Chapter-3

Relational Languages and Relational Model.

3.1 Introduction to SQL
3.2 Features of SQL
3.3 Queries and sub-Queries
3.4 set operations
3.5 Relations (joined, Desived)
3.6 Queries under DDL & DML commands
3.7 Embedded SQL
3.8 Views
3.9 Relational Algebra
3.10 Database Modification
3.11 QBE and domain relational calcus.

3.1 Introduction to SQL.

↳ SQL is a standard language for storing, manipulating and retrieving data in databases

↳ SQL is an ANSI and ISO standard computer language for creating and manipulating databases

↳ SQL allows the user to create, update, delete, and retrieve data from a database.

↳ SQL works with database programs like DB2, oracle, MS Access, Sybase, MS SQL server etc.

↳ It is the most widely used commercial relational database language.

Scanned by CamScanner

Features of SQL

↳ Allows users to access data in relational database management systems.

↳ Allows users to describe the data.

↳ Allows users to define the data in database and manipulate that data.

↳ Allows to embed within other languages using SQL modules, libraries and pre-compilers

↳ Allows users to create and drop databases and tables.

↳ Allows users to create view, stored procedures, functions in a database.

↳ Allows users to set permissions on tables, procedures and views.

Advantages of SQL:

High Speed

SQL Queries can be used to retrieve large amounts of records from a database quickly and efficiently.

Scanned by CamScanner

Well Defined standard Exist.

SQL databases use long established standard, which is being adopted by ANSI & ISO. Non-SQL databases donot adhere to any clear standard.

NO Coding Required

Using standard SQL it is easier to manage database systems without having to write substantial amount of code.

Emergence of ORDBMS

Previously SQL databases were synonymous with relational database. With the emergence of object oriented DBMS, object storage capabilities are extended to relational databases.

Disadvantages of SQL:

Difficulty in Interfacing

Interfacing an SQL database is more complex than adding a few lines of code.

More Features implemented in Proprietary way

Although SQL databases conform to ANSI & ISO standards, some databases go for proprietary extensions to standard SQL to ensure vendor lock-in.

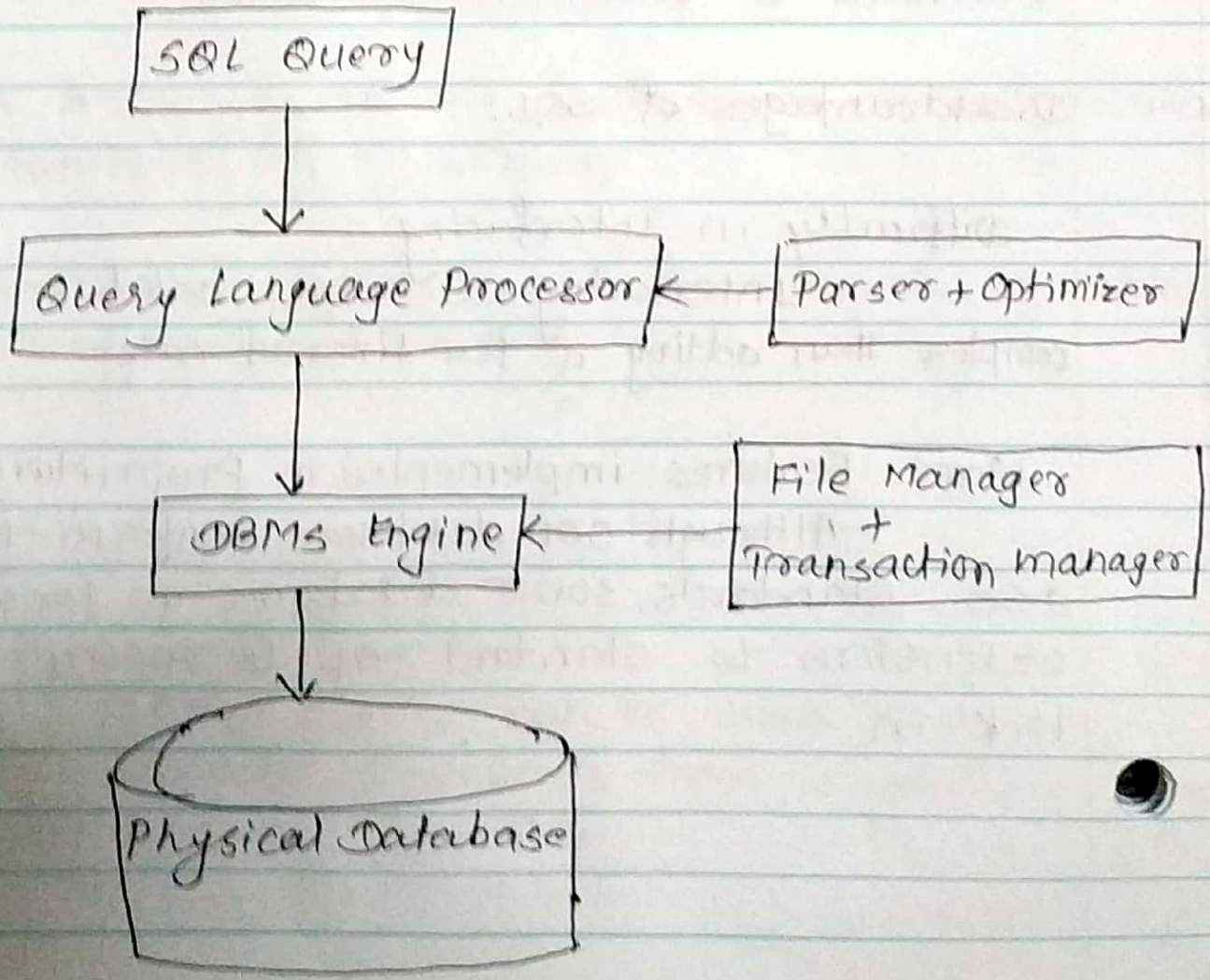
Scanned by CamScanner

SQL process:

When we are executing an SQL commands for any RDBMS, the system determines the best way to carry out our request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic query Engine & SQL query Engine etc.

