alphabetic order all customers who have a loan at the Perryridge branch
Solution:
```
select distinct customer-name
from borrower, loan
where borrower.loan-number = loan.loan-number and
branch-name = 'Perryridge'
order by customer-name
```

Example:
```
select *
from loan
order by amount desc, loan-number asc
```

1.6 Set Operations

The SQL operations union, intersect, and except operate on relations and correspond to the relational algebra operations $\cup$, $\cap$, and -.

- Find all customers having a loan, an account, or both at the bank.
  Solution:
  ```
  (select customer-name
  from depositor)
  union
  (select customer-name
  from borrower)
  ```
- Find all customers who have both a loan and an account at the bank.
  Solution:
  ```
  (select customer-name
  from depositor)
  intersect
  (select customer-name
  from borrower)
  ```
- Find all customers who have an account but no loan at the bank.
  Solution:
  ```
  (select customer-name
  from depositor)
  except
  (select customer-name
  from borrower)
  ```

1.7 Aggregate Functions

Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value. SQL offers five built-in aggregate functions:

1. Average: avg
2. Minimum: min
3. Maximum: max
4. Total: sum
5. Count: count

Example:

- Find the average account balance at the Perryridge branch.
  Solution:
  select avg (balance)
  from account
  where branch-name = 'Perryridge'

- Find the average account balance at each branch.
  Solution:
  select branch-name, avg (balance)
  from account
  group by branch-name

- Find the number of depositors for each branch.
  Solution:
  select branch-name, count (distinct customer-name)
  from depositor, account
  where depositor.account-number = account.account-number
  group by branch-name

- Find the branches where the average account balance is more than $1200.
  Solution:
  select branch-name, avg (balance)
  from account
  group by branch-name
  having avg (balance) > 1200

- Find the average balance for each customer who lives in Harrison and has at least three accounts.
  Solution:
  select depositor.customer-name, avg (balance)
  from depositor, account, customer

where depositor.account-number = account.account-number and depositor.customer-name = customer.customer-name and customer-city = ’Harrison’
group by depositor.customer-name
having count (distinct depositor.account-number) $\geq$ 3

1.8 Nested Subqueries

- Find all customers who have both a loan and an account at the bank.
  Solution:
  ```
  select distinct customer-name
  from borrower
  where customer-name in (select customer-name from depositor)
  ```

- Find all customers who have both an account and a loan at the Perryridge branch.
  Solution:
  ```
  select distinct customer-name
  from borrower, loan
  where borrower.loan-number = loan.loan-number and
  branch-name = ’Perryridge’ and (branch-name, customer-name)
                            in (select branch-name, customer-name
                            from depositor, account
                            where depositor.account-number = account.account-number)
  ```

- Find all customers who do have a loan at the bank, but do not have an account at the bank.
  Solution:
  ```
  select distinct customer-name
  from borrower
  where customer-name not in (select customer-name from depositor)
  ```

- Find the names of all branches that have assets greater than those of at least one branch located in Brooklyn.
  Solution:
  ```
  select branch-name
  from branch
  where assets > some (select assets
                      from branch
                      where branch-city = ’Brooklyn’)
  ```

- Find the names of all branches that have an asset value greater than that of each branch in Brooklyn.
  Solution:
  ```
  select branch-name
  from branch
  where assets > all (select assets
  ```

```
                                    from branch
                                    where branch-city = 'Brooklyn')
```

- Finds those branches for which the average balance is greater than or equal to all average balances.
  Solution:
  ```
  select branch-name
  from account
  group by branch-name
  having avg (balance) ≥ all (select avg (balance)
                              from account
                              group by branch-name)
  ```

1.9 Test for Empty Relations

The **exists** construct returns the value **true** if the argument subquery is nonempty.

Example: Find all customers who have both an account and a loan at the bank. **Solution:**

```
select customer-name
from borrower
where exists (select *
              from depositor
              where depositor.customer-name = borrower.customer-name)
```

We can test for the nonexistence of tuples in a subquery by using the not exists construct. We can use the not exists construct to simulate the set containment (that is, superset) operation: We can write "relation A contains relation B" as "not exists (B except A)."

