Chapter 2

Structured Query Language

LEARNING OBJECTIVES

- *Relational algebra*
- *Select operator*
- *Project operator*
- *Set operators*
- *Union compatible relations*
- *Union operation*
- *Aggregate operators*
- *Correlated nested queries*
- *Relational calculus*
- *Tuple relational calculus*
- *Tuple relational calculus*
- *DML*
- *Super key*
- *SQL commands*

Relational Algebra

1. A set of operators (unary or binary) that take relation instances as arguments and return new relations.
2. Gives a procedural method of specifying a retrieval query
3. Forms the core component of a relational query engine
4. *SQL* queries are internally translated into *RA* expressions
5. Provides a framework for query optimization

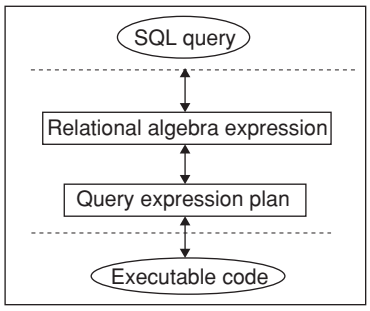
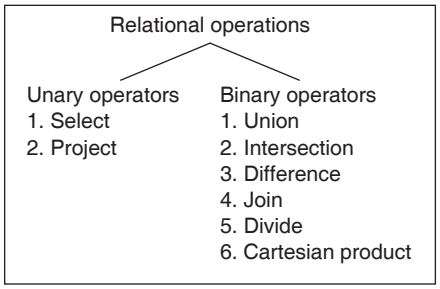


Figure 1 *Role of relational algebra in DBMS:*

Relational Operations

A collection of simple 'low-level' operations used to manipulate relations.

1. It provides a procedural way to query a database.
2. Input is one (or) more relations.
3. Output is one relation.



Select Operator (σ)

Select operator is an unary operator. It can be used to select those tuples of a relation that satisfy a given condition.

Notation: $\sigma_\theta(r)$

σ: Select operator(read as sigma)

θ: Selection condition

r: Relation name

Result is a relation with the same scheme as r consisting of the tuples in r that satisfy condition θ

Syntax: $\sigma_{\text{condition}}$ (relation)

Example:

Table 2.1 *Person*

Id	Name	Address	Hobby
112	John	12, SP Road	Stamp collection
113	John	12, SP Road	Coin collection
114	Mary	16, SP Road	Painting
115	Brat	18, GP Road	Stamp collection

$\sigma_{\text{Hobby = 'stamp. Collection'}}$(person)

The above given statement displays all tuples (or) records with hobby 'stamp collection'.

Output:

Id	Name	Address	Hobby
112	John	12, SP Road	Stamp collection
115	Brat	18, GP Road	Stamp collection

Selection condition can use following operators:
<, ≤, >, ≥, =, ≠

1. <attribute> operator <attribute>
2. <attribute> operator <constant>

Example: Salary ≥ 1000

3. <Condition> AND/OR <condition>

Example: (Experience > 3) AND (Age < 58)

4. NOT <condition>

Selection operation examples:

1. $\sigma_{\text{Id > 112 OR Hobby = 'paint'}}$ (person)
   It displays the tuples whose ID > 112 or Hobby is paint
2. $\sigma_{\text{Id > 112 AND Id < 115}}$ (person)
   It displays tuples whose ID is greater than 112 and less than 115
3. $\sigma_{\text{NOT (hobby = 'paint')}}$ (person)
   It displays tuples whose hobby is not paint
4. $\sigma_{\text{Hobby} \neq \text{'paint'}}$ (person)
   It displays tuples whose hobby is not paint, displays all tuples other than hobby paint.
   Selection operator: Produces table containing subset of rows of argument table which satisfies condition.

Project Operator (π)

The project operator is unary operator. It can be used to keep only the required attributes of a relation instance and throw away others.

Notation: $\pi_{A2, A2, \ldots Ak(r)}$ Where $A_1, A_2, \ldots A_K$ is a list L of desired attributes in the scheme of r.

Result = $\{(V_1, V_2, \ldots V_K)/V_i \in \text{DOM}(A_i), 1 \leq i \leq k$ and there is some tuple t in r, such that $t.A_1 = v_1, t.A_2 = v_2, \ldots t.A_K = V_K\}$
$\pi_{\text{Attribute List}}$ (Relation)

Take table 2.1 as reference.

1. π_{Name} (person)

Output:

Name
John
Mary
Bart

In the output table, John name has appeared once, project operation eliminated duplicates.

2. $\pi_{\text{Name, address}}$ (person)

Output:

Name	Address
John	12, SP Road
Mary	16, SP Road
Bart	18,GP Road

Expressions:

$\pi_{\text{id,name}}$ ($\sigma_{\text{hobby = 'stamp collection' OR Hobby= 'coin collection'}}$(person))

Output:

Id	Name
112	John
115	Bart

The above given relational algebra expression gives Ids, names of a person whose hobby is either stamp collection (or) coin collection.

Set Operators

Union (∪), Intersection (∩), set difference (–) are called *set operators*. Result of combining two relations with a set operator is a relation ⇒ all its elements must be tuples having same structure. Hence scope of set operations is limited to union compatible relations.

Union Compatible Relations

Two relations are union compatible if

1. Both have same number of columns
2. Names of attributes are same
3. Corresponding fields have same type
4. Attributes with the same name in both relations have same domain.
5. Union compatible relations can be combined using Union, Intersection, and set difference.

Example:

Consider the given tables.
Person (SSN, Name, Address, Hobby)
Professor (Id, Name, office, phone)
person and professor tables are not union compatible.

Union

The result of union will be a set consisting of all tuples appearing in either or both of the given relations. Relations cannot contain a mixture of different kinds of tuples, they must be 'tuple – homogeneous'. The union in the relational algebra is not the completely general mathematical union; rather, it is a special kind of union, in which we require the two input relations to be of the same type.

The general definition of relational union operator:

Given are two relations '*a*' and '*b*' of the same type. The union of those two relations, a union *b*, is a relation of the same type, with body consisting of all tuples '*t*' such that '*t*' appears in a or *b* or both.

* Union operation eliminates duplicates.

Here is a different but equivalent definition:

Given are two relations '*a*' and '*b*' of the same type. The union of those two relations, *a* union *b*, is a relation of the same type, with body consisting of all tuples t such that t is equal to (i.e., is a duplicate of) some tuple in *a* or *b* or both.

Union Operation (U)

When union operation is applied on two tables it gives all the tuples in both without Repetition.

Example:

Table 2 *Result of union operation*

R

Roll. no.	Name	Semester	Percentage
22	Arun	7	45%
31	Bindu	6	55%
58	Sita	5	35%

S

Roll no.	Name	Semester	Percentage
28	Suresh	4	65%
31	Bindu	6	55%
44	Pinky	4	75%
58	Sita	5	35%

$R \cup S$

Roll no.	Name	Semester	Percentage
22	Arun	7	45%
31	Bindu	6	55%
58	Sita	5	35%
44	Pinky	4	75%
28	Sita	5	35%

Intersection

Like union, Intersection operator requires its operands to be of the same type. Given are two relations *a* and *b* of the same type, then, the intersection of those two relations, '*a*' INTERSECT '*b*', is a relation of the same type, with body consisting of all tuples t such that t appears in both '*a*' and '*b*'.

Intersection operation returns tuples which are common to both tables

Table 3 *Result of intersection operation*

$R \cap S =$

Roll no.	Name	Semester	Percentage
31	Bindu	6	55%
58	Sita	5	35%

Difference

Like union and intersection, the relational difference operator also requires its operands to be of the same type. Given are two relations '*a*' and '*b*' of the same type, Then, the difference between those two relations, '*a*' MINUS '*b*' (in that order), is a relation of the same type, with body consisting of all types t such that t appears in a and not *b*.

1. MINUS has a directionality to it, just as subtraction does in ordinary arithmetic (e.g., '6 – 3' and '3 – 6' are not the same thing)
2. Redundant duplicate rows are always eliminated from the result of UNION, INTERSECTION, EXCEPT operations.
3. SQL also provides the qualified variants UNION ALL, INTERSECT ALL and EXCEPT ALL, where duplicates are retained

Set difference operation returns the tuples in the first table which are not matching with the tuples of other table.

Table 4 *Result of R – S*

R – S =

Roll no.	Name	Semester	Percentage
22	Arun	7	45%

Table 5 *Result of S – R*

S – R =

Roll no.	Name	Semester	Percentage
28	Suresh	4	65%
44	Pinky	4	75%

* R – S ≠ S –R (both are different)

Example:

A

Supplier number	Supplier name	Status	City
SN1	MAHESH	40	HYDERABAD
SN3	SURESH	40	HYDERABAD

B

Supplier number	Supplier number	Status	City
SN3	SURESH	40	HYDERABAD
SN4	RAMESH	30	CHENNAI

UNION ($A \cup B$)

Supplier number	Supplier name	Status	City
SN1	MAHESH	40	HYDERABAD
SN3	SURESH	40	HYDERABAD
SN4	RAMESH	30	CHENNAI

INTERSECTION ($A \cap B$)

Supplier number	Supplier name	Status	City
SN3	SURESH	40	HYDERABAD

DIFFERENCE ($A - B$)

Supplier name	Supplier name	Status	City
SN1	MAHESH	40	HYDERABAD

DIFFERENCE ($B - A$)

Supplier name	Supplier name	Status	City
SN4	RAMESH	30	CHENNAI

Cartesian Product

The Cartesian product of two sets is the set of all ordered pairs such that in each pair, the first element comes from the first set and the second element comes from second set.

The result consists of all the attributes from both of the two input headings. We define the Cartesian product of two relations '*a*' and '*b*', as

'*a*' times '*b*', where *a* and *b* have no common attribute names (If we need to construct the Cartesian product of two relations that do have any such common attribute names, therefore, we must use the RENAME operator first to rename attributes appropriately).

The Cartesian product operation is also known as CROSS PRODUCT. This is also a binary set operation, but the relations on which it is applied need not to be union compatible. This operation is used to combine tuples from two relations in a combinational fashion.

Example:

R

A	B
X_1	X_2
X_3	X_4

S

C	D
Y_1	Y_2
Y_3	Y_4

$R \times S =$

A	B	C	D
X_1	X_2	Y_1	Y_2
X_1	X_2	Y_3	Y_4
X_3	X_4	Y_1	Y_2
X_3	X_4	Y_3	Y_4

Example: Transcript (StuId, coursecode, semester, grade) Teaching (ProfId, coursecode, semester)

$$\pi_{\text{stuId,coursecode}} (\text{Transcript}) \times \pi_{\text{profId,coursecode}} (\text{Teaching})$$

The above expression returns

Table 6 *Result of cross product*

Stu Id	Course code	Prof Id	Course code
...	...	...	...

Aggregate Operators

SQL Supports the usual aggregate operators COUNT, SUM, AVG, MAX, MIN, EVERY and ANY, but there are a few SQL-specific points.

1. The argument can optionally be preceded by the keyword DISTINCT, for example SUM (DISTINCT column -name) to indicate that duplicates are to be eliminated before the aggregation is done. For MAX, MIN, EVERY and ANY, however, DISTINCT has no effect and should not be specified.
2. The operator COUNT (*), for this DISTINCT is not allowed, and is provided to count all rows in a table without any duplicate elimination.
3. Any NULLS in the argument column are eliminated before the aggregation is done, regardless of whether DISTINCT is specified, except in the case of COUNT (*), where nulls behave as if they were values.
4. After NULLS if any have been eliminated, if what is left is an empty set, COUNT returns zero. The other operators return NULL.

AVG, MIN, MAX, SUM, COUNT

These functions operate on the multiset of values of column of a relation and returns a value.