Following is a simple diagram showing SQL Architecture.



Scanned by CamScanner

Several parts of SQL Language.

Data Definition Language (DDL):

↳ DDL is a syntax for defining data structures, especially database schemas

↳ DDL statements create, modify, and remove database objects such as tables, indexes, and users.

↳ DDL used in database applications are considered a subset of SQL.

↳ DDL has a pre-defined syntax for describing data. For example, to build a new table using SQL syntax, the CREATE command is used, followed by parameters for the table name and the column definitions.

↳ Since DDL is a subset of SQL, it doesnot include all the possible SQL commands For example, commands such as SELECT and INSERT are considered part of the DML. While access command such as CONNECT and EXECUTE are part of the DCL.

↳ Once table is created, it can be modified using the ALTER command if the table is no longer needed, the DROP command can be used to delete the table.

Scanned by CamScanner

↳ SQL uses the following set of commands to define database schema.

CREATE
DROP
ALTER

Data Manipulation Language (DML):

↳ A Data Manipulation language is a syntax for adding (inserting), deleting and modifying (updating) data in a database.

↳ A DML is often a sublanguage of a broader database language such as SQL.

↳ The SQL statements that are in DML class are SELECT, INSERT, UPDATE and DELETE

SELECT
Retrieves certain records from one or more tables

INSERT
Creates a record.

UPDATE
Modifies records

DELETE

Scanned by CamScanner

Deletes records.

Data Control Language (DCL)

↳ A Data Control language (DCL) is a syntax used to control access data stored in a database.

↳ DCL is a component of structured Query Language.

↳ DCL is used to create roles, permissions, and referential integrity as well as it is used to control access to database by securing it.

↳ Only Database Administrators or owners of the database object can provide/remove privileges on a database object.

↳ SQL DCL command
- GRANT
- REVOKE

GRANT syntax: GRANT privilege-name
ON object_Name
TO {user-name|PUBLIC|role-name}
[WITH GRANT OPTION];

Data Control Language (DCL): REVOKE

The REVOKE command removes user access rights

Scanned by CamScanner

or privileges to the database object.

Syntax: REVOKE privilege-name
ON object-name
FROM {user-name|PUBLIC|role-name}

Example: REVOKE SELECT ON employee FROM user1;

This command will REVOKE a SELECT privilege on employee table from user1.

↳ We cannot REVOKE privileges if they were not initially granted by us.

Domain Types in SQL.

Char (n) : Fixed length character string, with user specified length n.

varchar(n): Variable length character strings, with user-specified maximum length n.

int. Integer

smallint. small integer (-32,768 to 32,767)

numeric(p,d): Fixed point number with user specified precision of p digits, with n digits to the right of decimal point.

Scanned by CamScanner

real, double precision: Floating point & double-precision floating point numbers, with machine-dependent precision.

float(n): Floating point number, with uses-specified precision of at least n digits.

Null values are allowed in all the domain types. Declaring an attribute to be not null prohibits null value for that attribute.

date: Dates, containing a (4 digit) year, month and date.

time. Time of day, in hours, minutes and seconds.

Integrity constraints in Create Table.

Constraint restricts the values that the table can store.

There are six constraints

1. Primary key
2. Foreign key
3. Not Null
4. Unique key
5. Default
6. Check

Scanned by CamScanner

Primary key:

↳ It is a combination of unique and a not null constraint; it willnot allow null & duplicate values.

↳ Primary key (A1, ..., An)

↳ Primary key can be declared on two or more columns as a composite primary key

Foreign Key:

↳ It refers the primary key of other table.

↳ This column will take null values & ensure the referential integrity

Foreign key (A1, ..., An) references Table Name.

Not Null :

↳ if a column in a table is specified as Not Null, then it is not allowed to insert null in that column.

↳ It can be implemented with create & alter commands.

Scanned by CamScanner

Unique key:

↳ It doesn't allow duplicate values in a column.

↳ If the unique constraint encompasses two or more columns, no two equal combinations are allowed.

Default:

↳ Used to set default value for the specified attributes.

↳ When a tuple is inserted into the relation, if no value is provided for the attributes, its value is set to default value.

Check:

↳ Used to restrict the values before insesting into a table.

↳ A common use of the check is to ensure that attribute values satisfy specified conditions, in effect creating a powerful type system.

Create Table construct.

An SQL relation is defined using create table command:

Scanned by CamScanner

create table $r(A_1 D_1, A_2 D_2, \ldots A_n D_n)$,
(integrity-constraint$_1$),
...,
(integrity-constraint$_k$))

Where,

r is the name of the relation.

each A_i is an attribute name in the schema of relation r.

D_i is the data type of values in the domain of attribute A_i.

Example:

```
create table instructor(
    ID        char(5)
    name      varchar(20) not null,
    dept_name varchar(20),
    salary    numeric(8,2))
```

↳ insert into instructor values ('10211', 'Ramesh', 'Biology', 66000);

↳ insert into instructor values ('10211', null, 'Biology', 66000);

Scanned by CamScanner

Example: Declare ID as the primary key for instructor.

```
create table instructor (
        ID        char(5),
        name      varchar(20) not null,
        dept-name varchar(20)
        salary    numeric(8,2)
        primary key (ID),
        foreign key (dept-name) references department)
```

primary key declaration on an attribute automatically ensures not null.