Example: Find all customers who have an account at all the branches located in Brooklyn. **Solution:**

```
select distinct S.customer-name
from depositor as S
where not exists ((select branch-name
                   from branch
                   where branch-city = 'Brooklyn')
                   except
                   (select R.branch-name
                   from depositor as T, account as R
                   where T.account-number = R.account-number and
                   S.customer-name = T.customer-name))
```

1.10 Test for the Absence of Duplicate Tuples

SQL includes a feature for testing whether a subquery has any duplicate tuples in its result. The **unique** construct returns the value **true** if the argument subquery contains no duplicate tuples.

Example: Find all customers who have at most one account at the Perryridge branch.
Solution:

```
select T.customer-name
from depositor as T
where unique (select R.customer-name
              from account, depositor as R
              where T.customer-name = R.customer-name and
              R.account-number = account.account-number and
              account.branch-name = 'Perryridge')
```

We can test for the existence of duplicate tuples in a subquery by using the **not unique** construct.

Example: Find all customers who have at least two accounts at the Perryridge branch.
Solution:

```
select distinct T.customer-name
from depositor T
where not unique (select R.customer-name
                  from account, depositor as R
                  where T.customer-name = R.customer-name and
                  R.account-number = account.account-number and
                  account.branch-name = 'Perryridge')
```

1.11 Some other complex queries

- Find the average account balance of those branches where the average account balance is greater than $1200.
  Solution
  ```
  select branch-name, avg-balance
  from (select branch-name, avg (balance)
  from account
  group by branch-name)
  as branch-avg (branch-name, avg-balance)
  where avg-balance > 120
  ```

- Find the maximum across all branches of the total balance at each branch.
  Solution
  ```
  select max(tot-balance)
  from (select branch-name, sum(balance)
  from account
  group by branch-name) as branch-total (branch-name, tot-balance)
  ```

1.12 Example

Consider the following database schemas and corresponding its database:-

Sailors(sid: integer, sname: string, rating: integer, age: real)
Boats(bid: integer, bname: string, color: string)
Reserves(sid: integer, bid: integer, day: date)

Write the following queries in SQL:-

1. Find the names of sailors who have reserved boat number 103.
   Solution:
   ```
   SELECT S.sname
   FROM Sailors S, Reserves R
   WHERE S.sid = R.sid AND R.bid=103
   ```

2. Find all sailors with a rating above 7.
   Solution:
   ```
   SELECT S.sid, S.sname, S.rating, S.age
   FROM Sailors AS S
   WHERE S.rating > 7
   ```

3. Find the sids of sailors who have reserved a red boat.
   Solution:
   ```
   SELECT R.sid
   FROM Boats B, Reserves R
   WHERE B.bid = R.bid AND B.color = ‘red’
   ```

4. Find the names of sailors who have reserved a red boat.
   Solution:
   ```
   SELECT S.sname
   FROM Sailors S, Reserves R, Boats B
   WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = ‘red’
   ```

5. Find the colors of boats reserved by Lubber.

   Solution:
   ```
   SELECT B.color
   FROM Sailors S, Reserves R, Boats B
   WHERE S.sid = R.sid AND R.bid = B.bid AND S.sname = ‘Lubber’
   ```

6. Find the names of sailors who have reserved at least one boat.

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Gates	*null*	*null*	Redmond	5300

Employee ⟖ FT-works

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*
Gates	*null*	*null*	Redmond	5300

Employee ⟗ FT-works

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Instance S_3 of sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Instance R_2 of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Instance B_1 of Boats Reserves

Solution:
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid

7. Compute increments for the ratings of persons who have sailed two different boats on the same day.
Solution:
SELECT S.sname, S.rating+1 AS rating
FROM Sailors S, Reserves R1, Reserves R2
WHERE S.sid = R1.sid AND S.sid = R2.sid
AND R1.day = R2.day AND R1.bid <> R2.bid

8. Find the ages of sailors whose name begins and ends with B and has at least three characters.
Solution:
SELECT S.age
FROM Sailors S
WHERE S.sname LIKE 'B_%B'

9. Find the names of sailors who have reserved a red or a green boat. **Solution:**
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND (B.color = 'red' OR B.color = 'green')

This query can also be written as following:-
SELECT S.sname FROM Sailors S, Reserves R, Boats B WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red' UNION SELECT S2.sname FROM Sailors S2, Boats B2, Reserves R2 WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'

10. Find the names of sailors who have reserved both a red and a green boat. **Solution:**
SELECT S.sname
FROM Sailors S, Reserves R1, Boats B1, Reserves R2, Boats B2
WHERE S.sid = R1.sid AND R1.bid = B1.bid
AND S.sid = R2.sid AND R2.bid = B2.bid
AND B1.color='red' AND B2.color = 'green'

