Date
Page No. ✓

SQL

SQL or SEQUEL

↓ ↓

Structured Query Language — Structured English Query Language

Database

↓

Collection of data stored in a format that can easily be accessed.

DBMS

Relational — NOSQL

↓

MySQL, Oracle, SQL server — own query language

Language - SQL

Basic Format

↓

USE _____

SELECT _____
FROM _____
WHERE _____

Date
Page No. ✓

SELECT

- *
- column name (written in order required)
- AS _____ (to name or rename)
- DISTINCT _____ (to omit the repeat value)
- IF (exp 1st, 2nd);
- IFNULL ();
  - can substitute the null value
  - Eg. IFNULL (id_name, 'Null') AS id
- COALESCE ();
  - can assign new value if null,
  - can assign other column value.
  - Eg. COALESCE (id_name, id, 'null') As id ;
- CASE
  WHEN expression THEN 'value'
  WHEN expression THEN 'value'
  WHEN expression THEN 'value'
  ELSE 'value'
- aggregate functions :→
  - MAX();
  - MIN();
  - AVG();
  - SUM();
  - COUNT (); not count null value

Date
Page No.

- Numeric Function :-
  - ROUND();
    - Eg ROUND(5.3352,2)
      Round the last 2 digit
  - TRUNCATE();
    - Eg. ROUND(5.335,2)
      It will remove last 2 digit
  - CEILING();
    - Eg CEILING(5.2)
      smallest integer equal or greater than 5.2 => 6 (result)
  - FLOOR();
    - Eg FLOOR(5.2)
      greatest integer equal to or smaller than 5.2 => 5 (result)
  - ABS();
    - It will return positive and absolute value.
  - RAND();
    Random floating point value b/w 0 & 1
  - SQRT();
    Square root value
  - LENGTH();
    No. of character in string

SELECT LENGTH('sky') ⇒ 3

Date | Page No. ✓

- **String Function :-**
  - UPPER();
    convert string upper case
  - LOWER();
    convert string lower case
  - LTRIM();
    remove extra space from left
  - RTRIM();
    remove extra space from right

TRIM

  - LEFT();
    Eg. LEFT('Namastey', 4)
    show left 4 character
  - RIGHT();
    Eg. RIGHT('Namastey', 3)
    show right 3 character.
  - SUBSTRING();
    It will return few character randomly
    Eg. SUBSTRING('Namastey', 2, 3)
    starting point ↓ length
  - LOCATE();
    Return the location, not case sensitive
    LOCATE('N', Nameskar)
  - REPLACE();
    REPLACE('Namaskar', 'a', 'd');
    a will replaced by d
  - CONCAT();
    CONCAT('first_name', ' ', 'last_name')
    It will combine first name and last name.

Date | Page No.

- **Data Function :-**
  - NOW() It show current time & date
    Eg. MONTH(NOW())
    DAYNAME(NOW())
    EXTRACT(DAY FROM NOW())
  - CURDATE()
  - CURTIME()

- **Formatting date and time :-**
  - DATE_FORMAT(NOW(), '%Y %d %m')
    - Y = YYYY
    - y = yy
    - M = month name
    - m = in integer
    - D
    - d = date in integer
  - TIME_FORMAT(NOW(), '%H:%i %p')
    - H = 00 to 23
    - h = 00 to 21
    - I = 00 to 21
    - i = min (00 to 59)
    - p = A.M. or P.M.

- **Calculating Date and Time** (can take +ve & -ve value to add & sub day)
  - DATE_ADD(NOW(), INTERVAL 1 DAY)
  - DATE_SUB(NOW(), INTERVAL 2 DAY)
  - DATEDIFF('2019-01-23', '2019-03-20')

- TIME_TO_SEC()

- Subquery in SELECT clause

```
1 SELECT invoice_id, invoice_total,
        (SELECT AVG(invoice_total)
        FROM invoices) AS average,
        invoice_total - (SELECT average)
FROM invoices
```

- IFNULL

if the value is null then it is used to assign the value

```
1. SELECT order_id, IFNULL(shipper_id, 'Not assig
2. FROM orders
```

- COALESCE

It is used to assign new value if other column value.

```
1 SELECT COALESCE(shipper_id, comments, 'Null')
2 FROM orders
```

- IF

IF (expression, 'if true value', if false value)

- CASE WHEN expression THEN val

ELSE

END

Date | Page No. ✓

repetition 0 to more | 1 or more | m times | m or more

{ * | + | ? | {m} | {m,} | {m,n}

~* | ~

case insesitive | Case sensitive

? → 0 or 1 time

{m,} → m or more

{m,n} → not more than n times

\s → space

\. → .