1. Find the average account balance at the Perryridge branch.

Solution: SELECT AVG (balance) FROM account WHERE branch.name = 'perryridge'

2. Find the number of tuples in customer relation.

Solution: SELECT count (*) FROM customer

3. Find the number of depositors for each branch.

Solution: SELECT branch.name, COUNT (distinct customer-name) FROM depositor, account WHERE depositor. account-no = account account-no GROUPBY branch-name.

Nested Queries

Some queries require that existing values in the database be fetched and then used in a comparison condition.

Such queries can be conveniently formulated by using nested queries, which are complete SELECT – FROM –WHERE blocks within the WHERE clause of another query. The other query is called the *outer query*.

In-Comparison Operator

The comparison operator IN, which compares a value '*v*' with a set of values '*V*' and evaluates to TRUE if '*v*' is one of the elements in *V*.

Example: Consider the given database scheme and the statement:

EMPLOYEE

FNAME	INITIAL	LNAME	ENO	DOB	ADDRESS	SALARY	DNO

DEPARTMENT

D NAME	DNO	MANAGER-NO

DEPARTMENT–LOCATIONS

DNO	D-LOCATION

PROJECT

PNAME	PNO	P-LOCATION	DNO

WORKS–ON

ENO	PNO	HOURS

Example: Select distinct PNO from project where PNO IN (select PNO from project, department, employee where P. DNO = D. DNO AND MANAGER. NO = ENO AND LNAME = 'RAMYA')

The first query selects. The project numbers that have a 'Ramya' involved as manager, while the second selects the project numbers of projects that have a 'Ramya' involved as worker.

If a nested returns a single value, in such cases, it is permissible to use = instead of IN for the comparison operator.

In general, the nested query will return a table, Which is a set of multiset of tuples.

* SQL allows the use of tuples of values in comparisons by placing them within parentheses.

Example: SELECT DISTINCT ENO FROM WORKS - ON WHERE (PNO, HOURS) IN (SELECT PNO, HOURS FROM WORKS - ON WHERE ENO = 929).

This query will select the employee numbers of all employees who work on the same (PROJECT, HOURS) combination on some project a particular employee whose ENO = '929' works on. In this example, the IN operator compares the subtuple of values in parentheses (PNO, HOURS) for each tuple in works on with the set of union-compatible tuples produced by the nested query.

Correlated nested queries: Nested queries can be evaluated by executing the sub query (or) Inner query once and substituting the resulting value (or) values into the WHERE clause of the outer query.

1. In co-related nested queries, the inner query depends on the outer query for its value.
2. Sub-query is executed repeatedly, once for each row that is selected by the outer query.
3. A correlated subquery is a sub query that contains a reference to a table that also appears in the outer query.

Example: Consider the following correlated nested query:

```
SELECT *
FROM table1
WHERE col1 ≥ ALL
(SELECT col1 FROM table2 WHERE table2. col2 = table1. col2)
```

1. The subquery contains reference to a column of table1, even though the sub-queries FROM clause does not mention a table table1
2. SQL has to check outside the sub-query and find Table 1 in the outer query
3. Suppose that Table 1 contains a row where col1 = 3 and col2 = 4 and Table 2 contains a row where col1 = 5 and col2 = 4
4. The expression

```
WHERE col1 ≥ All (SELECT col1
FROM table2
```

3 ≥ 5 (false)

```
(WHERE condition TRUE) Table1.
col2 = table2. col2 4 = 4
```

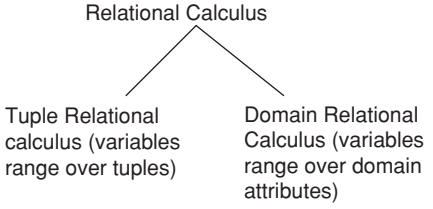
So the expression as a whole is FALSE.

5. It is evaluated from outside to inside

Relational Calculus

Relational calculus can define the information to be retrieved

1. In this, there is no specific series of operations.
2. Relational algebra defines the sequence of operations.
3. Relational calculus is closer to how users would formulate queries, in terms of information requirements, rather than in terms of operations.



4. Relational calculus is based on predicate logic, gives the usual quantifiers to construct complex queries.

Tuple Relational Calculus

Example: Employee

E Id	F Name	L Name	S alary
201	John	James	3000
202	Brat	Frank	2000
203	Mary	Jennifer	3000
204	Adam	Borg	2000
205	Smith	Joyce	1000

Query 1: Display the employees whose salary is above 2000.

$\{E \mid \exists E \in \text{Employee}(E.\text{salary} > 2000)\}$

Output:

E Id	F Name	L Name	Salary
201	John	James	3000
203	Mary	Jennifer	3000

Query 2: Display the employee Ids whose salary is above 1000 and below 3000.

$\{P \mid \exists E \in \text{Employee}((E.\text{salary} > 1000 \wedge E.\text{salary} < 3000) \wedge P.\text{EId} = E.\text{EId})\}$

P is a table, in which EIds are stored, from tuples which satisfies the given condition.

Tuple Relational Calculus

A non-procedural language, where each query is of the form $\{t \mid p(t)\}$. It is a set of all tuples t such that predicate p is true for t, t is a tuple variable, $t[A]$ denotes the value of tuple 't' on attribute A.

$$\{T \mid p(T)\}$$

T is a tuple and $P(T)$ denotes a formula in which tuple variable T appears.

1. $\forall \times (P(X))$
   $\forall$ is called the universal or 'for all' quantifier because every tuple in 'the universe of' tuples must make F true to make the quantified formula true.
   Only true if $p(X)$ is true for every X in the universe.

   Example: $\forall \times (x.\text{color} = \text{'Red'})$
   means everything that exists is red.

   Example: $\forall \times ((x \in \text{Boats}) \Rightarrow (X.\text{color} = \text{'Red'}))$
   '$\Rightarrow$' is a logical implication. $a \Rightarrow b$ means that if a is true, b must be true
   (or)
   $\exists \times \in \text{Boats}\ (X.\text{color} = \text{'Red'})$
   For every 'x' in the boats relation, the color must be red.

2. $\exists \times (P(X))$
   $\exists$ is called the *existential* or '*there exists*' quantifier because any tuple that exists in 'the universe of' tuples may take F true, to make the quantified formula true.

   Example: $\exists \times ((X \in \text{Boats}) \wedge X.\text{color}) = \text{'Red'})$
   There exists a tuple X in the boats relation whose color is red.
   (or)
   $\exists \times \in \text{Boats}\ (X.\text{color} = \text{'Red'})$

Examples:

1. Find all sailors with rating above 8.

Sailors:

Sid	Sname	Rating	Age
28	Yuppy	9	35
35	Rubber	8	55
44	grove	5	35
58	rusty	10	35

Solution: $\{s \mid s \in \text{sailors} \wedge s.\ \text{rating} > 8\}$

Output

$R =$

Sid	Sname	Rating	Age
28	yuppy	9	35
58	rusty	10	35

2. Find names and ages of sailors with rating > 8.

   Solution: $\{R \mid \exists S \in \text{sailors}\ (s.\text{rating} > 8 \wedge R.\text{sname} = s.\text{name} \wedge R.\text{age} = s.\text{age})$

   Output:

sname	age
yuppy	35
rusty	35

Join Operation in Tuple Relational Calculus

Examples:

3. Find sailors rated > 7 who have reserved boat = 103.

Solution: $\{S \mid S \in \text{sailors} \wedge s.\text{rating} > 7 \wedge \exists R\ (R \in \text{reserves} \wedge R.\text{sid} = s.\text{sid} \wedge R.\text{bid} = 103)\}$

4. Find sailors rated > 7 who have reserved a red boat.

Solution: $\{S \mid S \in \text{sailors} \wedge s.\text{rating} > 7 \wedge \exists R\ (R \in \text{reserves} \wedge R.\text{sid} = s.\text{sid} \wedge \exists B(\text{Boats} \wedge B.\text{bid} = R.\text{bid} \wedge B.\text{color} = \text{'Red'}))\}$

Division Operation in Tuple Relational Calculus

Examples

1. Find sailors who have reserved all boats.

Solution: $\{S \mid S \in \text{sailors} \wedge \forall B \in \text{boats}\ (\exists R \in \text{reserves}\ (s.\text{sid} = R.\text{sid} \wedge B.\text{bid} = R.\text{bid}))\}$

Domain Relational Calculus

1. Tuple relational and domain relational are semantically similar.
2. In *TRC*, tuples share an equal status as variables, and field referencing can be used to select tuple parts.

3. In *DRC* formed variables are explicit.
4. *DRC* query has the following form.
   $\{<x_1, x_2, \ldots x_n> / P(<x_1, x_2, \ldots, x_n>\}$
   Result included all tuples $<x_1, x_2, \ldots x_n>$
   That make the formula $p(<x_1, x_2, \ldots x_n>)$ true.
5. Formula given in *DRC* is recursively defined. First start with simple atomic formula and expand the formulas by using the logical connectives.
6. A variable that is not bound is free.
7. The variable $X_1, X_2, \ldots, X_n$ that appear in the left side of '/' must be the only free variable in the formula $p(\ldots)$.

Example: Consider the employee table given in the above Example.
The use of quantifiers $\exists x$ and $\forall x$ in a formula is said to bind x

Query 1: Display the Employees whose salary is above 2000?

$$\{<I, F, L, S> / <I, F, L, S> \in \text{Employee} \wedge S > 2000\}$$

Query 2: Display the EIds of Employees, whose salary is above 1000 and below 3000?

$$\{<I> / \exists F, L, S(<I, F, L, S> \in \text{Employee} \wedge (S > 1000 \wedge S < 3000))\}$$

SQL (Structured Query Language)

When a user wants to get some information from a database file, he/she can issue a query. A query is a user-request to retrieve data (or) information with a certain condition. SQL is a query language that allows user to specify the conditions (instead of algorithms)

Concept of SQL

The user specifies a certain condition. The program will go through all the records in the database file and select those records that satisfy the condition. The result of the query will be stored in the form of a table.

Features of SQL

1. SQL is a language of database. It includes database creation, deletion, fetching rows and modifying rows.
2. SQL is a structured query language for storing, manipulating and retrieving data stored in relational database.
3. It allows users to describe the data.
4. It allows users to create and drop database and tables.
5. It allows users to create view, functions in a database.
6. Allows users to set permissions on tables and views.
7. The standard SQL commands to interact with relational database are CREATE, SELECT, UPDATE, INSERT, DROP and DELETE.
8. The commands can be classified as follows:
   - *Data query language*: SELECT – It retrieves particular rows which satisfies the given condition.
   - *Data definition language*: CREATE, ALTER, DROP.
   - *Data manipulation language*: INSERT, UPDATE, DELETE

Features

1. Strong data protection
2. Robust transactional support
3. High performance
4. High availability
5. Security and flexibility to run anything
6. Easy to manage
7. User friendly

General Structure

SELECT ... FROM ... WHERE
SQL is divided into two languages

1. DML (data manipulation language)
   SELECT: Extracts data from a database table.
   UPDATE: Updates data in a database table.
   DELETE: Deletes data from a database table.
   INSERT INTO: Inserts new data into database table.
2. DDL (data definition language)
   CREATE TABLE - creates a new database table.
   ALTER TABLE - Alters a database table.
   DROP TABLE - deletes a database table.
   CREATE INDEX - Creates an index (search key).
   DROP INDEX - Deletes an index.
   RENAME – Changes the name of the table.

Types of keys:

1. Candidate key
2. Primary key
3. Super key
4. Foreign key
5. Composite primary key

In relational database, 'keys' play a major role. Keys are used to establish and identify relation between relations (or) tables.

Keys are used to ensure that each record within a table can be uniquely identified by combining one or more fields (or) column headers within a table.

Candidate key: A candidate key is a column or set of columns in a table that contains unique values, with these we can uniquely identify any database record without referring to any other columns data.

Each table may have one or more candidate keys, among the available candidate keys, one key is preserved for primary key.

A candidate key is a subset of a super key.