Some more examples:

```
create table student (
        ID        varchar(5),
        name      varchar(20) not null,
        dept-name varchar(20),
        tot-cred  numeric(3,0)
        primary key (ID)
        foreign key (dept-name) references department).
```

```
create table takes (
        ID        varchar(5),
        course-id varchar(8),
        sec-id    varchar(8),
        semester  varchar(6),
```

Scanned by CamScanner

```
year      numeric (4,0)
grade     varchar (2)
primary key (ID, course-id, sec-id, semester, year),
foreign key (ID) references student,
foreign key (course-id, sec-id, semester, year)
        references section);
```

sec-id can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester.

```
Create table course (
    course-id  varchar(8) primary key,
    title      varchar(50),
    dept-name  varchar(20),
    credits    numeric(2,0)
    foreign key (dept-name) references department);
```

primary key declaration can be combined with attribute declaration as shown above.

Scanned by CamScanner

Drop and Alter Table constructs.

drop table student
Deletes the table and its contents.

delete from student
Deletes all contents of table, but retains table.

alter table.

1. alter table r add A D
   ↳ where A is the name of the attribute to be added to relation r and D is the Domain of A.
   ↳ All the tuples in the relation are assigned null as the value for the new attribute.

2. alter table r drop A
   ↳ where A is the name of an attribute of relation r.

Basic Query Structure.

↳ The SQL data-manipulation language (DML) provides the ability to query information, and insert, delete and update tuples.

↳ A typical SQL query has the form:

Scanned by CamScanner

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

A_i represents an attribute
R_i represents a relation
P is a predicate.

↳ The result of SQL query is a relation.

The select clause

The select clause list the attributes desired in the result of a query.

Example: find the name of all instructors:
select name
from instructor.

NOTE: SQL names are case insensitive (i.e we may use upper or lower case letters.)
E.g Name ≡ NAME ≡ name.

↳ SQL allows duplicates in relations as well as in query results.

↳ To force the elimination of duplicates insert the keyword distinct after select.

Scanned by CamScanner

Find the names of all departments with instructor, and remove duplicates

```
select distinct dept-name
from instructor.
```

The keyword all specifies that duplicates not be removed

```
select all dept-name
from instructor
```

An asterik in the select clause denotes "all attributes"

```
select *
from instructor.
```

The select clause can contain arithmetic expn involving the operation +, -, * and / and operating on constants or attributes of tuples.

The query :

```
select ID, name, salary/12
from instructor
```

would return a relation that is the same as the instructor relation, except that the value of the attribute salary is divided by 12.

Scanned by CamScanner

The where clause.

The where clause specifies conditions that the result must satisfy corresponds to the selection predicate of the relational algebra.

To find all instructors in comp. sci. dept with salary > 80000

```
select name
from instructor
where dept-name = "Comp. Sci" and salary >
        80 000
```

↳ Comparison results can be combined using the logical connectives and, or, and not.

The from clause.

↳ The from clause lists the relations involved in the query.
corresponds to the cartesian product operation of the relational algebra

↳ Find the cartesian product instructor x teaches

```
select *
from instructor, teaches.
```

Scanned by CamScanner

generates every possible instructor-teaches pair, with all attributes from both relations.

↳ cartesian product not very useful directly but useful combined with where clause condition.

SQL | join (Inner, Left, Right and Full joins)

↳ A SQL join statement is used to combine data or rows from two or more tables based on a common field between them.

↳ Different types of joins are:

i. INNER JOIN
ii. LEFT JOIN
iii. RIGHT JOIN
iv. FULL JOIN

consider the two tables below:

student

Roll-NO	NAME	ADDRESS	PHONE	Age
1.	Rajesh	kathmandu	98.....	18
2.	Pratik	Baglung	98...	19
3.	Priyanka	Biratnagar	98...	20
4.	Deep	Dharan	98...	18
5.	Bhuwan	Pokhara	98...	19
6.	Dhanraj	Nepalgunj	98...	20
7.	Rohit	Surkhet	98...	19

Scanned by CamScanner

Student Course

COURSE-ID	ROLL-NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

1. INNER JOIN

↳ The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies.

↳ This keyword will create the result set by combining all rows from both the tables where the condition satisfies i.e value of common field will be same.

Syntax:

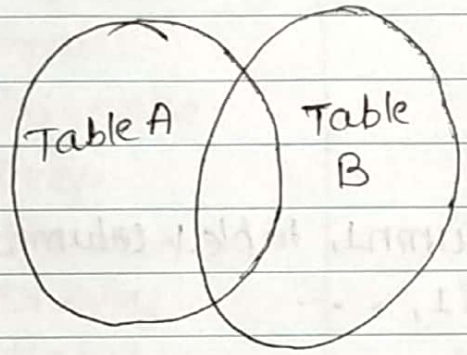
```
SELECT table1.column1, table1.column2, table2.column1,
...
FROM table1
INNER JOIN table2
ON table1.matching-column = table2.matching-column;
```

Scanned by CamScanner

table 1 : First table.
table 2 : second table
matching - column: column common to both the tables.

Note: we can write JOIN instead of INNER JOIN.
JOIN is same as INNER JOIN.