_ → _

Date
Page No. ✓

WHERE

- >, >=, <, <=, =, != or <>
  (String are written in '' or "")
  Date format 'yyyy-mm-dd'
- AND
- OR
- NOT
- IN() or =ANY()
  Ex state =IN('VA','ST','VS')
- BETWEEN
  Eg. points BETWEEN 10 AND 1000
- LIKE
  Eg. last_name LIKE '%b' b in the last
  'b%' b in the starting
  '%b%' b anywhere in middle
  '_b' b at 3rd position
  'b-' b at 2nd last position
  'a--y'
- REGEXP
  'field' field in beginning
  'fields$' in the end
  'field|Mac' field or mac
  '[gun]e' ge, ue, ne
  '[a-h]e' ae, be, ce, de, ee, fe, ge, he
- NULL
  eg WHERE phone IS NULL
- MOD
  To get even id: MOD(id,2) = 0

- EXISTS

```
SELECT *
FROM clients
WHERE clients_id IN (
SELECT DISTINCT client_id
FROM invoices
)
```

0

```
SELECT *
FROM clients C
WHERE EXISTS (
SELECT client_id
FROM invoices
WHERE client_id = C.client_id
)
```

Date | Page No

CREATE FUNCTION

Return only one value.

C

set parameter

deterministic
Modifies SQL data
Read SQL Data

```
CREATE FUNCTION name-of-fun
RETURNS INTGER → data-type it retur
BEGIN attribute
BEGIN .....
    RETURN _____
END
```

- DROP FUNCTION
- DROP FUNCTION IF EXISTS

Date
Page No. ✓

ORDER BY

- Write column name, or no.
- DESC (to order in descending order)
- ORDER BY state, first_name ≠ ORDER BY first_name, state

LIMIT

How many result we want to see

Eg. 1. SELECT *
2. FROM clients
3. LIMIT 3.

Eg. LIMIT 6, 3
(skip 1st 6 record and show next 3)

INNER JOIN / JOIN

Basic structure JOIN table_name ON

```
1. Eg. SELECT *
2.     FROM orders o
3.     Join customers c
4.       ON o.customer_id = c.customer_id
5.     Join orders_status os
6.       ON o.status = os.status_id
```

Date | Page No. ✓

SELF JOIN

```
1. SELECT   E.employee_id,
2.          E.first_name,
3.          E.last_name,
4.          M.first_name AS reports_to
5. FROM employee e
6. JOIN employee m
7.     ON e.reports_to = M.employee_id
```

COMPOUND JOIN

```
1. SELECT *
2. FROM order_items oi
3. JOIN order_item_notes oin
4.     ON oi.order_id = oin.order_id
5.     AND oi.product_id = oin.product_id
```

IMPLICIT JOIN SYNTAX

```
SELECT *
FROM orderss o, customers c
WHERE o.customer_id = c.customer_id
```

CROSS JOIN

IMPLICIT format

```
1. SELECT *
2. FROM orderss, customers
```

Date | | | |
Page No. ✓

EXPLICIT SYNTAX

```
1. SELECT *
2. FROM orders
3. CROSS JOIN products
```

- Every record from 1st table will join to every record in second Table

OUTER JOIN

- **LEFT JOIN**: To see every record in left table weather condition is true or not
- **RIGHT JOIN**: To see every record in right Table

USING

used for joining when column name is same.

```
1. Eg. SELECT *
2. FROM orders
3. Join customers
4.    USING(customer_id)
5.
6. Eg. USING (order_id, product_id)
```

Date | Page No. ✓

NATURAL JOIN

automatical join on the base of common name column.