Example: Student

StudentId	First name	Last name	Course Id
*C*S00345	Jim	Black	*C*2
*C*S00254	Carry	Norris	*C*1
*C*S00349	Peter	Murray	*C*1
*C*S00196	John	Mc Cloud	*C*3
*C*S00489	Brat	Holland	*C*4
*C*S00553	Mary	Smith	*C*5

In the above table, we have studentId that uniquely identifies the students in a student table. This would be a candidate key.

In the same table, we have student's first name and last name, which are also candidate keys.

1. If we combine first name and last name then also it becomes a candidate key.
2. Candidate key must (have)
   - Unique values
   - No null values
   - Minimum number of fields to ensure uniqueness.
   - Uniquely identify each record in the table.
3. The candidate keys which are not selected for primary key are known as secondary keys or alternative keys.

Primary key: A primary key is a candidate key that is most suitable (or) appropriate to become main key of the table.

1. It is a special relational database table column ((or) combination of columns)
2. Primary key main features are
   - It must contain a unique value for each row of data.
   - It cannot contain null values.

Example: We can choose primary key as studentId which is mentioned in the table given in above example.

Composite primary key: A key that consists of two or more attributes that uniquely identify an entity is called *composite key* or *composite primary key.*

Example: Customer

Cust-Id	Order-Id	Sale-details
*C*1	*O* – 2	Sold
*C*1	*O* – 3	Sold
*C*2	*O* – 2	Sold
*C*2	*O* – 3	Sold

Composite primary key is {cust-Id, order-Id}

Super key: A super key is a combination of attributes that can be uniquely used to identify a database record. A table can have any number of super keys.

1. Candidate key is a special subset of super keys.

Example: Customer

Customer name	Customer Id	SSN	Address	DOB

Assume that we can guarantee uniqueness only for SSN field, then the following are some of the super keys possible.

1. {Name, SSN, DOB}
2. {ID, Name, SSN}

In a set of attributes, there must be at least one key (could be primary key or candidate key)

Foreign key: A foreign key is a column or group of columns in a relational database table that provides connectivity between data in two tables.

1. The majority of tables in a relational database system adhere to the concept of foreign key.
2. In complex databases, data must be added across multiple tables, thus the link or connectivity has to be maintained among the tables.
3. The concept of Referential Integrity constraint is derived from Foreign key.

Example: Emp

EId	EName	Dept – No

Dept

Dept-No	DName

In the above specified tables, Dept-No is common to both the tables, In Dept table it is called as primary key and in Emp table it is called as *foreign key*.

These two tables are connected with the help of 'Dept-No' field

1. For any column acting as a foreign key, a corresponding value should exist in the link (or) connecting table.
2. While inserting data and removing data from the foreign key column, a small incorrect insertion or deletion destroys the relationship between the two tables.

SQL Commands

SELECT statement

The most commonly used SQL command is SELECT statement. The SQL SELECT statement is used to query or retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table. To create a simple SQL SELECT statement, you must specify the column(s) names and the table name.

Syntax: SELECT column-name (s) from table name

Example: Persons

Lastname	Firstname	Address	City
Hansen	Ola	SpRoad,-20	Hyd
Svendson	Tove	GPRoad,-18	Secbad
Petterson	Kari	RpRoad,-19	Delhi

1. SELECT lastname FROM persons

   Output:

Lastname
Hansen
Svendson
Petterson

2. SELECT lastname, firstname FROM persons

Output:

Lastname	Firstname
Hansen	Ola
Svendson	Tove
Petterson	Kari

DISTINCT statement

Returns distinct values. It eliminates duplicate values.

Syntax: Select DISTINCT column_name (s) from table-name

Example: Orders

Company	Order.No
IBM	3412
DELL	5614
WIPRO	4412
DELL	4413

1. SELECT company FROM orders

Output:

Company
IBM
DELL
WIPRO
DELL

2. SELECT DISTINCT company FROM orders

Company
IBM
DELL
WIPRO

WHERE statement

The WHERE clause is used when you want to retrieve specific information from a table excluding other irrelevant data. By using WHERE clause, we can restrict the data that is retrieved. The condition provided in the WHERE clause filters the rows retrieved from the table and gives only those rows which were expected. WHERE clause can be used along with SELECT, DELETE, UPDATE statements.

The WHERE clause is used to specify a selection condition. All conditions are specified in this clause.

Syntax: SELECT column FROM table WHERE column operator value.

Operates used in where clause:

=
< > (not equal) (or) ! =
>
<
> =
< =

BETWEEN - Between an inclusive range.
LIKE - Search for a pattern

Example: Persons

Lastname	Firstname	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956
Svendson	Tiva	GPRoad, 18	Sec	1977
Smith	Ole	RPRoad, 19	Hyd	1986
Petterson	Kari	SPRoad, 17	Sec	1985

1. SELECT * FROM persons
Output: It displays the entire table

2. SELECT * FROM persons WHERE city = 'Hyd'

Output:

Lastname	Firstname	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956
Smith	Ole	RPRoad, 19	Hyd	1986

LIKE condition

The LIKE operator is used to list all rows in a table whose column values match a specified pattern. It is useful when you want to search rows to match a specific pattern, or when you do not know the entire value. For this purpose, we use a wildcard character '%'.

The LIKE condition is used to specify a search for a pattern in a column.

A '%' sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

Syntax: SELECT column FROM table WHERE column LIKE pattern

1. SELECT * FROM persons WHERE Firstname LIKE 'O%'

Solution: SQL statement will return persons with first names that start with a letter '*O*'

Output:

Lastname	Firstname	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956
Smith	Ole	RPRoad, 19	Hyd	1986

2. SELECT * FROM persons WHERE Firstname LIKE '%a'

Solution: SQL statement will return persons whose first name ends with letter 'a'.

Output:

Last name	First name	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956
Svendson	Tiva	GPRoad, 18	Sec. bad	1977

3. SELECT * FROM persons WHERE firstname LIKE '%la%'

Solution: SQL statement returns persons whose firstname contains 'la'. The word sequence 'la' may come at any place in the word.

Output:

Last name	First Name	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956

String operations

1. '%idge%' matches 'Rockridge', 'Ridgeway', 'Perryridge'.
2. '_____' matches a string of three characters.
3. '_____%' matches a string of at least rhree characters.

INSERT INTO statement

This statement is used to insert new rows into a table. While inserting a row, if you are adding values for all the columns of the table you need not specify the column(s) name in the SQL query. But you need to make sure the order of the values is in the same order as the columns in the table. When adding a row, only the characters or data values should be enclosed with single quotes and ensure the data type of the value and the column matches. One can specify the columns for which you want to insert data

Syntax: INSERT INTO table-name (column1, column2 …) VALUES (value 1, value2 …)

1. INSERT INTO persons VALUES ('Hetland', 'Camilla', 'HPRoad 20', 'Hyd')

Output:

Last name	First Name	Address	City
Hansen	Ola	S.P Road 16	Hyd
Svesdon	Tiva	GP Road 18	Secbad
Smith	Ole	RP Road 19	Hyd
Petterson	Kari	SP Road 17	Secbad
Hetlan	Camilla	HPRoad, 20	Hyd

2. Insert data into specified columns
   INSERT INTO persons (Lastname, Address) VALUES ('Rasmussen', 'street 67')

Output:

Last name	First Name	Address	City
Hansen	Ola	SP Road 16	Hyd
Svesdon	.Tiva	GP Road 18	Secbad
Smith	Ole	RP Road 19	Hyd
Petterson	Kari	SP Road 17	Secbad
Hetlan	Camilla	HP Road 20	Hyd
Rasmussen		Street 67	

UPDATE

The update statement is used to modify the data in a table.

Syntax: UPDATE table_name
SET Column_name = new_value
WHERE column_name = some_value.

1. Add a first name (Nine) to the person whose last name is 'Rasmussen'?

Solution: UPDATE person SET Firstname = 'Nine'
WHERE Lastname = 'Rasmussen'

2. Change the address and add the name of the city as Hyd of a person with last name Rasmussen?

Solution: UPDATE person
SET Address = 'street 12',
city = 'Hyd'
WHERE Lastname = 'Rasmussen'

DELETE statement

The DELETE statement is used to delete rows from a table. The WHERE clause in the SQL delete command is optional, and it identifies the rows in the column that gets deleted. If you do not include the WHERE clause, all the rows in the table will be deleted.

Syntax: DELETE FROM table_name
WHERE column_name = some_value

1. Delete all rows?

Solution: DELETE * FROM table_name

Cartesian product

The Cartesian product of two sets is the set of all ordered pairs of elements such that the first element in each pair belongs to the first set and the second element in each pair belongs to the second set. It is denoted by cross(X).

For example, given two sets:

$S1 = \{1, 2, 3\}$ and $S2 = \{4, 5, 6\}$

The Cartesian product $S1 \times S2$ is the set

$\{(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)\}$

Example:

Female	
Name	Job
Komal	Clerk
Ankita	Sales

Male	
Name	Job
Rohit	Clerk
Raju	Sales

Assume that the tables refer to male and female staff, respectively. Now, in order to obtain all possible inter-staff marriages, the cartesian product can be taken.

Male-Female

Female name	Female job	Male name	Male job
Komal	Clerk	Rohit	Clerk
Komal	Clerk	Raju	Sales
Ankita	Sales	Rohit	Clerk
Ankita	Sales	Raju	Sales

Examples:

1. Find the Cartesian product of borrower and loan?

Solution: SELECT * FROM borrower, loan

2. Find the name, loan-no, and loan amount of all customers having a loan at the Perryridge branch?

Solution: SELECT customer_name, borrower. loan_number, amount FROM borrower, loan WHERE borrower. loan-no = Loan.loan_no AND branch_name = 'perryridge'

3. Find all loan numbers for loans made at the perryridge branch with loan amount greater than 1200?

Solution: SELECT loan-no FROM loan WHERE branch. name = 'perryridge' AND amount > 1200

Comparison operator

Relation algebra includes six comparison operators (=, < >, <, >, < =, > =). These are proposition forming operators on terms. For example, $x < > 0$ asserts that x is not equal to 0. It also includes three logical operators (AND, OR, NOT). These are proposition forming operators on propositions.

Example: $x > 0$ and $x < 8$

Comparison results can be combined using the logical connections AND, OR NOT

1. Find the loan-no of those loans with amounts between 90,000 and 1,00,000?

Solution: SELECT loan-no FORM loan WHERE amount BETWEEN 90,000 AND 1,00,000. SQL allows renewing relations and attributes using 'AS' clause

2. Find the name, loan-no and loan amount of all customers, rename the column name loan-no as loan.id?

Solution: SELECT customer.name, borrower.loan no AS loan.id, amount FROM borrower, loan, WHERE borrower. loan-no = loan.loan-no

Ordering of Tuples

It lists the tuples in alphabetical order.

Example: List in alphabetic order, the names of all customers having a loan in Perryridge branch?

Solution: SELECT customer-name FROM borrower WHERE branch.name = 'perryridge' ORDERBY customer-name

We may specify 'desc' for descending order (or) 'asc' for ascending order. - 'asc' is default.

Example: ORDERBY customer-name desc.

Join (⋈)

SQL Join is used to get data from two (or) more tables, which appear as single table after joining.

1. Join is used for combining columns from two or more tables by using values common to both tables.
2. Self Join: A table can also join to itself is known as self join. Types of JOIN
   1. INNER JOIN
   2. OUTER JOIN
      (i) LEFT OUTER JOIN
      (ii) RIGHT OUTER JOIN
      (iii) FULL OUTER JOIN

1. INNER JOIN (or) EQUI JOIN

It is a simple JOIN in which result is based on matching tuple, depending on the equality condition specified in the query.

Syntax: SELECT Column-names FROM table name1 INNER JOIN table name 2 WHERE table name 1. Column name = table name 2.column – name.