INNER JOIN



This query will show the names and age of students enrolled in different courses.

```
SELECT studentcourse.COURSE-ID, student.NAME,
       student.AGE from student
INNER JOIN studentcourse
ON student.Roll-NO = studentcourse.ROLL-NO;
```

Output

COURSE-ID	Name	Age
1	Rajesh	18
2	Pratik	19
2	Priyanka	20
3	Deep	18
1	Bhuwan	19

Scanned by CamScanner

2. LEFT JOIN:

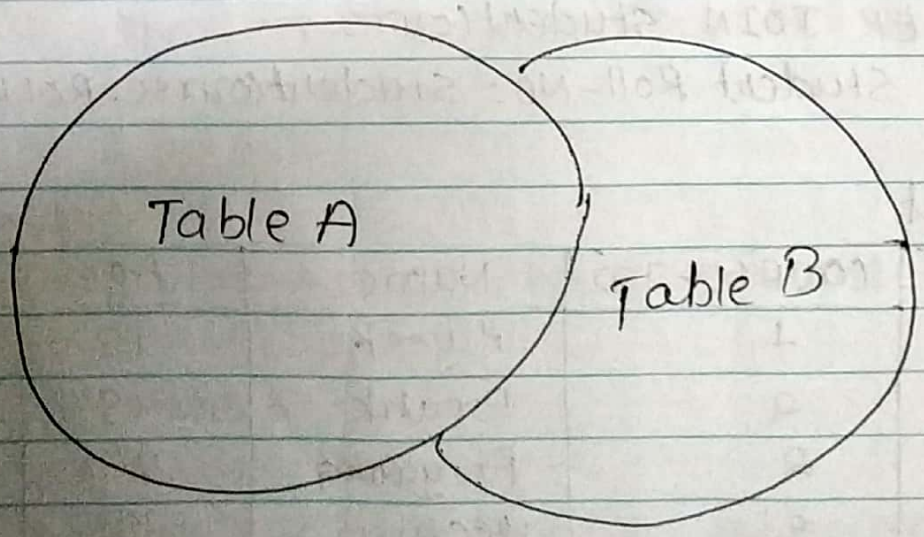
↳ This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of the join.

↳ The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN

Syntax:

```
SELECT table1.column1, table1.column2, table2.
        column1, ...
FROM table1
LEFT JOIN table2
ON table1.matching-column = table2.matching-column
```

Note: we can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Scanned by CamScanner

```
SELECT student.NAME, studentcourse.COURSE_ID
FROM student
LEFT JOIN studentcourse.
ON studentcourse.ROLL_NO = student.ROLL_NO.
```

Output:

NAME	COURSE-ID
Rajesh	1
Pratik	2
Priyanka	2
Deep	3
Bhuwan	1
Dhiraj	NUL
Rohit	NULL

RIGHT JOIN:

↳ This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join.

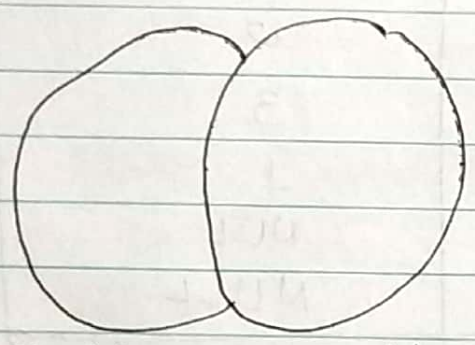
↳ The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN

Syntax:

Scanned by CamScanner

```
SELECT table1.column1, table1.column2,
table2.column1, ...
 FROM table1
 RIGHT JOIN table2
 ON table1.matching-column = table2.matching-column;
```

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN




```
SELECT student.NAME, studentcourse.COURSE_ID
   FROM student
 RIGHT JOIN studentcourse
 ON studentcourse.Roll_NO = student.ROLL_NO;
```

Output:

NAME	COURSE_ID
Rajesh	1
Pratik	2
Priyanka	2
Deep	3
Bhuwan	1
NULL	4
NULL	5

Scanned by CamScanner

FULL JOIN:

↳ FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN.

↳ The result set will contain all the rows from both tables. The rows for which there is no matching, the result set will contain NULL values.

Syntax:

```
SELECT table1.column1, table1.column2, table2.column1,
       . . .
FROM table1
ON table1.matching-column = table2.matching-column;
```

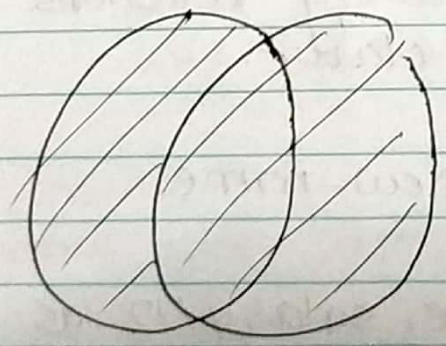


Table A Table B

```
SELECT Student.NAME, StudentCourse.COURSE-ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL-NO = student.ROLL-NO;
```

Scanned by CamScanner

Output:

NAME	COURSE_ID
Rajesh	1
Pratik	2
Priyanka	2
Deep	3
Bhuwan	1
Dhanraj	NULL
Rohit	NULL
NULL	4
NULL	5
NULL	4

The Rename Operation.

The SQL allows renaming relations and attributes using as clause.

old-name as new-name

E.g: select ID, name, salary/12 as monthly-salary
from instructor.

String Operations.

SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are

Scanned by CamScanner

described using two special characters:

percent (%): The % character matches any substring.

underscore (_): The _ character matches any character.

Example:
Find the names of all instructors whose name includes the substring "dar".

```
select name
from instructor
where name like '%dar%'
```

Match the string "100%"

```
like '100\%' escape '\'
```

Patterns are case sensitive.

Pattern matching example.

↳ 'Intro%' matches any string beginning with "Intro"

↳ '%comp%' matches any string containing "comp" as a substring.

Scanned by CamScanner

↳ '- - -' matches any string of exactly characters

↳ '- - - %' matches any string of at least three characters.

↳ SQL supports variety of string operations such as

concatenation (using "||")
converting from upper to lower case (and vice versa)
finding string length, extracting substrings etc.

Ordering the Display of Tuples.

↳ List in alphabetic order the names of all instructors.

```
select name
from instructor
order by name
```

↳ We may specify desc for descending order or asc for ascending order, for each attribute; ascending order is the default.