This query can also be written as following:-
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT S2.sname

```
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

11. Find the sids of all sailors who have reserved red boats but not green boats.

    Solution:
    ```
    SELECT S.sid
    FROM Sailors S, Reserves R, Boats B
    WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
    EXCEPT
    SELECT S2.sid
    FROM Sailors S2, Reserves R2, Boats B2
    WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
    ```

12. Find the names of sailors who have not reserved a red boat.
    Solution:
    ```
    SELECT S.sname
    FROM Sailors S
    WHERE S.sid NOT IN ( SELECT R.sid
                         FROM Reserves R
                         WHERE R.bid IN ( SELECT B.bid
                                          FROM Boats B
                                          WHERE B.color = 'red' ))
    ```

13. Find sailors whose rating is better than some sailor called Horatio.
    Solution:
    ```
    SELECT S.sid
    FROM Sailors S
    WHERE S.rating > ANY ( SELECT S2.rating
                           FROM Sailors S2
                           WHERE S2.sname = 'Horatio' )
    ```

14. Find the sailors with the highest rating.
    Solution:
    ```
    SELECT S.sid
    FROM Sailors S
    WHERE S.rating >= ALL ( SELECT S2.rating
                         FROM Sailors S2 )
    ```

15. Find the names of sailors who have reserved all boats.
    Solution:
    ```
    SELECT S.sname
    FROM Sailors S
    WHERE NOT EXISTS (( SELECT B.bid
    ```

```
                         FROM Boats B )
                         EXCEPT
                         (SELECT R.bid
                         FROM Reserves R
                         WHERE R.sid = S.sid ))
```

An alternative way to do this query without using EXCEPT follows:

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS ( SELECT B.bid
                   FROM Boats B
                   WHERE NOT EXISTS ( SELECT R.bid
                                      FROM Reserves R
                                      WHERE R.bid = B.bid
                                            AND R.sid = S.sid ))
```

16. Find the average age of all sailors.
    Solution:
    ```
    SELECT AVG (S.age)
    FROM Sailors S
    ```

17. Find the average age of sailors with a rating of 10.
    Solution:
    ```
    SELECT AVG (S.age)
    FROM Sailors S
    WHERE S.rating = 10
    ```

18. Find the name and age of the oldest sailor.
    Solution:
    ```
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.age = ( SELECT MAX (S2.age)
                    FROM Sailors S2 )
    ```

19. Count the number of different sailor names.
    Solution:
    ```
    SELECT COUNT ( DISTINCT S.sname )
    FROM Sailors S
    ```

20. Find the names of sailors who are older than the oldest sailor with a rating of 10.
    Solution:
    ```
    SELECT S.sname
    FROM Sailors S
    ```

```
WHERE S.age > ( SELECT MAX ( S2.age )
                FROM Sailors S2
                WHERE S2.rating = 10 )
```

This query could alternatively be written as follows:

```
SELECT S.sname
FROM Sailors S
WHERE S.age > ALL ( SELECT S2.age
                    FROM Sailors S2
                    WHERE S2.rating = 10 )
```

21. Find the age of the youngest sailor for each rating level.
    Solution:
    ```
    SELECT S.rating, MIN (S.age)
    FROM Sailors S
    GROUP BY S.rating
    ```

22. Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.
    Solution:
    ```
    SELECT S.rating, MIN (S.age) AS minage
    FROM Sailors S
    WHERE S.age >= 18
    GROUP BY S.rating
    HAVING COUNT (*) > 1
    ```

23. For each red boat, find the number of reservations for this boat.
    Solution:
    ```
    SELECT B.bid, COUNT (*) AS sailorcount
    FROM Boats B, Reserves R
    WHERE R.bid = B.bid AND B.color = ‘red’
    GROUP BY B.bid
    ```

24. Find the average age of sailors for each rating level that has at least two sailors.
    Solution:
    ```
    SELECT S.rating, AVG (S.age) AS average
    FROM Sailors S
    GROUP BY S.rating
    HAVING COUNT (*) > 1
    ```

25. Find the average age of sailors who are of voting age (i.e., at least 18 years old) for each rating level that has at least two sailors.
    Solution:
    ```
    SELECT S.rating, AVG ( S.age ) AS average
    FROM Sailors S
    WHERE S. age >= 18
    GROUP BY S.rating
    HAVING 1 < ( SELECT COUNT (*)
    ```

```
                FROM Sailors S2
                WHERE S.rating = S2.rating )
```