Example: Class

SID	Name
11	Ana
12	Bala
13	Sudha
14	adam

Info

SID	City
11	Bangalore
12	Delhi
13	Hyderabad

SELECT *
FROM Class INNER JOIN Info
WHERE Class.SID = Info.SID

Result:

SID	Name	SID	City
11	Ana	11	Banglore
12	Bala	12	Delhi
13	Sudha	13	Hyderabad

NATURAL JOIN:

NATURAL JOIN is a type of INNER JOIN which is based on column having same name and same data type present in two tables on which join is performed.

Syntax: SELECT *
FROM table-name1 NATURAL JOIN table-name 2

Example: Consider the tables class and Info, and the following Query

SELECT *
FROM class NATURAL JOIN Info

Result:

SID	Name	City
11	Ana	Bangalore
12	Bala	Delhi
13	Sudha	Hyderabad

Both tables being joined have SID column (same name and same data type), the tuples for which value of SID matches in both the tables, appear in the result.

Dangling tuple: When NATURAL JOIN is performed on two tables, there would be some missing tuples in the result of NATURAL JOIN

Those missing tuples are called *Dangling tuples*. In the above example, the number of dangling tuples is 1 that is

14	Adam

OUTER JOIN: Outer Join is based on both matched and unmatched data.

LEFT OUTER JOIN: Left outer Join returns the tuples available in the left side table with the matched data of 2 tables and null for the right tables column.

Example: Consider the table's class and Info

SELECT *
FROM class LEFT OUTER JOIN Info
ON(class.SID = Info. SID)

Result:

SID	Name	City
11	Ana	Banglore
12	Bala	Delhi
13	Sudha	Hyderabad
14	adam	NULL

RIGHT OUTER JOIN: RIGHT OUTER JOIN returns the tuples available in the Right side table with the matched data of 2 tables and NULL for the left table's column.

Example: Class 1

SID	Name
16	Arun
17	Kamal

Info 1

SID	City
16	Chennai
17	Noida

Query:
SELECT *
FROM Class1 RIGHT OUTER JOIN Info1
ON(class1.SID = Info1.SID)

Result:

SID	Name	City
16	Arun	Chennai
18	NULL	Noida

FULL OUTER JOIN: The full outer Join returns the tuples with the matched data of two tables, remaining rows of both left table and Right table are also included.

Example: Consider the tables class 1 and Info1

Query:
SELECT *
FROM class1 FULL OUER JOIN Info1
ON(class1.SID = Info1.SID)

Result:

SID	Name	City
16	Arun	Chennai
17	Kamal	NULL
18	NULL	Noida

ALTER command: ALTER command is used for altering the table structure

1. It is used to add a new column to existing table.
2. To rename existing column.
3. ALTER is used to drop a column.
4. It is used to change data type of any column or modify its size.

Add new column: By using alter command, we can add a new column to the table.

Syntax: ALTER table table-name ADD(column-name data type).

Example: Consider a student table.

SID	S Name	Grade

Add a new column called address

```
ALTER table student ADD (address char);
```

Example: Add multiple columns, parent-name, course-Name, date-of-birth to student table.

```
ALTER table student ADD (parent-name
varchar(60), course-Name varchar(20),
date-of-birth date);
```

Example: Change the data type of column address to varchar?

```
ALTER table student modify(address varchar(30))
```

Example: Rename a column address to Location

```
ALTER table student rename address to Location
```

TRUNCATE command: Truncate command removes all tuples from a table, this command will not destroy the tables structure.

Syntax: Truncate table table-name

DROP Command: DROP query removes a table completely from database. This command will destroy the table structure.

Syntax: Drop table table-name

Rename: This command is used to rename a table.

Syntax: Rename table old-table-name to new-table-name.

Example: Rename table Employee to New-Employee.

DROP a column: Alter command can be combined with DROP command to remove columns from a table.

Syntax: alter table table-name DROP(column-name)

Example: Alter table student DROP (grade)

EXERCISES

Practice Problems I

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. Consider the given table called *Persons*

P-Id	Lastname	Firstname	Address	City
1	Hansen	ola	Timoteivn -10	Sandnes
2	Svendson	Tove	Brazil-50	Sandnes
3	Petterson	Kari	Storgt-20	Stavanger
4	Joseph	ole	Brazil-20	Sandnes

Write a query to select the persons with first name 'Tove' and last name 'Svendson'?

(A)
```
SELECT *
FROM Persons
WHERE first-name='tove'
AND last-name='svendson'
```
(B)
```
SELECT *
FROM Persons
WHERE first-name='tove'
OR last-name='svendson'
```
(C)
```
SELECT first-name
FROM Persons
WHERE first-name='tove'
AND last-name='svendson'
```
(D)
```
SELECT last-name
FROM Persons
WHERE first-name='tove'
AND last-name='svendson'
```

2. Write a query to select only the persons with last name 'Svendson' and the first name equal to 'Tove' or 'ola'?

(A)
```
SELECT *
FROM Persons
WHERE last-name='svendson'
AND first-name='tove'
```
(B)
```
SELECT *
FROM Persons
WHERE  last-name='svendson'
AND (first-name='tove' OR first-name='ola')
```
(C)
```
SELECT *
FROM Persons
WHERE  last-name='svendson'
AND (first-name='tove' AND first-name='ola')
```
(D)
```
SELECT *
FROM Persons
WHERE  last-name='svendson'
OR (first-name='tove' AND first-name='ola')
```

3. Write an SQL statement to add a new row, but only in specified columns, for the persons table add data into columns 'P-Id', 'Last name' and the 'First name' with values (5, Teja, Jakob)?

(A) INSERT INTO Persons VALUES(5,'teja','jakob')
(B) INSERT INTO Persons VALUES(5,teja,jakob)
(C) INSERT INTO Persons (P-Id, last-name, first-name) VALUES(5,'teja','jakob')
(D) INSERT INTO Persons(P-Id, last-name, first-name) VALUES(5,teja,jakob)

4. Write an SQL statement:

(i) To select the persons living in a city that starts with 'S' from the 'Persons' table?

(A)
```
SELECT *
FROM Persons
WHERE  city LIKE 's__'.
```
(B)
```
SELECT *
FROM Persons
WHERE  city LIKE 's%'.
```
(C)
```
SELECT *
FROM Persons
WHERE  city LIKE '%s'.
```
(D)
```
SELECT *
FROM Persons
WHERE city LIKE '_s%'.
```

(ii) To select the persons living in a city that contains the pattern 'tav' from 'Persons' table?

(A)
```
SELECT *
FROM Persons
WHERE city LIKE '_tav_'.
```
(B)
```
SELECT *
FROM Persons
WHERE city LIKE '_tav%'.
```

(C) SELECT *
FROM Persons
WHERE city LIKE '%tav_'.

(D) SELECT *
FROM Persons
WHERE city LIKE '%tav%'.

(iii) To select the persons whose last name starts with '*b*' or '*s*' or '*p*' ?

(A) SELECT *
FROM Persons
WHERE last-name LIKE '*b-s-p*'

(B) SELECT *
FROM Persons
WHERE last-name LIKE '*b%s%p*'

(C) SELECT *
FROM Persons
WHERE last-name LIKE '*b%s%p%*'

(D) SELECT *
FROM Persons
WHERE last-name LIKE '[*bsp*]%'

5. Consider the given table called 'Persons'

P-Id	Last-name	First-name	Address	City
1	Hansen	ola	Timoteivn-10	Sandnes
2	Svendson	Tove	Brazil-50	Sandnes
3	Petterson	Kari	Storgt-20	Stavanger

and the 'Orders' table

O-Id	Order No	P-Id
11	77895	3
12	44678	3
13	22456	1
14	24562	1
15	34764	5

perform NATURAL JOIN operation on both the tables and what is are the O_Id's displayed in the result?

(A) 11, 12, 13 (B) 11, 13, 14
(C) 11, 12, 13, 14 (D) 12, 13, 14

6. Write an SQL to perform FULL JOIN operation on both 'Person' and 'Orders' tables and What is the number of tuples in the Result?

(A) 4 (B) 5
(C) 6 (D) 7

7. Consider the given table 'Result'.

Student Name	Marks
A	55
B	90
C	40
D	80
E	85
F	95
G	82

(i) Find out the students who have scored more than 80 marks, and display them in descending order according to their marks?

(A) SELECT student-name,marks
FROM Result
WHERE marks > 80
ORDERBY marks DESC

(B) SELECT *
FROM Result
WHERE marks > 80
ORDERBY marks DESC

(C) SELECT student-name,marks
FROM Result
WHERE marks > 80
ORDERBY marks

(D) (A) and (B)

(ii) From the above table, find out the top-most three students.

(A) SELECT student-name
FROM Result
ORDERBY marks DESC > 3

(B) SELECT student-name
FROM Result
ORDERBY marks DESC = 3

(C) SELECT student-name
FROM Result
ORDERBY marks DESC limit 3

(D) None of these

8. From the table 'Results', Identify the suitable SQL expression?

(i) Find out the student Who stood 2nd?

(A) SELECT student-name
FROM Result
ORDERBY marks DESC limit 2

(B) SELECT student-name
FROM Result
ORDERBY marks DESC limit 1,1

(C) SELECT student-name
FROM Result
ORDERBY marks DESC limit 1,2

(D) SELECT student-name
FROM Result
ORDERBY marks DESC limit 2,1

(ii) Find out how many students scored > = 80.

(A) SELECT COUNT(*)
FROM Result
WHERE marks > = 80

(B) SELECT COUNT
FROM Result
WHERE marks > = 80

(C) SELECT SUM(*)
FROM Result
WHERE marks > = 80

(D) SELECT SUM
FROM Result
WHERE marks > = 80

9. Consider the given tables:

Customer

Customer name	Customer street	Customer city
Sonam	Mirpurroad	Dhaka
Sonam	Aga KhaRoad	Bogra
Anusha	XYZRoad	Kanchi
Nandy	MirpurRoad	Dhaka

Account

Account number	Customer name	Balance
A-101	Anusha	1000
A-102	Anusha	1500
A-103	Sonam	2000
A-104	Nandy	2500

From the customer table, find out the names of all the customers who live in either Dhaka or Bogra?

(A) SELECT customer-name
FROM customer
WHERE customer-city='dhaka' OR customer-city='bogra'

(B) SELECT customer-name
FROM customer
WHERE customer-city=dhaka OR customer-city='bogra'

(C) SELECT customer-name
FROM customer
WHERE customer-city='dhaka' AND customer-city='bogra'

(D) SELECT customer-name
FROM customer
WHERE customer-city='dhaka' EXIST customer-city='bogra'

10. Consider the given tables

Loan

Loan Number	Branch Name	Amount
L-101	Dhaka	1000
L-103	Khulna	2000

Borrower:

Customer name	Loan number
Sonam	L-101
Nandy	L-103
Anusha	L-103

(i) What are the number of tuples present in the result of cross product of the above two tables?

(A) 4 (B) 5
(C) 6 (D) 7

(ii) Find the loan-numbers from loan table where branch-name is Dhaka?

(A) SELECT loan-number
FROM loan
WHERE branch-name='dhaka'

(B) SELECT loan-number
FROM branch-name='dhaka'

(C) SELECT loan-number
FROM Loan × Borrower

(D) Both (A) and (C)

11. (i) Find all customers who have only accounts but no loans.

(A) SELECT customer-name
FROM depositor LEFT OUTER JOIN Borrower ON
Depositor.customer-name=Borrower.customer-name
WHERE loan-number IS NULL

(B) SELECT customer-name
FROM depositor LEFT OUTER JOIN Borrower ON
Depositor.customer-name = Borrower.customer-name
WHERE loan-number=NULL

(C) SELECT customer-name
FROM depositor RIGHT OUTER JOIN Borrower ON
Depositor.customer-name=Borrower.customer-name
WHERE loan-number IS NULL

(D) SELECT customer-name
FROM depositor RIGHT OUTER JOIN Borrower ON
Depositor.customer-name=Borrower.customer-name
WHERE loan-number=NULL

(ii) Find the names of all customers who have either an account or loan but not both.