Example: order by name desc.

Scanned by CamScanner

Where clause Predicates:

SQL includes a between comparison operator.

Example: Find the names of all instructors with salary between $ 90,000 and $100,000

```
select name
from instructor
where salary between 90000 and 100000.
```

Set operations

The SQL set operation is used to combine two or more SQL SELECT statements.

Types of set operation:

1. Union
2. UnionAll
3. Intersect
4. Minus.

Union:

↳ The SQL Union operation is used to combine the result of two or more SQL SELECT queries

↳ In the union operation, all the number of data-type and columns must be same in both the tables

Scanned by CamScanner

on which UNION operation is being applied.

↳ The union operation eliminates the duplicate rows from its resultset.

Syntax:

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2
```

Example:

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

Student_ID	Student_NAME
3	Jackson
4	Stephan
5	David

Scanned by CamScanner

Union SQL query will be

* SELECT * FROM First
UNION
SELECT * FROM Second;

* SELECT ID as SID, NAME as SName FROM First
UNION
SELECT Student-ID as SID, student-NAME as SName FROM second;

The resultset table will look like

ID	NAME
1	Jack
2	Harry
3	Jackson
4	stephan
5	David

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax :

SELECT column-name FROM table1

Scanned by CamScanner

```
UNION ALL
SELECT column_name FROM table 2;
```

Example:

The first table

ID	NAME
1	Jack
2	Harry
3	Jackson

The second table

ID	NAME
3	Jackson
4	stephan
5	David

Union All query will be like:

```
SELECT * FROM First
UNION ALL
SELECT * FROM second;
```

The result set table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson

Scanned by CamScanner

3	Jackson
4	stephan
5	David

3. INTERSECT

↳ It is used to combine two SELECT statements. The intersect operation returns the common rows from both the SELECT statements.

↳ In the intersect operation, the number of data-type and column must be the same.

↳ It has no duplicates and it arranges the data in ascending order by default.

Syntax:

```
SELECT column-name FROM table1
INTERSECT
SELECT column-name FROM table2;
```

Example:

Using the above first and second table

Intersect query will be:

```
SELECT * FROM First
INTERSECT
SELECT * FROM second;
```

Scanned by CamScanner

Output:

ID	NAME
3	Jackson

4. Minus

↳ It combines the result of two select statements. Minus operator is used to display the rows which are present in the first query but absent in second query.

↳ It has no duplicates and data arranged in ascending order by default.

Syntax:

```
SELECT column-name FROM table1
MINUS
SELECT column-name FROM table2;
```

Example:

Minus query will be:

```
SELECT * FROM First
MINUS
SELECT * FROM second;
```

Scanned by CamScanner

output:

ID	NAME
1	Jack
2	Harry

Aggregate functions in SQL

↳ In DBMS an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions:

i. Count ()
ii. Sum ()
iii. Avg ()
iv. Min ()
v. Max ()

ProductID	ProductName	SupplierID	Price	Unit
1	X	1	40	5
2	Y	1	35	7
3	Z	1	55	10
4	A	2	60	3
5	B	2	70	9

Scanned by CamScanner

COUNT () Example

```
SELECT COUNT (ProductID)
FROM Products;
```

Syntax:

```
SELECT (column-name)
FROM table-name
WHERE condition;
```

Note: NULL values arenot counted.

Find the numbes of tuples in the course relation

```
select count (*)
from course;
```

AVG() Example.

AVG() syntax

```
SELECT AVG (column-name)
FROM table-name
WHERE condition.
```

The following SQL statement finds the average price of all products.

Example:

```
SELECT AVG (Price)
FROM Products;
```

Note: NULL values are ignored.

Consider a "ordersDetails" table :

OrdersDetailID	OrdersID	ProductID	Quantity
1	10248	11	12
2	10248	42	10

Scanned by CamScanner

3	10248	72	5
4	10249	14	9
5	10249	51	40

SUM() Syntax:

```
SELECT SUM(column-name)
FROM table-name
WHERE condition;
```

SUM() Example

The following SQL statement finds the sum of the "Quantity" fields in the "OrdersDetails" table:

Example:

```
SELECT SUM(Quantity)
FROM ordersDetails;
```

Note: NULL values are ignored

MIN()

The MIN() function returns the smallest value of selected column

MIN() Syntax

```
SELECT MIN(column-name)
FROM table-name
WHERE condition;
```

Example:

```
SELECT MIN(Price) As SmallestPrice
FROM Products;
```

Scanned by CamScanner

MAX() Syntax

```
SELECT MAX (column-name)
from table-name
WHERE condition;
```

MAX() Example:

```
SELECT MAX(Price) As LargestPrice
FROM Products;
```

Group By statement.

The Group By statement group rows that have the same values in the summary rows like "find the numbers of customers in each city".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns

GROUP BY Syntax.