26. Find the average age of sailors who are of voting age (i.e., at least 18 years old) for each rating level that has at least two such sailors.
Solution:
```
SELECT S.rating, AVG ( S.age ) AS average
FROM Sailors S
WHERE S. age > 18
GROUP BY S.rating
HAVING 1 < ( SELECT COUNT (*)
                FROM Sailors S2
                WHERE S.rating = S2.rating AND S2.age >= 18 )
```

27. Find those ratings for which the average age of sailors is the minimum over all ratings.
Solution:
```
SELECT S.rating
FROM Sailors S
WHERE AVG (S.age) = ( SELECT MIN (AVG (S2.age))
                      FROM Sailors S2
                      GROUP BY S2.rating )
```

1.13 Cursor

We can declare a cursor on any relation or on any SQL query (because every query returns a set of rows). Once a cursor is declared, we can **open** it (which positions the cursor just before the first row); **fetch** the next row; **move** the cursor (to the next row, to the row after the next n, to the first row, or to the previous row, etc., by specifying additional parameters for the **FETCH** command); or **close** the cursor. Thus, a cursor essentially allows us to retrieve the rows in a table by positioning the cursor at a particular row and reading its contents.

1.14 TRIGGERS AND ACTIVE DATABASES

A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an active database. A trigger description contains three parts:

Event: A change to the database that activates the trigger.
Condition: A query or test that is run when the trigger is activated.
Action: A procedure that is executed when the trigger is activated and its condition is true.
A trigger can be thought of as a 'daemon' that monitors a database, and is executed when the database is modified in a way that matches the event specification. An insert, delete or update statement could activate a trigger, regardless of which user or application invoked the activating statement; users may not even be aware that a trigger was

executed as a side effect of their program.
A condition in a trigger can be a true/false statement (e.g., all employee salaries are less than $100,000) or a query. A query is interpreted as true if the answer set is nonempty, and false if the query has no answers. If the condition part evaluates to true, the action associated with the trigger is executed.
A trigger action can examine the answers to the query in the condition part of the trigger, refer to old and new values of tuples modified by the statement activating the trigger, execute new queries, and make changes to the database. In fact, an action can even execute a series of data-definition commands (e.g., create new tables, change authorizations) and transaction-oriented commands (e.g., commit), or call host language procedures.

1.15 Exercise

1. Consider the following employee database:-

   employee (employee-name, street, city)
   works (employee-name, company-name, salary)
   company (company-name, city)
   manages (employee-name, manager-name)

   where the primary keys are underlined. Give an expression in SQL for each of the following queries.

   (a) Find the names of all employees who work for First Bank Corporation.
   (b) Find the names and cities of residence of all employees who work for First Bank Corporation.
   (c) Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.
   (d) Find all employees in the database who live in the same cities as the companies for which they work.
   (e) Find all employees in the database who live in the same cities and on the same streets as do their managers.
   (f) Find all employees in the database who do not work for First Bank Corporation.
   (g) Find all employees in the database who earn more than each employee of Small Bank Corporation.
   (h) Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.
   (i) Find all employees who earn more than the average salary of all employees of their company.
   (j) Find the company that has the most employees.
   (k) Find the company that has the smallest payroll.
   (l) Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

Solution:

(a) select employee-name
from works
where company-name = 'First Bank Corporation'

(b) select e.employee-name, city
from employee e, works w
where w.company-name = 'First Bank Corporation' and w.employee-name = e.employee-name

(c) select *
from employee
where employee-name in
(select employee-name
from works
where company-name = 'First Bank Corporation' and salary > 10000)

(d) select e.employee-name
from employee e, works w, company c
where e.employee-name = w.employee-name and e.city = c.city and w.company-name = c.company-name

(e) select P.employee-name
from employee P, employee R, manages M
where P.employee-name = M.employee-name and M.manager-name = R.employee-name and P.street = R.street and P.city = R.city

(f) select employee-name
from works
where company-name ≠ 'First Bank Corporation'

(g) select employee-name
from works
where salary > all
(select salary
from works
where company-name = 'Small Bank Corporation')

(h) select S.company-name
from company S
where not exists ((select city
from company
where company-name = 'Small Bank Corporation')
except
(select city
from company T
where S.company-name = T.company-name))

(i) select employee-name
from works T
where salary > (select avg (salary)

from works S
where T.company-name = S.company-name)

(j) select company-name
from works
group by company-name
having count (distinct employee-name) >= all
(select count (distinct employee-name)
from works
group by company-name)

(k) select company-name
from works
group by company-name
having sum (salary) <= all (select sum (salary)
from works
group by company-name)

(l) select company-name
from works
group by company-name
having avg (salary) > (select avg (salary)
from works
where company-name = 'First Bank Corporation')

2. Consider the employee database of the previous questions. Give an expression in SQL for each of the following queries.