Eg.
```
SELECT *
FROM orders o
NATURAL JOIN customers c
```

UNIONS

- To add rows.
- column should be equal
- whatever we have in 1st querry will determine the column name

INSERT INTO

- To insert row in column

Eg 1.
```
1. INSERT INTO customers
2. VALUES (DEFAULT, 'John', 'Smith', NULL,
3.         DEFAULT, 'address', 'city',
4.         'state', DEFAULT)
```

Eg 2.
```
1. INSERT INTO customers (first_name,
2.                        last_name, address,
3.                        city, state)
4. VALUES ('John', 'Smith', 'address', 'city',
5.         'state')
```

TRUNCATE Table → Delete Table data
Drop Table ⇒ Delete Table

Date | Page No. ✓

Eg. add multiple rows

```
1 INSERT INTO shippers (name)
2 VALUES ('shipper1'),
3.       ('shipper2'),
4.       ('shipper3'),
5        ('shipper4')
```

Eg. Hierarchial rows

```
1. INSERT INTO orders(customer_id,
2.                    order_date,
3.                    status)
4 VALUES(1, '2019-01-02', 1);
5.
6. INSERT INTO order_items (
7. VALUES
8.     (LAST_INSERT_ID, 1, 2, 3.50),
9.     (LAST_INSERT_ID, 5, 1, 4.92)
```

Copy Table.

```
1. CREATE TABLE orders_archived AS
2. SELECT * FROM orders
```

⇒ SUB-QUERRY IN INSERT INTO

```
1. INSERT INTO orders_archived AS
2. SELECT *
3 FROM orders
4 WHERE order_date < '2019-01-01'
```

Date | Page No. ✓

UPDATE ROW

```
1. UPDATE invoices  Table name
2. SET payment_total = 10,        → column name
3.     payment_date = '2019-03-01'
4. WHERE invoice_id = 1
```

```
1. UPDATE invoices
2. SET payment_total = invoices * 0.5,
3.     payment_date = due_date
4. WHERE invoice_id = 3
```

for updating multiple rows
change preferences → SQL Editor
↓
uncheck the Safe Updates box

```
1. UPDATE invoices
2. SET payment_total = invoices_total * 0.5,
3.     payment_date = due_date
4. WHERE client_id IN
5.       (SELECT client_id
6.        FROM clients
7.        WHERE state IN ('CA', 'NY'))
```

Date
Page No.

DELETE.

```
1. DELETE FROM invoices        ← table name
2. WHERE client_id =
3.         (SELECT client_id
4.         FROM clients
5.         WHERE name = 'My works')
```

GROUP BY

Basic structure

```
SELECT
FROM
WHERE
GROUP BY        can't use AS name
ORDER BY
```

Having - used after group by clause
- ~~named used~~ column name used in having clause must be in SELECT clause

with Rollup - aggregate the all values.
- used after group by clause.

ERROR MESSAGE

```
IF payment_amount <= 0 THEN
    SIGNAL SQLSTATE '22003'
        SET MESSAGE_TEXT = 'Invalid payment amount';
END IF;
```

DECLARE variable_name

Date | Page No.

IFNULL CREATE VEIW

- It is just like table

1 CREATE VEIW name of veiw AS

2 SELECT ..... WITH CHEK OPTION

- DROR VEIW name of veiw to be deleted
- CREATE OR REPLACE VIEW AS
  name of veiw to edit

If veiw code donot have: DISTINCT, Aggregate Fun., GROUP BY / HAVING, UNION.
Then it is updateable ~~table~~ veiw

- WITH CHECK OPTION
  modifying veiw where row can be deleted it will prevent.

CREATE STOREDPROCEDURES

DROP PROCEDURE IF EXISTS get_clients()
(can use any symbol in name)

```
DELIMITER $$
CREATE PROCEDURE get_clients ()
BEGIN
    SELECT * FROM clients;
END $$
DELIMITER ;
```

To call stored Procedure
CALL get_clients()

Better to
- DROP PROCEDURE IF EXISTS get_clients()
- DROP PROCEDURE get_clients()
- IN TO , OUT , DECLARE, IF END IF;

Question / Answers

Date | Page No.

Ques. Return all the products name, unit price, new price (up * 1.1)

→
```
SELECT name,
       unit_price,
       (unit_price * 1.1)
FROM products
```

Ques. Get Order placed this year

→
```
SELECT *
FROM orders
WHERE order_date >= '2021-01-01'
```
YYYY-MM-DD

Ques. From order_items table, get the items for order id 6 and where the total price is greater than 30.

→
```
SELECT *
FROM order_items
WHERE order_id = 6 AND
      unit_price * quantity > 30
```

Ques Return products with quantity in stock equal to 49, 38, 70

→
```
SELECT *
FROM products
WHERE quantity_in_stock IN (49,38,70)
```

Date | Page No.