```
SELECT column-name(s)
FROM table-name
WHERE condition
GROUP BY column-name(s)
ORDER BY column-name(s);
```

Scanned by CamScanner

customers:

CustomerID	CustomersName	ContactName	Address	City
1	Alfreds	Maria Anders	Obere 57	Berlin
2	Ana Trujilo	Ana Trujilo	Avda 222	Mexico
3	Antonio Taquera	Antonia Moreno	Malado 2023	Mexico
4	Around the horn	Thomas Hardy	Hanover 12	London
5	Berglunds snabkop	Christing Berglud	Berguv 8	Luleq

Postal code	country
12209	Germany
05021	Mexico
05023	Mexico
WA1 1DP	UK
S-95822	Sweden.

output:

1	German
2	Mexico
1	Sweden
1	UK

```
SELECT COUNT (customersID), country
FROM customers
GROUP BY country
```

The following statement lists the number of customers in each country, sorted high to low

```
SELECT COUNT (customersID), country
FROM customers
GROUP BY country
ORDER BY COUNT (customerID) DESC;
```

Scanned by CamScanner

GROOP BY With JOIN Example:

Orders

OrderID	CustomesID	EmployeeID	OrdesDate	ShippesID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2
10251	29	3	1996-07-09	2

Shippers

ShipperID	ShippesName
1	Speedy Express
2	United Package
3	Federal Shipping

```
SELECT Shippers.ShippesName, COUNT(orders.OrdesID)
AS NumbesofOrders FROM orders
  LEFT JOIN Shippers
  ON Ordess.shippesID = Shippers.ShippesID
  GROUP BY ShippesName;
```

Output

ShippesName	NumbesofOrders
Fedesal Shipping	1
Speedy Express	1
United Package	2

Scanned by CamScanner

Having Clause.

The HAVING clause was added to SQL because the WHERE keyword couldnot be used with aggregate functions

HAVING Syntax:

```
SELECT column-name(s)
FROM table-name
WHERE condition
GROUP BY column-name(s)
HAVING condition
```

```
SELECT COUNT(customerID), country
FROM customers
GROUP BY country
HAVING COUNT(customerID)>1;
```

	country
2	Mexico

Scanned by CamScanner

NULL values:

↳ It is possible for tuples to have a null value, denoted by null, for some of their attributes.

↳ null signifies an unknown value or that a value doesnot exist.

↳ The result of any arithmetic expressions involving null is null.

Example: 5 + null returns null.

↳ The predicate is null can be used to check for null values

Example:

Find all instructors whose salary is null.

```
select name
from instructor
where salary is null.
```

Null values and Three Valued Logic.

Any comparison with null returns unknown.

Example: 5 < null or null <> null or null = null.

Scanned by CamScanner

↳ Three - valued logic using the truth value unknown.

OR: (unknown or true) = true,
(unknown or false) = unknown,
(unknown or unknown) = unknown.

AND: (true and unknown) = unknown,
(false and unknown) = false
(unknown and unknown) = unknown.

NOT : (not unknown) = unknown

↳ P "is unknown" evalutes to true if predicate P evaluates to unknown.

↳ Result of where clause predicate is treated as false if it evaluates to unknown.

Null values and Aggregates.

Total all salaries

```
select sum(salary)
from instructor
```

↳ Above statement ignores null amounts.
↳ Result is null if there is no non-null amount.

Scanned by CamScanner

↳ all aggregate operations except count(*) ignore tuples with null values on the aggregated attributes

↳ what if collection has only null values?

count returns 0
all other aggregates return null.

SQL sub queries:

↳ In SQL a subquery can be simply defined as a query within another query.

↳ Subquery is a query that is embedded in WHERE clause of another SQL query.

↳ A subquery is a query within another query. The outer query is called as main query and inner query is called as subquery.

↳ subquery must be enclosed in parenthesis.

↳ subqueries are on the right side of the comparison operator.

↳ ORDER BY command cannot be used in subquery. GROUPBY command can be used

Scanned by CamScanner

to perform same function as ORDER BY command.

Sample Table:

Address-table

NAME	Roll-No	Location	Phone-Num
Ram	101	Narayangadh	9888775566
Raj	102	Pokhara	9877665544
Sashi	103	Nepalgunj	9877665533
sumitra	105	kathmandu	9889856868
Ravi	104	Birathagar	9885 33344

Class-table

NAME	ROLL-NO	SECTION
Ravi	104	A
sumitra	105	B
Raj	102	A
Ram	101	B

TO display NAME, LOCATION, PHONE-NUMBER of the students from Address table whose section is A.

select NAME, Location, Phone-Num as PHONE-NUMBER from Address table
WHERE Roll-NO IN

Scanned by CamScanner

(SELECT Roll-NO from class-table WHERE SECTION = 'A');

↳ First subquery executes " SELECT Roll-NO. from class-table WHERE SECTION = 'A' " returns Roll-NO from class-table whose SECTION is 'A'. Then outer query executes it and return the NAME, location, PHONE-NUMBER from the Address-table of the student whose Roll-NO is returned from inner subquery.

Output:

NAME	Roll-NO	Location	PHONE-NUMBER
Ravi	104	Biratnagar	988533344
Raj	102	Pokhara	9877665544

Insert query Example:

Table 1: Student 1

NAME	ROLL-NO	LOCATION	PHONE-NUMBER
Ram	101	Bharatpur	9888775566
Raju	102	Pokhara	9877665544
Ravi	103	Biratnagar	988533344

Scanned by CamScanner

Table 2: student 2

NAME	ROLL-NO	LOCATION	PHONE-NUMBER
Ranjit	111	Bharatpur	9887878787
Prem	112	Dhangadi	9865656565
Arun	113	Pokhara	9878787878

To insert student2 into student1 table:

```
INSERT INTO student1 SELECT * FROM student2;
```

Output:

NAME	ROLL-NO	LOCATION	PHONE-NUMBER
Ram	101	Bharatpur	
Raju	102	Pokhara	
Ravi	103	Bisatnagar	
Ranjit	111	Bharatpur	
Prem	112	Dhangadi	
Arun	113	Pokhara	

↳ To delete from Student2 table whose rollno is same as that in student1 table and having location as ~~[illegible]~~

```
DELETE FROM student2
WHERE ROLL-NO IN ( SELECT ROLL-NO
                   FROM student1
                   WHERE LOCATION = 'Bharatpur');
```

Scanned by CamScanner

To update name of students to Pokhreli in student2 table whose location is same as Raju, Ravi in student1 table.