(a) Modify the database so that Jones now lives in Newtown.

(b) Give all employees of First Bank Corporation a 10 percent raise.

(c) Give all managers of First Bank Corporation a 10 percent raise.

(d) Give all managers of First Bank Corporation a 10 percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3 percent raise.

(e) Delete all tuples in the works relation for employees of Small Bank Corporation.

Solution:

(a) update employee
set city = 'Newton'
where person-name = 'Jones'

(b) update works
set salary = salary * 1.1
where company-name = 'First Bank Corporation'

(c) update works
set salary = salary * 1.1
where employee-name in (select manager-name from manages)

and company-name = 'First Bank Corporation'

(d) update works
set salary = salary * 1.03
where employee-name in (select manager-name from manages)
and salary * 1.1 > 100000
and company-name = 'First Bank Corporation'

update works
set salary = salary * 1.1
where employee-name in (select manager-name from manages)
and salary * 1.1 <= 100000
and company-name = 'First Bank Corporation'

(e) delete works
where company-name = 'Small Bank Corporation'

3. Consider the following employee database:-

person (driver-id, name, address)
car (license, model, year)
accident (report-number, date, location)
owns (driver-id, license)
participated (driver-id, license, report-number, damage-amount)

where the primary keys are underlined. Construct the following SQL queries for this relational database.

(a) Find the total number of people who owned cars that were involved in accidents in 1989.
(b) Find the number of accidents in which the cars belonging to "John Smith" were involved.
(c) Add a new accident to the database; assume any values for required attributes.
(d) Delete the Mazda belonging to "John Smith".
(e) Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

Solution:

(a) select count (distinct name)
from accident, participated, person
where accident.report-number = participated.report-number
and participated.driver-id = person.driver-id
and date between date '1989-00-00' and date '1989-12-31'

(b) select count (distinct *)
from accident

```
where exists
          (select *
          from participated, person
          where participated.driver-id = person.driver-id
          and person.name = 'John Smith'
          and accident.report-number = participated.report-number)
```

(c) We assume the driver was "Jones," although it could be someone else. Also, we assume "Jones" owns one Toyota. First we must find the license of the given car. Then the participated and accident relations must be updated in order to both record the accident and tie it to the given car. We assume values "Berkeley" for location, '2001-09-01' for date and date, 4007 for reportnumber and 3000 for damage amount.

```
insert into accident
values (4007, '2001-09-01', 'Berkeley')

insert into participated
select o.driver-id, c.license, 4007, 3000
from person p, owns o, car c
where p.name = 'Jones' and p.driver-id = o.driver-id and
o.license = c.license and c.model = 'Toyota'
```

(d)
```
delete car
where model = 'Mazda' and license in
                    (select license
                    from person p, owns o
                    where p.name = 'John Smith' and
                         p.driver-id = o.driver-id)
```

(e)
```
update participated
set damage-amount = 3000
where report-number = "AR2197" and driver-id in
                         (select driver-id
                         from owns
                         where license = "AABB2000")
```

4. Consider the following relations:

Student(snum: integer, sname: string, major: string, level: string, age: integer)
Class(cname: string, meets at: time, room: string, fid: integer)
Enrolled(snum: integer, cname: string)
Faculty(fid: integer, fname: string, deptid: integer)

Write the following queries in SQL. No duplicates should be printed in any of the answers.

(a) Find the names of all Juniors (Level = JR) who are enrolled in a class taught by I. Teach.

(b) Find the age of the oldest student who is either a History major or is enrolled in a course taught by I. Teach.

(c) Find the names of all classes that either meet in room R128 or have five or more students enrolled.

(d) Find the names of all students who are enrolled in two classes that meet at the same time.

(e) Find the names of faculty members who teach in every room in which some class is taught.

(f) Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.

(g) Print the Level and the average age of students for that Level, for each Level.

(h) Print the Level and the average age of students for that Level, for all Levels except JR.