Ques Return customers born b/w 1/1/1990 and 1/1/2000

→
```
SELECT *
FROM customers
WHERE birth_date BETWEEN
    '1990-01-01' AND '2000-01-01'
```

Ques Get customers whose address contain TRAIL or AVENUE
phone no. ends with 9.

→
```
SELECT *
FROM customers
WHERE address IN ('%TRAIL%',
    '%AVENUE')
address LIKE '%trail%' OR
address LIKE '%avenue%'
phone_no LIKE '%9'
```

Ques Get the customers whose
1. 1st name are ALKA or AMBUR
2. last name ends with EY or ON
3. last name starts with MY or have SE
4. last name contains B followed R or O

→
```
1. SELECT * *
FROM customers
WHERE first_name REGEXP 'alka|amb
```

2. SELECT *
FROM customers
WHERE last_name REGEXP 'EY$|ON$'

3. SELECT *
FROM customers
WHERE last_name REGEXP '^MY|SE'

4. SELECT *
FROM customers
WHERE last_name REGEXP 'B[R,U]'

Ques → Get the orders not shipped

```
SELECT *
FROM orders
WHERE shipped_date IS NULL
```

Ques → Top 3 loyal customers.

```
SELECT customer_name
FROM customers
ORDER BY points DESC
LIMIT 3
```

Ques → Join order_items table to products

```
SELECT oi.order_id, oi.product_id
       p.product_name
FROM order_items oi
JOIN products p
   ON oi.product_id = p.product_id
```

Date | Page No.

Ques Customers with more than 3k points comments them as gold customers in orders table

```
1. UPDATE orders
2. SET comments = 'GOLD customer'
3. WHERE customer_id
4. IN (SELECT customer_id
5.     FROM customers
6.     WHERE points > 3000)
```

Ques show date and payment method and total payment

```
1. SELECT pdate, pm.name AS payment_method,
2.        SUM(amount) AS total_payment
3. FROM payments p
4. JOIN payment_methods pm
5.   ON p.payment_method = pm.payment_meth
6. GROUP BY p.date, payment_method
7. ORDER BY date
8.
```

Ques Get the customers located in virginia who spent more than $10

```
1. SELECT c.first_name, c.state,
2.        SUM(oi.quantity * oi.unitprice) AS total_by
3. FROM customers c
4. JOIN orders o USING (customer_id)
5. JOIN order_items USING(order_id)
6. WHERE state = 'VA'
7. GROUP BY customer_id
```

Date | Page No.

Ques Find products that are more expensive than Lettuce (id=3)

→
```
1. SELECT *
2. FROM products
3. WHERE unit-price >
4.          (SELECT unit_price
5.           FROM products
6.           WHERE product_id = 3)
```

Ques Find employees whose earn more than average

→
```
1.SELECT *
2.FROM employees
3.WHERE salary > (SELECT AVG(salary)
4.                FROM employees)
```

id Ques Find the products that have never been ordered

→
```
1. SELECT *
2. FROM products
3. WHERE product_id NOT IN
4.          (SELECT DISTINCT product_id
5.           FROM order_items)
```

Date | Page No.

Ques Find clients without invoices

```
SELECT *
FROM clients
WHERE client_id NOT IN (
                SELECT DISTINCT client_id
                FROM invoices)
```

OR

```
SELECT *
FROM clients
LEFT JOIN invoices USING (client_id)
WHERE invoice_id IS NULL
```

Ques Find customers who have ordered lettuce (id=3)

Select - customer_id, first_name, last_name

```
SELECT customer_id, first_name, last_name
FROM customers
WHERE customer_id IN (
                SELECT DISTINCT customer_id
                FROM orders
                WHERE orderid IN (
                        SELECT DISTINCT orde
                        FROM order_items
                        WHERE product_id IN (
                                SELECT DISTINCT
                                FROM products
                                WHERE prod
                                        name REGE
                        )))
```

OR

```
1. SELECT  customer_id,
2.         first_name,
3.         last_name
4. FROM customers
5. WHERE customer_id IN (
6.             SELECT o.customer_id
7.             FROM order_items oi
8.             JOIN orders o USING (order_id)
9.             WHERE product_id = 3 )
```

OR

```
1. SELECT customer_id,
2.        first_name,
3.        last_name
4. FROM customers c
5. JOIN orders o USING (customer_id)
6. JOIN order_items oi USING (order_id)
7. WHERE oi.product_id = 3
```

ques SELECT invoices bigger than all invoices of client 3

```
1. SELECT *
2. FROM invoices
3. WHERE invoice_total >
4.         (SELECT MAX (invoice_total)
5.         FROM invoices
6.         WHERE client_id = 3)
```

Date
Page No.