```
UPDATE student2
SET NAME = 'Pokhreli'
WHERE LOCATION IN( SELECT LOCATION FROM
        student1
    WHERE NAME IN('Raju', 'Ravi'));
```

Output:

1 row updated successfully.

Display student2 table:

NAME	ROLL-NO	LOCATION	PHONE NUMBER
Ranjt	111	Bharatpur	
Prem		Dhangadi	
Pokhreli		Pokhara	

Subqueries in the from clause

↳ SQL allows a subquery expression to be used in the clause.

↳ Find the average instructor's salaries of those department where the average salary is greater than 42,000.

Scanned by CamScanner

```
select dept-name, avg-salary
from( select dept-name, avg(salary) as avg-salary
        from instructor
        group by dept-name)

where avg-salary > 42000;
```

Note: We donot need to use the having clause.

↳ Another way to write above query

```
select dept-name, avg-salary
from(select dept-name, avg(salary)
        from instructor
        group by dept-name)
as dept-avg(dept-name, avg-salary)

where avg-salary > 42000
```

Modification of the database.

↳ Deletion of tuples from a given relation.
↳ Insertion of new tuples into a given relation
↳ Updating values in some tuples in a given relation.

Delete

Delete all instructors

```
delete from instructor
```

Scanned by CamScanner

↳ Delete all instructors from the Finance department.

```
delete from instructor
where dept-name = 'Finance';
```

↳ Delete all tuples in the instructor relation for those instructors associated with a department located in the watson building

```
delete from instructor
where dept-name in (select dept-name
        from department
            where building = 'watson');
```

↳ Delete all instructors whose salary is less than the average salary of instructors

```
delete from instructor
where salary < (select avg(salary) from instructor);
```

Insertion:

Add a new tuple to course

```
insert into course
    values('CS-437', 'Database Systems', 'Comp. Sci', 4);
```

Scanned by CamScanner

or

```
insert into course (course-id, title, dept-name,
                    credits)
    values ('CS-437', 'Database Systems', 'Comp Sci', 4);
```

↳ Add a new tuple to student with tot_creds set to null

```
insert into student
    values ('3003', 'Green', 'Finance', null);
```

↳ Add all instructors to the student relation with tot_creds set to 0

```
insert into student
    select ID, name, dept_name, 0
        from instructor.
```

Updates:

Increase salaries of instructors whose salary is over 1,00,000 by 3% and all others received a 5% raise.

```
update instructor
    set salary = salary * 1.03
    where salary > 100000
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

Scanned by CamScanner

Using case statement:

```
update instructor
set salary = case
        when salary <= 100000 then salary * 1.05
        else salary * 1.03
        end
```

SQL views:

↳ In SQL, a view is a virtual table based on the result-set of an SQL statement.

↳ A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

↳ we can add SQL functions, WHERE, and JOIN statements to a view and present data as if data were coming from one single table.

↳ It can be classified into two categories on the way they created.

Simple view : It is created from a single table.

Complex view : It is created from

Scanned by CamScanner

multiple tables.

Syntax:

create view <viewName> as <Query>

Example:

The following SQL creates a view that shows all customers from Brazil

```
CREATE VIEW [Brazil customers] AS
SELECT customerName, contactName
FROM customers
WHERE country = "Brazil";
```

We can query the view above as follows:

```
SELECT * FROM [Brazil customers];
```

The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products.
WHERE Price > (SELECT AVG(Price) FROM
        Products);
```

Scanned by CamScanner

We can query the view above as follows:

```
SELECT * FROM [Products Above Average
    Price];
```

SQL Updating a view.

A view can be updated with the CREATE OR REPLACE command.

Syntax:

```
CREATE OR REPLACE VIEW view-name AS
SELECT column1, column2, ...
FROM table-name
WHERE condition;
```

The following SQL adds the "city" column to the "Brazil Customers" view.

Example:

```
CREATE OR REPLACE VIEW [Brazil Customers]
AS
SELECT CustomersName, ContactName, City
FROM Customers
WHERE Country = "Brazil";
```

Scanned by CamScanner

SQL Dropping a View

Syntax:

```
DROP VIEW view-name;
```

Example:

```
DROP VIEW [Brazil customers];
```

Sample Tables:

StudentDetails

S-ID	NAME	ADDRESS
1	Hari	kathmandu
2	Ashish	Pokhara
3	Pratik	Beni
4	Dhanraj	Birgunj
5	Ram	Dhasan

StudentMarks

ID	NAME	MARKS	AGE
1	Hari	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Scanned by CamScanner

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS,
    StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME
```