(i) Find the names of students who are enrolled in the maximum number of classes.

(j) Find the names of students who are not enrolled in any class.

(k) For each age value that appears in Students, find the level value that appears most often.
For example, if there are more FR level students aged 18 than SR, JR, or SO students aged 18, you should print the pair (18, FR).

Solution:

(a)
```
SELECT DISTINCT S.Sname
FROM Student S, Class C, Enrolled E, Faculty F
WHERE S.snum = E.snum AND E.cname = C.name AND C.fid = F.fid AND
F.fname = ‘I.Teach’ AND S.level = ‘JR’
```

(b)
```
SELECT MAX(S.age)
FROM Student S
WHERE (S.major = ‘History’)
OR S.snum IN (SELECT E.snum
FROM Class C, Enrolled E, Faculty F
WHERE E.cname = C.name AND C.fid = F.fid
AND F.fname = ‘I.Teach’ )
```

(c)
```
SELECT C.name
FROM Class C
WHERE C.room = ‘R128’
OR C.name IN (SELECT E.cname
FROM Enrolled E
GROUP BY E.cname
HAVING COUNT (*) >= 5)
```

(d) SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum IN (SELECT E1.snum
FROM Enrolled E1, Enrolled E2, Class C1, Class C2
WHERE E1.snum = E2.snum AND E1.cname <> E2.cname
AND E1.cname = C1.name
AND E2.cname = C2.name AND C1.meets at = C2.meets at)

(e) SELECT DISTINCT F.fname
FROM Faculty F
WHERE NOT EXISTS ((SELECT *
FROM Class C)
EXCEPT
(SELECTC1.room
FROM Class C1
WHERE C1.fid = F.fid))

(f) SELECT DISTINCT F.fname
FROM Faculty F
WHERE 5 > (SELECT COUNT (E.snum)
FROM Class C, Enrolled E
WHERE C.name = E.cname
AND C.fid = F.fid)

(g) SELECT S.level, AVG(S.age)
FROM Student S
GROUP BY S.level

(h) SELECT S.level, AVG(S.age)
FROM Student S
WHERE S.level ¡¿ 'JR'
GROUP BY S.level

(i) SELECT F.fname, COUNT(*) AS CourseCount
FROM Faculty F, Class C
WHERE F.fid = C.fid
GROUP BY F.fid, F.fname
HAVING EVERY (C.room = 'R128')

(j) SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum IN (SELECT E.snum
FROM Enrolled E
GROUP BY E.snum
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Enrolled E2

```
GROUP BY E2.snum ))
```

(k)
```
SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum NOT IN (SELECT E.snum
FROM Enrolled E )
```

(l)
```
SELECT S.age, S.level
FROM Student S
GROUP BY S.age, S.level,
HAVING S.level IN (SELECT S1.level
FROM Student S1
WHERE S1.age = S.age
GROUP BY S1.level, S1.age
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Student S2
WHERE s1.age = S2.age
GROUP BY S2.level, S2.age))
```

5. Consider the following schema:

   Suppliers(sid: integer, sname: string, address: string)
   Parts(pid: integer, pname: string, color: string)
   Catalog(sid: integer, pid: integer, cost: real)

   The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in SQL:

   (a) Find the pnames of parts for which there is some supplier.
   (b) Find the snames of suppliers who supply every part.
   (c) Find the snames of suppliers who supply every red part.
   (d) Find the pnames of parts supplied by Acme Widget Suppliers and by no one else.
   (e) Find the sids of suppliers who charge more for some part than the average cost of that part (averaged over all the suppliers who supply that part).
   (f) For each part, find the sname of the supplier who charges the most for that part.
   (g) Find the sids of suppliers who supply only red parts.
   (h) Find the sids of suppliers who supply a red part and a green part.
   (i) Find the sids of suppliers who supply a red part or a green part.