Or

```
1 SELECT *
2 FROM invoices
3 WHERE invoice_total > ALL
4             (SELECT invoice_total
5              FROM invoices
6              WHERE client_id = 3)
```

ALL

Ques. SELECT clientid with at least 2 invoices

→

```
1 SELECT client_id
2 FROM invoices
3 GROUP BY client_id
4 HAVING COUNT(*) >= 2
```

Ques Select employee whose salary is more than avg. salary

→

```
1 SELECT *
2 FROM employees e
3 WHERE salary > (SELECT AVG(salary)
4                 FROM employees
5                 WHERE office_id = e.office_id)
```

Correlated sub query

For each employee this is going to execute sub query

Date | Page No.

Ques Get invoices that are larger than the client's average invoice amount

→
```
1. SELECT *
2. FROM invoices i
3. WHERE invoice_total > (SELECT AVG(invoice_total)
4.                        FROM invoices
5.                        WHERE client_id =
6.                              c.client_id)
```

Ques Select the clients which have invoice

→
```
1 SELECT *
2 FROM clients c
3 WHERE ~~client~~ EXISTS (SELECT ~~client_id~~ *
4                          FROM clients
5                          WHERE client_id = c.client_id)
```

EXISTS

It is preferred to use EXISTS clause when IN operater have huge value.

For each client it going to check EXISTS clause

Ques. Find the product that have never been ordered

→
```
1 SELECT *
2 FROM products p
3 WHERE EXISTS (SELECT *
4                FROM order_items
5                WHERE product_id = p.product_id)
```

Date
Page No.

Ques Select client total sales and average of total sales and their differece

Subquery in SELECT

```
1. SELECT client_id,
2.        name,
3.        (SELECT SUM(invoice_total)
4.        FROM invoices
5.        WHERE client_id = c.client_id) AS sales,
6.        (SELECT AVG(invoice_total)
7.        FROM invoices) AS average,
8.        (SELECT total sales - average) AS differe
9. FROM clients c
```

Ques Label order active and archived acc. to their year of order

IF clause

```
1 SELECT order_id,
        order_date,
        IF(YEAR(order_date) = YEAR(NOW
            'Active', 'Archive') AS order_time
  FROM orders
```

Ques How many times the product have been placed

```
1 SELECT product_id, name, COUNT(*) AS
2        IF(COUNT(*) > 1, 'Manytimes', 'Once') AS
3 FROM products
4 JOIN order_items USING(product_id)
5 GROUP BY product_id, name
```

ders,
equency

Ques Label the order as Active, last_year, Archived and future acc to their order ye

```
1  SELECT order_id,
2         CASE                    YEAR(NOW)
3           WHEN YEAR(order_date) = 1
                THEN 'Active'
4           WHEN YEAR(order_date) =
                YEAR(NOW()) - 1 THEN
                    'last_year'
5           WHEN YEAR(order_date) <
                YEAR(NOW()) - 1
                THEN 'Archived'
6           ELSE 'Future'
7       END AS category
8  FROM orders.
```

Ques Create stored procedure of client table with having state parameter

```
1  DROP PROCEDURE IF EXISTS get_client_state;
2  DELIMITER $$
3  CREATE PROCEDURE get_client_state
4  ( state CHAR(2) )
5  BEGIN
6      SELECT * FROM clients c
7      WHERE c.state = state
8  END$$
9  DELIMITER ;
```

dress, equency

OR

Date | Page No.

```
6 IF STAT state IS NULL THEN
7.    SELECT * FROM clients c;
8.    ELSE
9.        SELECT * FROM clients
10        WHERE c.client.state = state
11.   END IF;
12 END $$
13. DELIMITER ;
```

OR.

```
6.    IFSELECT * FROM clients c
7.        WHERE c.state = IFNULL (state, c.state);
8. END $$
9. DELIMITER ;
```