To display data of View MarksView

```
SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Hari	Kathmandu	90
Pratik	Beni	80
Dhanraj	Birgunj	95
Ram	Dharan	85

Advantages of view

Security:
The confidential columns can be suppressed from viewing and manipulated.

Complexity can be avoided.

Views can join and simplify multiple tables into a single virtual table.

Views can limit the degree of exposure

Scanned by CamScanner

to a table or tables to the outer world.

Embedded SQL:

↳ Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language, which is known as host language.

↳ Since the host language cannot parse SQL, the inserted SQL is parsed by an embedded SQL preprocessor.

↳ Embedded SQL is a robust and convenient method of combining the computing power of a programming language with SQL's specialized data management and manipulation capabilities.

↳ Language constructs:

connecting to a database:

EXEC SQL CONNECT

Declaring variables:

EXEC SQL BEGIN(END) DECLARE SECTION

Statements:

EXEC SQL statement.

Scanned by CamScanner

Relational Algebra:

↳ The relational algebra is a theoretical language with opesations that work on one or more relations to define anothes relation without changing the original relations.

↳ Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output.

↳ It uses operators to perform quesies

↳ An operator can be either unary or binary.

↳ They accept relations as their input and yield relations as their output.

↳ Relational algebra is performed recussively on a relation and intermediate results are also considered relations

Basic operators:

Projection (π) Deletes unwanted columns from relation.

Selection (σ) selects a subset rows from relation.

Scanned by CamScanner

cross-product (X) Allows us to combine two relations

set-difference (-) Tuples in relation 1, but not in relation 2

Union (U) Tuples in relation 1 and in relation 2

Desired Operators:

Intersection, join, division, renaming.

Select operation (σ)

This is used to fetch rows (tuples) from table (relation) which satisfies given condition.

Syntax: $\sigma_p(r)$

Where σ represents the select predicate
r is the name of relation (table)
p is prepositional logic.

↳ In prepositional logic, one can use unary and binary operators like =, <, > etc. to specify the conditions.

Scanned by CamScanner

Example:

Students

ID	Name	Age	Gendes
1	Rajesh	17	Male
2	Ramesh	19	Male
3	Ranjit	15	Male
4	Ramila	13	Female

Fetch data for students with age more than 17

$\sigma_{age>17}$ (student)

We can also use, and, or etc. operators.

For example:

$\sigma_{age>17 \text{ and gendes} = \text{'Male'}}$ (student)

Project operation (Π)

Project operation is used to project only a certain set of attributes of a relation.

Syntax:

$$\Pi_{A_1, A_2 .. A_n}(R)$$

Where $A_1, A_2, .. A_n$ are attribute names of

Scanned by CamScanner

relation R.

↳ Schema of result contains exactly the fields in the projection list, with the same names that they had in the input relation.

Example:

$\Pi_{Name, Age}$ (Students)

Union operation (∪)

↳ This operation is used to fetch data from two relations or temporary relation (result of another relation)

↳ For this operation to work, the relations specified should have same number of attributes (columns) and same attribute domain.

↳ Duplicate tuples are automatically eliminated from the result.

Syntax: A ∪ B.

Where A and B are relations.

For example, if we have two tables Regular class and Extra class, both have a column student to save name of student then,

Scanned by CamScanner

Π student (Regular Class) $\cup$ Π student (Extra Class)

Set Difference:

↳ This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation.

Syntax:

A - B

where A and B are relations.

For example, if we want to find name of students who attend the regular class but not extra class then we can use following operation:

$\Pi_{student}$ (Regular Class) - $\Pi_{student}$ (Extra Class)

Intersection:

↳ This operation is used to find data present in both relation. This operation is also applicable on two relations, just like union operation.

Scanned by CamScanner

Syntax

$A \cap B$

Where A and B are relations.

Example:

Find the name of student who attend both Regular Class and Extra Class.

$\Pi_{student}$ (Regular Class) $\cap$ $\Pi_{student}$ (Extra Class)

Cartesian Product (X)

This is used to combine data from two different relations (tables) into one and fetch data from the combined relation.

Syntax:

$A \times B$

Where A and B are relations.

Example:

Information for Regular Class and Extra class which are conducted during morning, then,

$\sigma_{time = 'morning'}$ (Regular Class x Extra Class)

Scanned by CamScanner

Rename Operation (ρ)

↳ This operation is used to rename the output relation for any query operation which returns result like select, Project etc. or to simply rename a relation (table)

Syntax:

$$\rho(\text{RelationNew}, \text{RelationOld})$$

Theta (Θ) Join. ($R \bowtie_F S$)

The Theta join operation defines a relation that contains tuples satisfying the predicate F from the cartesian product of R and S. The predicate F is of the form $R.a_i \; \vartheta \; S.b_i$ where ϑ may be one of the comparison operators.

Example:

Mobile-Table

Mobile	Price
Mi	25000
Samsung	30000
Nokia	40000

Laptop-Table

Laptop	Price
HP	35000
Dell	30000
Sony	70000

Purchase both mobiles and laptop but mobile price should be less than laptop

Scanned by CamScanner

price.

Mobile-Table $\bowtie_{\text{Mobile-Table.Price} < \text{Laptop-Table.Price}}$ Laptop-Table

Equi-Join

When Theta join uses only comparison operator, it is said to be equijoin.

Example:

Mobile-Table $\bowtie_{\text{Mobile-Table.Price} = \text{Laptop-Table.Price}}$ Laptop-Table

Natural Join

↳ Natural join can be performed if there is at least one common attribute that exists between two relation and the attribute must have same name & domain.

Example:

Emp

Name	EmpId	Dname
Ram	1	Finance
Hari	2	Sales
Sita	3	Sales

Dept

Dname	Manager
Finance	Hrishikesh
Sales	Ramesh

Scanned by CamScanner

Emp ⋈ Dept

Name	empid	dname	Manager
Ram	1	Finance	Hrishikesh
Hari	2	Sales	Ramesh
Sita	3	Sales	Ramesh

Left Outer join (R ⟕ S)

↳ All the tuples from the left relation, R are included in the resulting relation

↳ If there are tuples in R without any matching tuple in Right relation S. then the S-attributes of the resulting relation are made NULL.

Courses

A	CName
100	Database
101	Mechanics
102	Electronics

HOD

C	HOD
100	Arun
102	Maya
104	Mira

Courses ⟕ HOD

A	CName	C	HOD
100	Database	100	Arun
101	Mechanics	—	—
102	Electronics	102	Maya

Scanned by CamScanner

Right outer Join (R ⟖ S)

Courses ⟖ HOD

A	CName	C	HOD
100	Database	100	Arun
102	Electronics	102	Maya
—	—	104	Mira

Full outer Join (R ⟗)

A	CName	C	HOD
100	Database	100	Arun
101	Mechanics	—	—
102	Electronics	102	Maya
—	—	104	Mira

Scanned by CamScanner