Borrower

Customer name	Loan no.
Sonam	L-101
Sonam	L-102
Anusha	L-103

Depositor

Customer name	Account no.
Anusha	A-102
Sonam	A-103
Nandy	A-104

(A) SELECT customer name
FROM depositor FULL OUTER JOIN
Borrower ON
Depositor.customer-name=Borrower.customer-name
WHERE loan-number IS NULL OR Account-number=NULL

(B) SELECT customer-name
FROM depositor FULL OUTER JOIN
Borrower ON
Depositor.customer-name = Borrower.customer-name
WHERE loan-number IS NULL OR Account-number IS NULL

(C) SELECT customer-name
FROM depositor FULL OUTER JOIN
Borrower ON
Depositor.customer-name = Borrower.customer-name
WHERE loan-number = NULL OR Account-number = NULL

(D) SELECT customer-name
FROM depositor FULL OUTER JOIN Borrower ON
Depositor.customer-name = Borrower.customer-name
WHERE loan-number=NULL OR Account-number IS NULL

12. Consider the following 'employee' table

Employee name	Branch name	Branch city	Salary
A	DU	Dhaka	1000
B	DU	Dhaka	2000
C	BUET	Dhaka	3000
D	KUET	Khulna	4000
E	KU	Khulna	5000
F	RU	Rajshahi	6000

(i) Find the distinct number of branches appearing in the employee relation.

(A) SELECT COUNT(branch-name)
FROM Employee

(B) SELECT COUNT(DISTINCT branch-name)
FROM Employee

(C) SELECT DISTINCT COUNT(branch-name)
FROM Employee

(D) SELECT COUNT(*)
FROM Employee

(ii) Find the total salary of all employees at each branch of the bank.

(A) SELECT branch-name, SUM(salary)
FROM Employee
GROUP BY Branch-city

(B) SELECT branch-name, SUM(salary)
FROM Employee
GROUP BY Branch-name

(C) SELECT SUM(salary)
FROM Employee
GROUP BY Branch-name

(D) SELECT branch-name, SUM(salary)
FROM Employee

(iii) Find branch city, branch name Wise total salary, average salary and also number of employees.

(A) SELECT branch-city, branch-name,
SUM (salary), AVG(salary),
COUNT (Employee-name)
FROM Employee
GROUP BY branch-city, branch-name

(B) SELECT branch-city, branch-name,
SUM (salary), AVG (salary),
COUNT (Employee-name)
FROM Employee
GROUP BY branch-city

(C) SELECT branch-city, branch-name,
SUM (salary), AVG (salary), COUNT (Employee-name)
FROM Employee
GROUP BY branch-name

(D) SELECT branch-name, SUM (salary),
AVG (salary), COUNT (Employee-name)
FROM Employee
GROUP BY branch-city, branch-name

Common data for questions 13 to 15: Consider the SHIPMENTS relation and write the SQL statements for the below

SUPPLIERS

Supplier number	Supplier name	Status	City
SN1	Suma	30	Hyderabad
SN2	Hari	20	Chennai
SN3	Anu	10	Hyderabad
SN4	Mahesh	20	Bombay
SN5	Kamal	30	Delhi

PARTS

Part number	Part name	Color	Weight	City
PN1	X	Red	13.0	Chennai
PN2	Y	Green	13.5	Bombay
PN3	X	Yellow	13.2	Hyderabad
PN4	Y	Green	14.1	Calcutta
PN5	Z	Red	14.3	Hyderabad
PN6	Z	Blue	14.2	Bombay

PROJECT

Project number	Project name	City
PJ1	Display	Chennai
PJ2	OCR	Bombay
PJ3	RAID	Chennai
PJ4	SORTER	Hyderabad
PJ5	EDS	Chennai
PJ6	Tape	Bombay
PJ7	Console	Hyderabad

SHIPMENTS

Supplier number	Part number	Project number	Quantity
SN1	PN1	PJ1	300
SN1	PN1	PJ4	400
SN2	PN3	PJ1	350
SN2	PN3	PJ2	450
SN2	PN3	PJ3	640
SN2	PN3	PJ4	320
SN2	PN3	PJ5	330
SN2	PN3	PJ6	520
SN2	PN3	PJ7	480
SN2	PN5	PJ2	460
SN3	PN3	PJ1	440
SN3	PN4	PJ2	410
SN4	PN6	PJ3	310
SN4	PN6	PJ7	320
SN5	PN2	PJ2	340
SN5	PN2	PJ4	350
SN5	PN5	PJ5	360
SN5	PN5	PJ7	370
SN5	PN6	PJ2	380
SN5	PN1	PJ4	420
SN5	PN3	PJ4	440
SN5	PN4	PJ4	450
SN5	PN5	PJ4	400
SN5	PN6	PJ4	410

13. (i) For each part supplied, get the part number and the total shipment quantity?

(A) SELECT shipments.part-number, SUM (shipments.quantity)
FROM Shipments
GROUP BY shipments.part-number

(B) SELECT SUM(shipments.quantity)
FROM Shipments
GROUP BY shipments.part-number

(C) SELECT shipments.part-number, SUM (shipments.quantity)
FROM Shipments
GROUP BY shipments.quantity

(D) SELECT shipments.part-number, SUM (shipments. part-number)
FROM Shipments
GROUP BY shipments.part-number

(ii) Get part numbers for parts supplied by more than two suppliers?

(A) SELECT shipments.part-number
FROM Shipments
GROUP BY shipments.part-number
HAVING COUNT(shipments.supplier-number) > 2

(B) SELECT shipments.part-number
FROM Shipments
GROUP BY shipments.part-number
HAVING COUNT(shipments.supplier-number)>=2

(C) SELECT shipments.part-number
FROM Shipments
GROUP BY shipments.part-number>2

(D) SELECT shipments.part-number, COUNT (shipments.supplier-number)>2
FROM Shipments
GROUP BY shipments.part-number

(iii) Get supplier names for suppliers who supply part PN3?

(A) SELECT DISTINCT suppliers.supplier-name
FROM Supplier
WHERE suppliers.supplier-number IN (SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number='PN3')

(B) SELECT DISTINCT suppliers.supplier-name
FROM Supplier
WHERE suppliers.supplier-number NOT IN(SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number='PN3')

(C) SELECT DISTINCT suppliers.supplier-name
FROM Supplier
WHERE suppliers.supplier-number EXCEPT (SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number='PN3')

(D) SELECT DISTINCT suppliers, supplier-name
FROM Supplier
WHERE suppliers.supplier-number
UNION
SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number='PN3'

14. (i) Get supplier names for suppliers who supply at least one blue part.

(A) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE suppliers.supplier-number
IN (SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number
IN (SELECT Parts.part-number
FROM Parts
WHERE Parts.color='Blue'))

(B) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE suppliers.supplier-number
IN (SELECT Shipments.supplier-number
FROM Shipments

WHERE Shipments.part-number NOT
IN(SELECT Parts.part-number
FROM Parts
WHERE Parts.color='Blue'))

(C) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE suppliers.supplier-number NOT
IN(SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number
IN (SELECT Parts.part-number
FROM Parts
WHERE Parts.color='Blue'))

(D) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE suppliers.supplier-number
IN (SELECT Shipments.supplier-name
FROM Shipments
WHERE Shipments.part-number
IN (SELECT Parts.part-number
FROM Parts
WHERE Parts.color='Blue'))

(ii) Get supplier numbers for suppliers with status less than the current maximum status in the suppliers table:

(A) SELECT Suppliers.supplier-number
FROM suppliers
WHERE Suppliers.status < (SELECT MAX
(Suppliers.status)
FROM Suppliers)

(B) SELECT Suppliers.supplier-number
FROM suppliers
WHERE Suppliers.status<=(SELECT MAX
(Suppliers.status)
FROM Suppliers)

(C) SELECT Suppliers.supplier-number,
MAX (Suppliers.status)
FROM suppliers
WHERE Suppliers.status

(D) SELECT Suppliers.supplier-number
FROM suppliers
WHERE Suppliers.status=MAX(Suppliers.
status)

(iii) Get supplier names for suppliers who supply part PN2?

(A) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(B) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(C) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
OR
Shipments.part-number='PN2')

(D) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
UNION
Shipments.part-number='PN2')

15. (i) Get supplier names for suppliers who do not supply part PN2.

(A) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(B) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(C) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXCEPT(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(D) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Shipments

WHERE Shipments.supplier-number = suppliers.supplier-number
OR
Shipments.part-number='PN2')

(ii) Get supplier names for suppliers who supply all parts.

(A) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Part
WHERE NOT EXIST(SELECT * FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number=Parts.part-number))

(B) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Part
WHERE NOT EXIST(SELECT * FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number=Parts.part-number))

(C) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Part
WHERE EXIST(SELECT * FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number=Parts.part-number))

(D) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Part
WHERE EXIST(SELECT * FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number=Parts.part-number))

(iii) Get part numbers for parts that either weigh more than-16 pounds or are supplied by supplier *SN*3, or both?

(A) SELECT parts.part-number
FROM parts
WHERE Parts.weight>18
UNION
SELECT Shipments.part-number
FROM shipments
WHERE Shipments.supplier-number='SN2'

(B) SELECT parts.part-number
FROM parts
WHERE Parts.weight>18
UNION
SELECT Shipments.supplier-name
FROM shipments
WHERE Shipments.supplier-number='*SN*2'

(C) SELECT parts.part-number
FROM parts
WHERE Parts.weight>18
UNION
SELECT Shipments.part-number,Shipments.supplier-name
FROM shipments
WHERE Shipments.supplier-number='*SN*2'

(D) SELECT parts.part-Number, parts.color
FROM parts
WHERE Parts.weight>18
UNION
SELECT Shipments.part-number
FROM shipments
WHERE Shipments.supplier-number='*SN*2'

Common data for questions 16 and 17: Consider the following relation: Teach

Name	Address	course
Zohar	40B,east city	MD
Nisha	16/2, hyd	BDS
Zohar	40B, East city	MS
Ravi	New York	MBA

16. The teacher with name Zohar teaching the course MS?

(A) σ_{Name} = 'Zohar' teach = MS.
(B) π_{Name} = 'Zohar' teach = MS.
(C) σ_{name} = 'Zohar' and course = 'MS' (teach).
(D) π_{Name} = 'Zohar' and course = 'MS' (teach).

17. Select the names of courses taught by Zohar?

(A) $\pi_{course}(\sigma_{Name\ =\ 'Zohar'}(Teach))$
(B) $\sigma_{course}(\pi_{Name\ =\ 'Zohar'}(Teach))$
(C) $\pi_{course}(\sigma_{Name\ =\ 'MD'}(Teach))$
(D) None

18. Consider the join of a relation *A* with a relation *B*. If *A* has *m* tuples and *B* has *n* tuples. Then the maximum and minimum sizes of the join respectively are.