   Solution:

(a)
```
SELECT pname
FROM Parts, Catalog
WHERE Parts.pid = Catalog.pid
```

(b)
```
SELECT Sname
FROM Suppliers
WHERE NOT EXISTS ( SELECT pid
                   FROM Part)
                   EXCEPT
                   ( SELECT pid
                   FROM Catalog
                   WHERE Suppliers.sid = Catalog.sid)
```

(c)
```
SELECT Sname
FROM Suppliers
WHERE NOT EXISTS ( SELECT pid
                   FROM Part
                   WHERE color = 'red')
                   EXCEPT
                   ( SELECT pid
                   FROM Catalog
                   WHERE Suppliers.sid = Catalog.sid)
```

(d)
```
SELECT pname FROM Parts, Catalog, Suppliers
WHERE Parts.pid = Catalog.pid AND Catalog.sid = Suppliers.sid AND
        sname = 'Acme Widget' AND
        pid NOT IN (SELECT pid
                    FROM Catalog, Suppliers
                    WHERE Catalog.sid = Suppliers.sid AND
                            sname <> 'Acme Widget')
```

(e)
```
SELECT sid
FROM Catalog
WHERE cost > (SELECT avg(cost)
              FROM Catalog as T
              WHERE Catalog.pid = T.pid)
```

(f)
```
SELECT pid, sname
FROM Suppliers, Catalog
WHERE Suppliers.sid = Catalog.sid AND cost = (SELECT max(cost)
                                             FROM Catalog as T
                                             WHERE Catalog.pid = T.pid)
```

(g)
```
SELECT sid
FROM Catalog, Parts
WHERE Catalog.pid = Parts.pid AND
        NOT EXISTS ((SELECT pid
                     FROM Parts as P
                     WHERE Parts.pid = P.pid)
                     EXCEPT
                     (SELECT pid
                    FROM Parts
                    WHERE color = 'red'))
```

(h)
```
(SELECT sid
FROM Catalog, Parts
WHERE Catalog.pid = Parts.pid AND color = 'red')
INTERSECT
(SELECT sid
FROM Catalog, Parts
WHERE Catalog.pid = Parts.pid AND color = 'green')
```

(i)
```
(SELECT sid
FROM Catalog, Parts
WHERE Catalog.pid = Parts.pid AND color = 'red')
UNION
(SELECT sid
FROM Catalog, Parts
WHERE Catalog.pid = Parts.pid AND color = 'green')
```

6. The following relations keep track of airline flight information:

   Flights(<u>flno: integer</u>, from: string, to: string, distance: integer, departs: time, arrives: time, price: integer)
   Aircraft(<u> aid: integer</u>, aname: string, cruisingrange: integer)
   Certified(<u> eid: integer, aid: integer</u>)
   Employees(<u> eid: integer</u>, ename: string, salary: integer)

   Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly. Write each of the following queries in SQL.

   (a) Find the names of aircraft such that all pilots certified to operate them earn more than 80,000.
   (b) For each pilot who is certified for more than three aircraft, find the eid and the maximum cruisingrange of the aircraft that he (or she) is certified for.
   (c) Find the names of pilots whose salary is less than the price of the cheapest route from Los Angeles to Honolulu.
   (d) For all aircraft with cruisingrange over 1,000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.
   (e) Find the names of pilots certified for some Boeing aircraft.
   (f) Find the aids of all aircraft that can be used on routes from Los Angeles to Chicago.
   (g) Identify the flights that can be piloted by every pilot who makes more than$100,000. (Hint: The pilot must be certified for at least one plane with a sufficiently large cruisingrange.)
   (h) Print the enames of pilots who can operate planes with cruisingrange greater than 3,000 miles, but are not certified on any Boeing aircraft.

(i) A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New York by 6 p.m.

(j) Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).

(k) Print the name and salary of every nonpilot whose salary is more than the average salary for pilots.

Solution:

(a)
```
SELECT DISTINCT A.aname
FROM Aircraft A
WHERE A.Aid IN (SELECT C.aid
                FROM Certified C, Employees E
                WHERE C.eid = E.eid AND
                NOT EXISTS ( SELECT *
                          FROM Employees E1
                          WHERE E1.eid = E.eid AND E1.salary < 80000))
```

(b)
```
SELECT C.eid, MAX (A.cruisingrange)
FROM Certified C, Aircraft A
WHERE C.aid = A.aid
GROUP BY C.eid
HAVING COUNT (*) > 3
```

(c)
```
SELECT DISTINCT E.ename
FROM Employees E
WHERE E.salary < ( SELECT MIN (F.price)
               FROM Flights F
               WHERE F.from = ‘Los Angeles’ AND F.to = ‘Honolulu’ )
```

(d)
```
SELECT Temp.name, Temp.AvgSalary
FROM ( SELECT A.aid, A.aname AS name, AVG (E.salary) AS AvgSalary
       FROM Aircraft A, Certified C, Employees E
       WHERE A.aid = C.aid AND C.eid = E.eid
             AND A.cruisingrange > 1000
       GROUP BY A.aid, A.aname ) AS Temp
```

