SQL

Data types

1) Character string
2) Numeric
3) Date & Time
4) Binary string (photos, videos etc)

1) Character String

1) char(n) → stores n characters
2) nchar(n) → stores n Unicode (non-English) characters

→ varchar(n) → store upto n char.

Diff. betn char & varchar →
In varchar if the length of string is less than n it doesn't pad it with extra blank spaces, whereas char pads.

Numeric Data types

1) Int → stores integer
2) float(n) → stores float with prec. n.
3) ~~money~~ →
3) money → stores monetary values
4) decimal(p,s) → stores decimal val. with prec. p & scale s.

Date & Time

1) date → stores dates
2) ~~dt~~
2) datetime → Both date & time
3) time → only time

Binary Data type

1) bit → stores 0 or 1 bit
2) binary(n) → stores n bytes of data
3) varbinary(n) → upto n bytes of data

Scanned with OKEN Scanner

1) DDL (Data def. lang.)
Deals with DB schemas & descriptions, of how data should reside in DB.
E.g. Create, Alter, drop

2) DML (Data Manipulation lang.)
Used to store, modify, retrieve, delete & update data in DB.
E.g. Select, Insert, update, delete

3) DCL (Data control lang.)
Concerned mostly with rights, permissions & other controls of DB system.
E.g. Grant, Revoke

4) TCL (Transaction control lang.)
Deals with transaction within a DB.
E.g. Commit, Rollback, Save point

DDL

CREATE → Used to create new SQL DB.

Syntax:
CREATE DATABASE <DB_name>

For e.g.
Create database Test1

CREATE TABLE: Used to create tables on ~~select DB~~ selected DB.

Syntax:

CREATE TABLE <table_name> (

col. A data_type,
col. B data_type

);

~~For e.g.~~
~~For eg.~~ Use Test1
Create Table testtable
(
emp_id int,
l_name varchar(50) not null,
f_name " " ",

);

Alter table:

→ Add delete modify col. from existing DB.
→ Add drop constraints on existing table

Syntax:
→ ALTER TABLE <table_name> ADD <col_name>
<data type>

Scanned with OKEN Scanner

→ ALTER TABLE <table name> DROP column <col_name>

→ ALTER TABLE <table name> ALTER COLUMN <col_name> <datatype>

For e-g.

→ Alter Table Testtable Add Age int

↳ Adds new col. Age

→ Alter Table Testtable drop column Age

↳ Deletes Age column.

→ Alter Table Testtable alter column empId char(10)

↳ Data type of employee is changed from int to char

DROP TABLE

Syntax: DROP TABLE <table name>

Key Constraints:

→ Constraints enforce rules on the table whenever rows are inserted, updated & deleted from the table.

→ Prevents deletion of table if there are dependencies from other tables.
→ Defines constraints at col. level or table level.
→ Constraints can be applied during creation of table or after creation by alter command.

Const. Constraint	Description
Not Null	Specifies that col must have value
Unique	|| —— col. must have unique val.
Primary key	A col. or set of col. uniquely identifies
Foreign key	~~for~~ It ref. col. of other table
Check	cond. satisfied by all rows in a table

E.g. Creating PK on table itself

(1) Create Table emp ← Most pref. approach for 1 PK
(
emp_id int primary key,
l_name varchar(50) not null,
f_name varchar(50) not null,
);

(2) Create Table emp1
(emp_id int,
l_name varchar(50) not null,
f_name varchar(50) not null,
Constraint emp_pk Primary key (emp_id)
);

Scanned with OKEN Scanner

More than 1 PK

```
Create table emp2 (
  l_n varchar(50) not null,
  f_n ~~var~~ " ------- " ,
  salary money,
  Constraint emp_pk1 primary key (l_n, f_n)
  );
```

Adding a key after table creation

Syntax: ~~Add~~

~~ADD CONSTRAINT emp_pk PRIMARY KEY~~

```
ALTER TABLE <table_name>
ADD CONSTRAINT <name_PK> PRIMARY KEY ~~(col_name)~~
                                     (col_name);
```

For e.g.

~~Alter table~~
~~Add consta~~

```
Alter table emp
Add constraint emp_pk Primary key (emp_id);
```

~~As for add for PK we can use the same syntax for FK.~~

Foreign key

E.g. Creation within table

parent table

```
Create table product
( prod_id int primary key,
prod_name varchar (50) not null,
category  "  (25)
);
```

child table

```
Create table Orders
(
order_id int primary key,
prod_id int not null,
quantity int,
~~Constraint fk_product_id~~
Foreign key (prod_id)
references product (prod_id)
or
); prod_id int Foreign key
references product (prod_id);
```

PK of parent table is foreign key of child table

E.g. ~~Af~~ AFTER CREATION OF TABLE

Page No.:
Date. / /

~~Syntax~~:

Orders 1

ALTER TABLE ~~<table name>~~
ADD ~~CONSTRAINT fk_product_id~~
FOREIGN KEY (prod_id)
REFERENCES product (prod_id);

Note: When we have a PK constraint on parent table which is a FK ~~on~~ in child table & we try to delete that col from parent table, it would give us an error, to do so we need to write 'ON Delete cascade' ~~below creat~~ below foreign key.