(A) *mn* and $m + n$ (B) $m + n$ and $(m - n)$
(C) *mn* and *m* (D) *mn* and 0

19. Match the following:

I	Set intersection	1	$R \mid\times\mid S$
II	Natural join	2	$r - (r - s)$
III	Division	3	$\leftarrow$
IV	Assignment	4	$\pi_{R-S}(r) - \pi_{R-S}$ $(\pi_{R-S}(r) \times s)$ $-\pi_{R-S}, s(r)$

(A) I – 2, II – 1, III – 4, IV – 3
(B) I – 3, II – 4, III – 2, IV – 1
(C) I – 1, II – 2, III – 3, IV – 4
(D) I – 2, II – 3, III – 4, IV – 1

20. Which one is correct for division operations for relation r and s

(A) $r \div s$
(B) $\pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - \pi_{R-S}), s(r)$
(C) Temp 1 $\leftarrow \pi_{R-S}(r)$
Temp 2 $\leftarrow \pi_{R-S}(\text{temp1} \times s) - \pi_{R-S}, s(r)$
result = temp 1 – temp 2
(D) All the above

Practice Problems 2

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. The correct order of SQL expression is
(A) Select, group by, where, having
(B) Select, where, group by, having
(C) Select, group by, having, where
(D) Select, having, where, group by

2. Which one is not a query language?
(A) SQL (B) QBE
(C) Data log (D) MySQL

3. Like '*a b* \ % *c d*' escape '\' matches all the strings
(A) Ending with *a b c d*
(B) Beginning with *a b c d*
(C) Beginning with *a b c d*
(D) Beginning with *a b* % *c d*

4. '_ _ _%' matches any string of
(A) At least three characters
(B) At most three characters
(C) Exactly three characters
(D) exactly three characters ending with %

5. Which of the following are set operations?
(i) Union
(ii) Intersection
(iii) Set Difference
(iv) Cartesian Product
(A) (i), (ii), (iii)
(B) (i), (iii), (iv)
(C) (i), (iii), (ii), (iv)
(D) (i), (ii), (iv)

6. What is the purpose of project operation?
(A) It selects certain columns
(B) It selects certain rows
(C) It selects certain strings
(D) It selects certain integers

Common data for questions 7 and 8: Person

Id	Name	Age	Hobby
11	Anu	21	Stamp Collection
22	Kamal	32	Painting
33	Ravi	24	Dancing
44	Ram	22	Singing

7. Select the persons whose hobby is either painting (or) singing.
(A) $\sigma_{\text{Hobby = 'painting' OR Hobby = 'singing'}}$ (person)
(B) $\sigma_{\text{Hobby = 'painting','singing'}}$ (person)
(C) $\sigma_{\text{Hobby = 'painting' OR 'singing'}}$ (person)
(D) All are correct

8. Select the persons whose age is above 21 and below 32:
(A) $\sigma_{\text{age > 21 AND age < 32}}$ (person)
(B) $\sigma_{\text{21 < age < 32}}$ (person)
(C) $\sigma_{\text{age > 21 OR age < 32}}$ (person)
(D) $\sigma_{\text{age < 21 AND age > 32}}$ (person)

Common data for questions 9 and 10: Consider the following relation: Teach

Name	course	Rating	Age
Zohar	MD	7	35
Nisha	BDS	8	27
Zohar	MS	7	34
Ravi	MBA	9	33

9. Select the teachers whose rating is above 7 and whose age is less than 32?
(A) $s_{\text{Rating > 7 AND Age < 32}}$ (Teach)
(B) $s_{\text{Rating} \geq \text{7 AND Age < 32}}$ (Teach)
(C) $s_{\text{Rating > 7 AND < 32}}$ (Teach)
(D) Both (A) and (B)

10. Select the courses with rating above 7?
(A) $\pi_{\text{course}}(\sigma_{\text{rating > 7}}(\text{Teach}))$
(B) $\sigma_{\text{course}}(\pi_{\text{rating > 7}}(\text{Teach}))$
(C) $\pi_{\text{name, course}}(\sigma_{\text{rating > 7}}(\text{Teach}))$
(D) None

Common data for questions 11 and 12: Consider the following schema of a relational database employee (empno, ename, eadd) project (pno, pname) Work–on (empno, pno) Part(partno, partname, qty-on-hand, size) Use (empno, pno, partno, number)

11. Display the names of the employees who are working on a project named 'VB'.
(A) $\sigma_{\text{name}}(\text{employee} \bowtie (\sigma_{\text{pname = 'VB'}}, \text{project}) \bowtie \text{worked on})$
(B) $\sigma_{\text{name}}(\text{employee} \bowtie (\pi_{\text{pname = 'VB'}}, (\text{project}) \bowtie \text{work on})$
(C) $\pi_{\text{name}}(\text{employee} \bowtie (\sigma_{\text{pname = 'VB'}}, (\text{project}) \bowtie \text{work on})$
(D) $\pi_{\text{name}}(\text{employee} \bowtie (\pi_{\text{pname}} = \text{'VB'}(\text{project}) \bowtie \text{work on})$

12. Display the names of the people who are not working for any project.

(A) π_{name} (employee $\bowtie$(π_{name} (employee + work on)

(B) π_{name} (employee – π_{name} (employee $\cap$ work on)

(C) π_{name} (employee – π_{name} (employee $\bowtie$ work on)

(D) σ_{name} (employee – σ_{name} (employee $\bowtie$ work on)

13. Consider the following tables:

A	B	C	D
b	c	e	f
a	b	i	j
b	c	g	h
b	c	a	d
d	i	g	h
d	j	j	k
d	i	e	f

C	D
e	f
g	h

$R \div S$

A	B
b	c
d	i

Which of the following statements is true?

(A) $R \div S = p_{A,B}(R) - p_{A,B}(p_{A,B}(R) \times S + R)$

(B) $R \div S = p_{A,B}(R) - p_{A,B}(p_{A,B}R \times S - R)$

(C) $R \div S = p_{A,B}(R) - p_{A,B}((p_{A,B}(R) \times S) - R)$

(D) $R \div S = p_{A,B}(R) - p_{A,B}(p_{A,B}(R) \times R - S)$

Common data for questions 14 and 15: Consider the following schema of a relational data base
student (sno, name, address)
project (pno, Pname) work-on (sno, pno)
Part (part no, part name, qtyon hand size)
Use (sno, pno, part no, number)

14. List the names of the students who are participating in every project and have used every part.

(A) σ_{name}(student $\bowtie$(((Workon) $\div$ σ_{pro}(project)) $\cap$ (σ_{sno}, $_{part\ no}$(use) $\div$ $\sigma_{part\ no}$ (part)))

(B) π_{name}(student $\bowtie$(((Workon) $\div$ π_{pro}(project)) $\cap$ (π_{sno}, $_{partno}$(use) $\div$ $\sigma_{part\ no}$ (part)))

(C) π_{name}(student $\bowtie$(((Workon) $\div$ π_{partno} (project)) $\cap$ ($\pi_{sno,}$ $_{partno}$(use) $\div$ $\sigma_{part\ no}$ (part)))

(D) π_{name}(student ∞ (((Workon) $\div$ π_{pro}(project)) $\cup$ (π_{ssno}, $_{partno}$(use) $\div$ $\pi_{part\ no}$ (part)))

15. The following query gives π_{name} (employee $\bowtie$(work on $\div$ π_{pro} ($\sigma_{Pname\ =\ 'MS'\ AND\ 'MD'}$(project)))

(A) Names of the students who are working in either projects 'MS' or 'MD'

(B) Names of the students who are working in both the projects 'MS' or 'MD'

(C) Names of the students who are not working in any of the projects 'MS' or 'MD'

(D) None of the above

16. 'All rows corresponding to students whose sno's are between 10 and 20

(i) Select * form student where SNo are between 5 AND 10

(ii) Select * from student where SNO IN(5, 10)

(A) Only (i) (B) Only (ii)

(C) Both (A) and (B) (D) None

17. UPDATE account SET
DA = basic * .2,
GROSS = basic * 1.3, Where basic > 2000;

(A) The above query displays *DA* and gross for all those employees whose basic is ≥ 2000

(B) The above query displays *DA* and Gross for all employees whose basic is less than 2000

(C) The above query displays updated values of *DA* as well as gross for all those employees whose basic is > 2000

(D) All the above

18. Given two union compatible relations $R_1(A, B)$ and $R_2(C, D)$, what is the result of the operation
$R_1\ A = CAB = DR_2$?

(A) $R_1 \cup R_2$ (B) $R_1 \times R_2$

(C) $R_1 - R_2$ (D) $R_1 \cap R_2$

19. Which of the following queries finds the clients of banker Agassi and the city they live in?

(A) $\pi_{client'cname'Ccity}$($\sigma_{client.cname}$ = customer c name ($\sigma_{Banker.}$ $_{name = Aggassi}$ (client × customer)

(B) $\pi_{Client.c\ city}$ ($\sigma_{Banker\ name\ =\ 'Aggasi'}$(client × customer)

(C) $\pi_{client'c\ name'Cucity}$($\sigma_{client.c\ name\ =\ 'Aggasi'}$ ($\sigma_{client'name\ =\ Cutome}$r (client × customer)

(D) $\pi_{.c\ name'Cucity}$($\sigma_{Bankers}$ name = name ($\sigma_{Banker.\ =\ agassi}$ (client × customer)

20. Consider the following schema pertaining to students data
Student (rno, name, add)

Enroll (rno, Cno, Cname) Where the primary keys are shown Underlined. The no. of tuples in the student and Enroll tables are 120 and 8 respectively. What are the maximum and minimum no. of tuples that can be present in (student * Enroll) where '*' denotes natural join.

(A) 8, 8 (B) 120, 8

(C) 960, 8 (D) 960, 120

Previous Years' Questions

1. Consider the relation account (customer, balance) where customer is a primary key and there are no null values. We would like to rank customers according to decreasing balance. The customer with the largest balance gets rank 1, ties are not broke but ranks are skipped; if exactly two customers have the largest balance they each get rank 1 and rank 2 is not assigned.

Query 1: select A.customer, count (B.customer) from account *A*, account *B* where A.balance <= B.balance group by A.customer
Query 2: select A.customer, 1 + count (B.balance) from account *A*, account *B* where A.balance < B.balance group by A.customer Consider these statements about Query1 and Query2.

1. Query1 will produce the same row set as Query2 for some but not all databases.
2. Both Query1 and Query2 are correct implementation of the specification.
3. Query1 is a correct implementation of the specification but Query2 is not.
4. Neither Query1 nor Query2 is a correct implementation of the specification.
5. Assigning rank with a pure relational query takes less time than scanning in decreasing balance order assigning ranks using ODBC.

Which two of the above statements are correct? **[2006]**

(A) 2 and 5 (B) 1 and 3
(C) 1 and 4 (D) 3 and 5

2. Consider the relation enrolled (student, course) in which (student, course) is the primary key, and the relation paid (student, amount) where student is the primary key. Assume no null values and no foreign keys or integrity constraints. Given the following four queries:

Query1: select student from enrolled where student in (select student from paid)

Query2: select student from paid where student in (select student from enrolled)

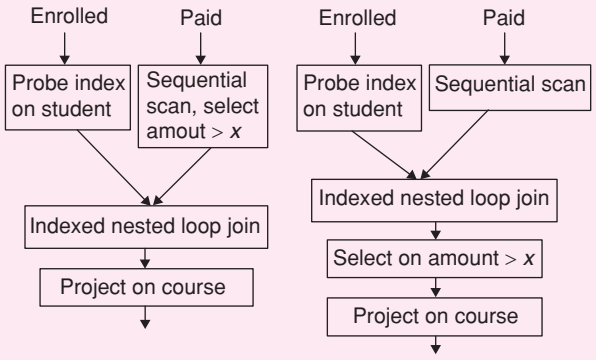
Query3: select E.student from enrolled E, paid P where E.student = P.student

Query4: select student from paid where exists (select * from enrolled where enrolled.student = paid.student)

Which one of the following statement is correct? **[2006]**

(A) All queries return identical row sets for any database
(B) Query2 and Query4 return identical row sets for all databases but there exist databases for which Query1 and Query2 return different row sets
(C) There exist databases for which Query3 returns strictly fewer rows than Query2
(D) There exist databases for which Query4 will encounter an integrity violation at runtime

3. Consider the relation enrolled (student, course), in which (student, course) is the primary key, and the relation paid (student, amount) where student is the primary key. Assume no null values and no foreign keys or integrity constraints. Assume that amounts 6000, 7000, 8000, 9000 and 10000 were each paid by 20% of the students. Consider these query plans (plan 1 on left, plan 2 on right) to 'list all courses taken by students who have paid more than *x*'



A disk seek takes 4 ms, disk data transfer bandwidth is 300 MB/s and checking a tuple to see if amount is greater than x takes 10 μs. Which of the following statements is correct? **[2006]**

(A) Plan 1 and Plan 2 will not output identical row sets for all databases
(B) A course may be listed more than once in the output of Plan 1 for some databases
(C) For $x = 5000$, Plan 1 executes faster than Plan 2 for all databases
(D) For $x = 9000$, Plan 1 executes slower than Plan 2 for all databases

4. Information about a collection of students is given by the relation **studinfo** (studId, name, sex). The relation **enroll** (studId, courseId) gives which student has enrolled for (or taken) what course(s). Assume that every course is taken by at least one male and at least one female student. What does the following relational algebra expression represent?

$$\Pi_{courseId}((\Pi_{studid}(\sigma_{sex = \text{'female'}}(studInfo)) \times \Pi_{courseId}(enroll)) - enroll)$$ **[2007]**

(A) Courses in which all the female students are enrolled
(B) Courses in which a proper subset of female students are enrolled.