(e)
```
SELECT DISTINCT E.ename
FROM Employees E, Certified C, Aircraft A
WHERE E.eid = C.eid AND C.aid = A.aid AND A.aname LIKE ‘Boeing
```

(f)
```
SELECT A.aid
FROM Aircraft A
WHERE A.cruisingrange > ( SELECT MIN (F.distance)
                      FROM Flights F
                      WHERE F.from = ‘Los Angeles’ AND F.to = ‘Chicago’
)
```

(g)
```
SELECT DISTINCT F.from, F.to
FROM Flights F
WHERE NOT EXISTS ( SELECT *
                   FROM Employees E
                   WHERE E.salary > 100000 AND
                   NOT EXISTS (SELECT *
                               FROM Aircraft A, Certified C
                               WHERE A.cruisingrange > F.distance
                               AND E.eid = C.eid AND A.aid = C.aid))
```

(h)
```
SELECT DISTINCT E.ename
FROM Employees E
WHERE E.eid IN ( SELECT C.eid
                 FROM Certified C
                 WHERE EXISTS ( SELECT A.aid
                                FROM Aircraft A
                                WHERE A.aid = C.aid AND
                                      A.cruisingrange > 3000 )
                      AND NOT EXISTS ( SELECT A1.aid
                                       FROM Aircraft A1
                                       WHERE A1.aid = C.aid
                                       AND A1.aname LIKE 'Boeing%'))
```

(i)
```
SELECT F.departs
FROM Flights F
WHERE F.flno IN ( ( SELECT F0.flno
                  FROM Flights F0
                  WHERE F0.from = 'Madison' AND F0.to = 'New York'
                        AND F0.arrives < '18:00' )
                  UNION
                ( SELECT F0.flno
                FROM Flights F0, Flights F1
                WHERE F0.from = 'Madison' AND F0.to <> 'New York'
                      AND F0.to = F1.from AND F1.to = 'New York'
                    AND F1.departs > F0.arrives AND F1.arrives < '18:00'
)
                  UNION
                  ( SELECT F0.flno
                  FROM Flights F0, Flights F1, Flights F2
                  WHERE F0.from = 'Madison' AND F0.to = F1.from
                        AND F1.to = F2.from AND F2.to = 'New York'
                        AND F0.to <> 'New York'
                        AND F1.to <> 'New York'
                        AND F1.departs > F0.arrives
                        AND F2.departs> F1.arrives
                        AND F2.arrives < '18:00' ))
```

(j)
```
SELECT Temp1.avg - Temp2.avg
FROM (SELECT AVG (E.salary) AS avg
          FROM Employees E
          WHERE E.eid IN (SELECT DISTINCT C.eid
                                  FROM Certified C )) AS Temp1,
          (SELECT AVG (E1.salary) AS avg
          FROM Employees E1 ) AS Temp2
```

(k)
```
SELECT E.ename, E.salary
FROM Employees E
WHERE E.eid NOT IN ( SELECT DISTINCT C.eid
                                  FROM Certified C )
          AND E.salary > ( SELECT AVG (E1.salary)
                                  FROM Employees E1
                                  WHERE E1.eid IN ( SELECT DISTINCT C1.eid
                                                             FROM Certified C1 ) )
```

7. Consider the following relational schema. An employee can work in more than one department; the pct time field of the Works relation shows the percentage of time that a given employee works in a given department.

   Emp(eid: integer, ename: string, age: integer, salary: real)
   Works(eid: integer, did: integer, pct time: integer)
   Dept(did: integer, budget: real, managerid: integer)

   Write the following queries in SQL:

   (a) Print the names and ages of each employee who works in both the Hardware department and the Software department.
   (b) For each department with more than 20 full-time-equivalent employees (i.e., where the part-time and full-time employees add up to at least that many full-time employees), print the did together with the number of employees that work in that department.
   (c) Print the name of each employee whose salary exceeds the budget of all of the departments that he or she works in.
   (d) Find the managerids of managers who manage only departments with budgets greater than $1,000,000.
   (e) Find the enames of managers who manage the departments with the largest budget.
   (f) If a manager manages more than one department, he or she controls the sum of all them budgets for those departments. Find the managerids of managers who control more than $5,000,000.
   (g) Find the managerids of managers who control the largest amount.

Solution:

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)