(C) Courses in which only male students are enrolled.
(D) None of the above

5. Consider the relation **employee** (name, sex, supervisorName) with *name* as the key. *supervisorName* gives the name of the supervisor of the employee under consideration. What does the following Tuple Relational Calculus query produce?

$$e \cdot \text{name} \mid \text{employee}(e) \wedge$$
$$(\forall x)[\neg \text{employee}(x) \vee x \cdot \text{supervisor Name} \neq e \cdot \text{name} \vee$$
$$x \cdot \text{sex} = \text{"male"}]\} \quad \textbf{[2007]}$$

(A) Names of employees with a male supervisor.
(B) Names of employees with no immediate male subordinates.
(C) Names of employees with no immediate female subordinates.
(D) Names of employees with a female supervisor.

6. Consider the table **employee** (empId, name, department, salary) and the two queries Q_1, Q_2 below. Assuming that department 5 has more than one employee, and we want to find the employees who get higher salary than anyone in the department 5, which one of the statements is **TRUE** for any arbitrary employee table?

Q_1: SELECT e.empId
FROM employee e
WHERE not exists
(Select * From employee s where s.department = '5' and s.salary >=e.salary)

Q_2: SELECT e.empId
FROM employee *e*
WHERE e.salary > Any
(Select distinct salary From employee *s* Where s.department = '5') **[2007]**

(A) Q_1 is the correct query
(B) Q_2 is the correct query
(C) Both Q1 and Q2 produce the same answer.
(D) Neither Q_1 nor Q_2 is the correct query

7. Let R and S be two relations with the following schema
R ($\underline{P}$, $\underline{Q}$, $R1$, $R2$, $R3$)
S ($\underline{P}$, $\underline{Q}$, $S1$, $S2$)
Where $\{P, Q\}$ is the key for both schemas. Which of the following queries are equivalent?

I. $\Pi_P (R \bowtie S)$
II. $\Pi_P (R) \bowtie \Pi_P (S)$
III. $\Pi_P (\Pi_{P,Q} (R) \cap \Pi_{P,Q} (S))$
IV. $\Pi_P (\Pi_{P,Q} (R) - (\Pi_{P,Q} (R) - (\Pi_{P,Q} (S)))$ **[2008]**

(A) Only I and II (B) Only I and III
(C) Only I, II and III (D) Only I, III and IV

8. Let R and S be relational schemes such that $R = \{a,b,c\}$ and $S = \{c\}$. Now consider the following queries on the database:

I. $\pi_{R-S}(r) - \pi_{R-S}(\pi_{R-S}(r) \times s - \pi_{R-S,S}(r))$
II. $\{t \mid t \in \pi_{R-S}(r) \wedge \forall u \in s(\exists v \in r(u = v[s] \wedge t = v[R-S]))\}$
III. $\{t \mid t \in \pi_{R-S}(r) \wedge \forall v \in r(\exists u \in s(u = v[s] \wedge t = v[R-S]))\}$
IV. SELECT R.a, R.b
FROM *R*, *S*
WHERE R.c = S.c

Which of the above queries are equivalent? **[2009]**
(A) I and II (B) I and III
(C) II and IV (D) III and IV

Common data for questions 9 and 10: Consider the following relational schema: Suppliers (sid: integer, sname: string, city: string, street: string) Parts(pid: integer, pname: string, color: string) Catalog (sid: integer, pid: integer, cost: real)

9. Consider the following relational query on the above database:

SELECT S.sname
FROM Suppliers S
WHERE S.sid NOT IN (SELECT C.sid
FROM Catalog C
WHERE C.pid NOT IN (SELECT P.pid FROM Parts P
WHERE P.color <> 'blue'))

Assume that relations corresponding to the above schema are not empty. Which one of the following is the correct interpretation of the above query? **[2009]**
(A) Find the names of all suppliers who have supplied a non-blue part.
(B) Find the names of all suppliers who have not supplied a non-blue part.
(C) Find the names of all suppliers who have supplied only blue parts.
(D) Find the names of all suppliers who have not supplied only blue parts.

10. A relational schema for a train reservation database is given below

Passenger (pid, pname, age)
Reservation (pid, cass, tid)

Table :Passenger
Table :Reservation

Pid	pname	Age
0	'Sachin'	65
1	'Rahul'	66
2	'Sourav'	67

Pid	class	tid
0	'AC'	8200
1	'AC'	8201
2	'SC'	8201

3	'Anil'	69	5	'AC'	8203
			1	'SC'	8204
			3	'AC'	8202

What pids are returned by the following SQL query for the above instance of the tables?

```
SELECT  pid
FROM    Reservation
WHERE   class = 'AC' AND
EXISTS (SELECT *
        FROM Passenger
        WHERE age > 65 AND
        Passenger.pid = Reservation.pid)
```

[2010]

(A) 1, 0 (B) 1, 2
(C) 1, 3 (D) 1, 5

11. Consider a relational table r with sufficient number of records, having attributes $A_1, A_2, \ldots A_n$ and let $1 \le p \le n$. Two queries $Q1$ and $Q2$ are given below.

$Q1$: $\pi_{A_1 \ldots A_n}(\sigma_{A_p=c}(r))$ where c is a constant.

$Q2$: $\pi_{A_1 \ldots A_n}(\sigma_{c_1 \le A_p \le c_2}(r))$ where c_1 and c_2 are constants.

The database can be configured to do ordered indexing on A_P or hashing on A_p. Which of the following statements is TRUE? **[2011]**

(A) Ordered indexing will always outperform hashing for both queries
(B) Hashing will always outperform ordered indexing for both queries.
(C) Hashing will outperform ordered indexing on $Q1$, but not on $Q2$.
(D) Hashing will outperform ordered indexing on $Q2$, but not on $Q1$.

12. Database table by name Loan_Records is given below.

Borrower	Bank manager	Loan amount
Ramesh	Sunderajan	10000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	7000.00

What is the output of the following SQL query?

```
SELECT count ( * )
FROM (Select Borrower, Bank_Manager FROM
Loan Records) AS S
NATURAL JOIN
(SELECT Bank_Manager, Loan_Amount FROM
Loan_Records) AS T;
```

[2011]

(A) 3 (B) 9
(C) 5 (D) 6

13. Consider a database table T containing two columns X and Y each of type *integer*. After the creation of the table, one record ($X = 1$, $Y = 1$) is inserted in the table. Let MX and MY denote the respective maximum values of X and Y among all records in the table at any point in time. Using MX and MY, new records are inserted in the table 128 times with X and Y values being $MX + 1$, $2 * MY + 1$ respectively. It may be noted that each time after the insertion, values of MX and MY change. What will be the output of the following SQL query after the steps mentioned above are carried out?

SELECT Y FROM T WHERE $X = 7$; **[2011]**

(A) 127 (B) 255
(C) 129 (D) 257

14. Which of the following statements are true about an SQL query?

P: An SQL query can contain a HAVING clause even if it does not have a GROUP BY clause
Q: An SQL query can contain a HAVING clause only if it has a GROUP BY clause
R: All attributes used in the GROUP BY clause must appear in the SELECT clause
S: Not all attributes used in the GROUP BY clause need to appear in the SELECT clause **[2012]**

(A) P and R (B) P and S
(C) Q and R (D) Q and S

15. Suppose $R_1(\underline{A}, B)$ and $R_2(\underline{C}, D)$ are two relation schemas. Let r_1 and r_2 be the corresponding relation instances. B is a foreign key that refers to C in R_2. If data in r_1 and r_2 satisfy referential integrity constraints, which of the following is always true? **[2012]**

(A) $\Pi_B(r_1) - \Pi_C(r_2) = \varnothing$
(B) $\Pi_C(r_2) - \Pi_B(r_1) = \varnothing$
(C) $\Pi_B(r_1) = \Pi_C(r_2)$
(D) $\Pi_B(r_1) - \Pi_C(r_2) \neq \varnothing$

Common data for questions 16 and 17: Consider the following relations A, B and C:

(A)

Id	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11

(B)

Id	Name	Age
15	Shreya	24
25	Hari	40
98	Rohit	20
99	Rohit	11

(C)

Id	Phone	Area
10	2200	02
99	2100	01

16. How many tuples does the result of the following SQL query contain?

SELECT A.Id
FROM A
WHERE A. Age > ALL (SELECT B. Age
FROM B
WHERE B. Name = 'Arun') **[2012]**

(A) 4 (B) 3
(C) 0 (D) 1

17. How many tuples does the result of the following relational algebra expression contain? Assume that the schema of $A \cup B$ is the same as that of A.

$(A \cup B) \bowtie_{A.Id > 40 \vee C.Id < 15} C$ **[2012]**

(A) 7 (B) 4
(C) 5 (D) 9

18. Consider the following relational schema. Students (rollno: integer, sname: string) Courses (courseno: integer, cname: string) Registration(rollno:integer,courseno: integer, percent: real)

Which of the following queries are equivalent to this query in English?

'Find the distinct names of all students who score more than 90% in the course numbered 107'

(I) SELECT DISTINCT S.sname FROM Students as *S*, Registration as *R* WHERE R.rollno=S.rollno AND R.courseno=107 AND R.percent>90

(II) $\pi_{sname}(\sigma_{courseno=107 \wedge percent>90}$ Registration $\bowtie$ Students)

(III) $\{T \mid \exists S \in$ Students, $\exists R \in$ Registration (S.rollno=*R*.rollno $\wedge$ R.courseno=107 $\wedge$ R.percent>90$\wedge$T.sname=S.sname)$\}$

(IV) $\{<S_N> \mid \exists S_R \exists R_P (<S_R, S_N> \in$ Students $\wedge <S_R, 107, R_P> \in$ Registration $\wedge R_P > 90)\}$ **[2013]**

(A) I, II, III and IV (B) I, II and III only
(C) I, II and IV only (D) II, III and IV only

19. Given the following statements:

S_1: A foreign key declaration can always be replaced by an equivalent check assertion in SQL.

S_2: Given the table *R* (*a*, *b*, *c*) where a and b together form the primary key, the following is a valid table definition.

CREATE TABLE *S* (
a INTEGER
d INTEGER,
e INTEGER,
PRIMARY KEY (d),
FOREIGN KEY (a) references *R*)

Which one of the following statements is CORRECT? **[2014]**

(A) S_1 is TRUE and S_2 is FALSE
(B) Both *S*1 and S_2 are TRUE
(C) S_1 is FALSE and S_2 is TRUE
(D) Both S_1 and S_2 are FALSE

20. Given the following schema:

Employees (emp–id, first-name, last– name, hire–date, dept–id, salary)

Departments (dept–id, dept–name, manager–id, location–id)

you want to display the last names and hire dates of all latest hires in their respective departments in the location ID 1700. You issue the following query:

```
SQL > SELECT last–name, hire–date
FROM employees
WHERE (dept–id, hire–date) IN
(SELECT dept–id, MAX (hire–date)
FROM employees JOIN departments USING (dept–id)
WHERE location–id = 1700
GROUP BY dept–id);
```

What is the outcome? **[2014]**

(A) It executes but does not give the correct result.
(B) It executes and gives the correct result.
(C) It generates an error because of pair wise comparison.
(D) It generates an error because the GROUP BY clause cannot be used with table joins in a subquery.

21. Given an instance of the STUDENTS relation as shown below:

Student ID	Student Name	Student Email	Student Age	CPI
2345	Shankar	shaker @ math	X	9.4
1287	Swati	swati @ ee	19	9.5
7853	Shankar	shankar @ cse	19	9.4
9876	Swati	swati @ mech	18	9.3
8765	Ganesh	ganesh@ civil	19	8.7

For (StudentName, StudentAge) to be a key for this instance, the value X should NOT be equal to _____. **[2014]**

22. Consider a join (relation algebra) between relations ($r(R)$) and ($s(S)$) using the nested loop method. There are three buffers each of size equal to disk block size, out of which one buffer is reserved for intermediate results. Assuming size $r(R) < size\ s(S)$, the join will have fewer number of disk block accesses if **[2014]**

(A) Relation $r(R)$ is in the outer loop
(B) Relation $s(S)$ is in the outer loop
(C) Join selection factor between $r(R)$ and $s(S)$ is more than 0.5
(D) Join selection factor between $r(R)$ and $s(S)$ is less than 0.5

23. SQL allows duplicate tuples in relations, and correspondingly defines the multiplicity of tuples in the result of joins. Which one of the following queries always gives the same answer as the nested query shown below:

Select * from R where a in (select S. a from S) **[2014]**

(A) Select $R.*$ from R, S where $R.a = S.a$

(B) Select distinct R * from R, S where $R.a = S.a$

(C) Select $R.*$ from R, (select distinct a from S) as $S1$ where $R.a = S1.a$

(D) Select $R.*$ from R, S where $R.a = S.a$ and is unique R

24. What is the optimized version of the relation algebra expression $\pi_{A_1}(\pi_{A_2}(\sigma_{F_1}(\sigma_{F_2(r)})))$, where A_1, A_2 are sets of attributes in r with $A_1 \subset A_2$ and F_1, F_2 are Boolean expressions based on the attributes in r? **[2014]**

(A) $\pi_{A_1}(\sigma_{(F_1 \wedge F_2)^{(r)}})$

(B) $\pi_{A_1}(\sigma_{(F_1 \vee F_2)^{(r)}})$

(C) $\pi_{A_2}(\sigma_{(F_1 \wedge F_2)^{(r)}})$

(D) $\pi_{A_2}(\sigma_{(F_1 \vee F_2)^{(r)}})$

25. Consider the relational schema given below, where **eId** of the relation **dependent** is a foreign key referring to **empId** of the relation **employee**. Assume that every employee has at least one associated dependent in the **dependent** relation.

Consider the following relational algebra query:

employee (empId, empName, empAge)

dependent (depId, eId, depName, depAge)

π_{empId}(employee)- π_{empId} (employee $\bowtie_{(empId = eID) \wedge (empAge \leq depAge)}$ dependent)

The above query evaluates to the set of empIds of employees whose age is greater than that of **[2014]**

(A) some dependent.

(B) all dependents.

(C) some of his/her dependents.

(D) all of his/her dependents.

26. Consider the following relational schema:

employee (empId, empName, empDept)

customer(custId, custName, salesRepid, rating)

salesRepId is a foreign key referring to empId of the employee relation. Assume that each employee makes a sale to at least one customer. What does the following query return?

```
SELECT empName
FROM employee E
WHERE NOT EXISTS
(SELECT custId
FROM customer C
WHERE C.salesRepId = E.empId
AND C.Rating < > 'GOOD');
```

[2014]

(A) Names of all the employees with at least one of their customers having a 'GOOD' rating.

(B) Names of all the employees with at most one of their customers having a 'GOOD' rating.

(C) Names of all the employees with none of their customers having a 'GOOD' rating.

(D) Names of all the employees with all their customers having a 'GOOD' rating.

27. SELECT operation in SQL is equivalent to **[2015]**

(A) The selection operation in relational algebra

(B) The selection operation in relational algebra, except that SELECT in SQL retains duplicates.

(C) The projection operation in relational algebra.

(D) The projection operation in relational algebra, except that SELECT in SQL retains duplicates.

28. Consider the following relations:

Student

Roll No	Student Name
1	Raj
2	Rohit
3	Raj

Performance

Roll No	Course	Marks
1	Math	80
1	English	70
2	Math	75
3	English	80
2	Physics	65
3	Math	80

Consider the following SQL query.

```
SELECT S.Student_Name, sum (P.Marks)
FROM Student S, Performance P
WHERE S.Roll_No = P.Roll_No
GROUP BY S.Student_Name
```

The number of rows that will be returned by the SQL query is _____ **[2015]**

29. Consider two relations $R_1(A, B)$ with the tuples (1, 5), (3, 7) and $R_2(A, C) = (1, 7), (4, 9)$. Assume that $R(A, B, C)$ is the full natural outer join of R_1 and R_2. Consider the following tuples of the form (A, B, C): a = (1, 5, null), b = (1, null, 7), c = (3, null, 9), d = (4, 7, null), e = (1, 5, 7), f = (3, 7, null), g = (4, null, 9). Which one of the following statements is correct? **[2015]**

(A) R contains a, b, e, f, g but not c, d.

(B) R contains all of a, b, c, d, e, f, g.

(C) R contains e, f, g but not a, b.

(D) R contains e but not f, g.

30. Consider the following relation

Cinema (theater, address, capacity)

Which of the following options will be needed at the end of the SQL query

SELECT P_1.address
FROM Cinema P_1
such that it always finds the addresses of theaters with maximum capacity? **[2015]**

(A) WHERE P_1.capacity > = All (select P_2. Capacity from Cinema P_2)
(B) WHERE P_1.capacity >= Any (select P_2. Capacity from Cinema P_2)
(C) WHERE P_1.capacity > All (select max(P_2. capacity) from Cinema P_2)
(D) WHERE P_1.capacity > Any (select max(P_2. capacity) from Cinema P_2)

31. Which of the following is NOT a superkey in a relational schema with attributes *V, W, X, Y, Z* and primary key *VY*? **[2016]**

(A) *V XYZ* (B) *V WXZ*
(C) *V WXY* (D) *V WXYZ*

32. Consider a database that has the relation schema EMP (EmpId, EmpName and DeptName). An instance of the schema EMP and a SQL query on it are given below.

EMP		
EmpId	**EmpName**	**DeptName**
1	XYA	AA
2	XYB	AA
3	XYC	AA
4	XYD	AA
5	XYE	AB
6	XYF	AB
7	XYG	AB
8	XYH	AC
9	XYI	AC
10	XYJ	AC
11	XYK	AD
12	XYL	AD
13	XYM	AE

```
SELECTIVE AVG(EC.Num)
FROM EC
WHERE (DeptName, Num) IN
      (SELECT DeptName, COUNT (EmpId) AS
                              EC(DeptName, Num)
       FROM EMP
       GROUP BY DeptName)
```

The output of executing the SQL query is _____. **[2017]**

33. Consider a database that has the relation schemas EMP(EmpId, EmpName, DeptId), and DEPT(DeptName, DeptId), Note that the DeptId can be permitted to be NULL in the relation EMP. Consider the following queries on the database expressed in tuple relational calculus.

(I) $\{t \mid \exists u \in \text{EMP}(t[\text{EmpName}] = u[\text{EmpName}] \wedge \forall v \in \text{DEPT}(t[\text{DeptId}] \neq v[\text{DeptId}]))\}$
(II) $\{t \mid \exists u \in \text{EMP}(t[\text{EmpName}] = u[\text{EmpName}] \wedge \exists v \in \text{DEPT}(t[\text{DeptId}] \neq v[\text{DeptId}]))\}$
(III) $\{t \mid \exists u \in \text{EMP}(t[\text{EmpName}] = u[\text{EmpName}] \wedge \exists v \in \text{DEPT}(t[\text{DeptId}] = v[\text{DeptId}]))\}$

Which of the above queries are safe? **[2017]**

(A) (I) and (II) only
(B) (I) and (III) only
(C) (II) and (III) only
(D) (I), (II) and (III)

34. Consider a database that has the relation schema CR (studentName, CourseName). An instance of the schema CR is as given below.

CR	
StudentName	**CourseName**
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The following query is made on the database.

$T1 \leftarrow \pi_{CouraseName}(\sigma_{StudentName='SA'}(CR))$

$T2 \leftarrow CR \div T1$

The number of rows in $T2$ is _____. **[2017]**

35. Consider the following database table named *top_scorer*.

top_scorer

player	country	goals
Klose	Germany	16
Ronaldo	Brazil	15
G Miiller	Germany	14
Fontaine	France	13
Pelé	Brazil	12
Klinsmann	Germany	11
Kocsis	Hungary	11
Batistuta	Argentina	10
Cubillas	Peru	10
Lato	Poland	10
Lineker	England	10
T Muller	Germany	10
Rahn	Germany	10

Consider the following SQL query:

```
SELECT ta.player FROM top_scorer AS ta
WHERE ta.goals >ALL (SELECT tb.goals
    FROM top_scorer AS tb
    WHERE tb.country = 'Spain')
AND ta.goals >ANY (SELECT tc.goals
    FROM top_scorer AS tc
    WHERE tc. country = 'Germany')
```

The number of tuples returned by the above SQL query is _____. **[2017]**

36. Consider the following two tables and four queries in SQL.

Book (isbn, bname), Stock (isbn, copies)

Query 1:
```
SELECT B.isbn, S.copies
FROM Book B INNER JOIN Stock S
ON B.isbn = S.isbn;
```

Query 2:
```
SELECT B.isbn, S.copies
FROM Book B LEFT OUTER
JOIN Stock S
ON B.isbn = S.isbn;
```

Query 3:
```
SELECT B.isbn, S.copies
FROM Book B RIGHT OUTER
JOIN Stock S
ON B.isbn = S.isbn;
```

Query 4:
```
SELECT B.isbn, S.copies
FROM Book B FULL OUTER
JOIN Stock S
ON B.isbn = S.isbn;
```

Which one of the queries above is certain to have an output that is a superset of the outputs of the other three queries? **[2018]**

(A) Query 1 (B) Query 2
(C) Query 3 (D) Query 4

37. Consider the relations $r(A, B)$ and $s(B, C)$, where $s \cdot B$ is a primary key and $r \cdot B$ is a foreign key referencing $s \cdot B$. Consider the query

$$Q: r \bowtie (\sigma_{B<5}(S))$$

Let LOJ denote the natural left outer-join operation. Assume that r and s contain no null values.

Which one of the following queries is NOT equivalent to Q? **[2018]**

(A) $\sigma_{B<5}(r \bowtie s)$ (B) $\sigma_{B<5}(r \text{ LOJ } s)$
(C) r LOJ $(\sigma_{B<5}(s))$ (D) $\sigma_{B<5}(r)$ LOJ s

Answer Keys

Exercises

Practice Problems 1

1. A **2.** B **3.** C **4.** (i) B (ii) D (iii) D **5.** C **6.** C **7.** (i) A (ii) C
8. (i) B (ii) A **9.** A **10.** A **11.** (i) A (ii) B **12.** (i) B (ii) B (iii) A
13. (i) A (ii) A (iii) A **14.** (i) A (ii) A (iii) A **15.** (i) A (ii) A (iii) A **16.** C
17. A **18.** D **19.** A **20.** D

Practice Problems 2

1. B **2.** D **3.** D **4.** A **5.** C **6.** A **7.** A **8.** A **9.** A **10.** A
11. C **12.** C **13.** C **14.** C **15.** C **16.** B **17.** C **18.** D **19.** B **20.** A

Previous Years' Questions

1. C **2.** A **3.** C **4.** B **5.** C **6.** B **7.** D **8.** A **9.** A **10.** C
11. C **12.** C **13.** A **14.** C **15.** A **16.** B **17.** A **18.** A **19.** D **20.** B
21. 19 **22.** A **23.** C **24.** A **25.** D **26.** D **27.** D **28.** 2 **29.** C **30.** A
31. B **32.** 2.6 **33.** D **34.** 4 **35.** 7 **36.** D **37.** C