Note: Diff. betn Unique key & Primary key
→ Unique key ~~may~~ allows null values for only one column, but PK doesn't allow.

DROP CONSTR.

If we have added a constr.
→ Alter table emp add constraint pk_empid Primary key (emp_id);

To drop a constr.
→ Alter table emp drop constraint pk_empid

CRETI

CREATING VIEWS :

A view is a named, derived, virtual table. A view takes the o/p of the query & treats it as a table.

Syntax :

CRETE VIEW <view_name> AS
SELECT * FROM <table_name> ;

To query the view

SELECT * FROM <view_name>

Create view with specific col.

CREATE VIEW <view-name> AS
SELECT <col-name>
,<col=name>
:
,<col_name>
FROM <table-name>
WHERE condition;

Use: When we need to write query quite often, so instead of writing that query everytime, we create a

view & query the view.

DML

Insert

→ It is used to insert data in DB.
→ Inserting values for the specific col. in the table. (Always include columns which are not null)

Syntax:

Insert into <Table-name>(Fieldname1, ... Fieldname2 ---) Values(val1, val2 ---)

E.g.
Insert into dept(deptno, dept name) values (20, 'HR');

If we want to enter values in all col. then no need to specify col. values, but order must be in sync with col. names.

E.g.

Insert into dept values(20, 'HR');

'Insert-As-Select' Statement

This st. allows us to insert into a table using o/p from another table. Record from one table would be inserted in another table.

Syntax:

```
INSERT INTO <table-name> (column-names)
SELECT column-names
FROM <table-name>
WHERE condition;
```

Eg:

① Insert into dept1 (Deptno, dept_name, loc) (Select * from dept);

~~② Insert~~

② Insert into dept1 (Deptno, dept_name, loc) (Select * from dept where deptno > 20);

③ Insert into dept1 (Deptno, dept_name) (Select deptno, dept_name from dept);

Order is very imp.

Update Statement:

→ Modifies existing data in table

→ Update single or multiple col in a single statement

Syntax:

UPDATE <table-name> Set <field-name> = <value> Where <condition> ;

Note:

→ Without where clause all rows will get updated

→ Updating multiple col. [while updating multiple col., col must be seperated by comma)

E.g.

~~upd~~
update dept set loc = 'Pink City', ~~dept~~ dept_name = 'HR' where deptno = 20;

Delete Statement

→ Helps delete rows from DB.
→ Exec of delete without where cond - would delete all records.

Syntax:

~~Delete from <table-name~~

~~Delete from <table-name> [where <condition>]~~

Delete from <table-name> [where <condition>];

E.g.

a) ⇒ To delete all rows.

⇒ delete from dept;

b) ⇒ Conditional deletion.

⇒ delete from dept where loc='Pink City';

Select Statement

→ Helps to retrieve records from DB.
→ where cond^n is optional.

Syntax:

Select <field 1, field 2 ...> from <table-name> where <condition>;

Page No.:
Date.

E-g. select * from dept ; (fetches all col. from dept.)

Eg. select top(3) * FROM dept;

fetches top 3 rows from dept.

Select Statement: Alias name for a field

~~Eg.~~ Using alias name for a field:

Syntax:

select <col1> as <alias name1>, <col2> as <alias name2> from <tab-name>;

Eg.
select dept name, loc as location from dept;

dept name, location

Distinct Values

Used to retrieve unique values for a col. Multiple rows can have same values for a col., distinct keyword helps to retrieve unique rows for a col.

Scanned with OKEN Scanner

Syntax:

select distinct <col_name> from <table>;

E.g.
select distinct loc from dept;

o/p:

LOC
Chennai
Banglore

ORDER By

→ Used along with where clause to display specified column in ascending or descending order
→ Default is ascending order.

Syntax:

~~SELE~~
SELECT [distinct] <columns(s)>
FROM <table>
WHERE <condition>
ORDER BY <column(s) [asc [desc]>

Scanned with OKEN Scanner

Page No.:
Date. / /

E.g.

SELECT f_name, d_no, hire_date
FROM employee
~~ORDER BY deptno;~~

SELECT firstname, deptno, hire_date
FROM employee
ORDER BY deptno ASC, hire_date desc;

ascending order

descending order

O/P:

First_name	deptno	hire_date
Smith	10	21-Jul-10
Nancy	10	12-Feb-99
John	30	20-Jan-10
Arun	30	05-Jun-08

LOGICAL OPERATORS

AND, OR, NOT

E.g. ~~ste~~
select * from employee where f_name = 'Adwait' OR f_name = 'Purao';

~~Replace Replace OR by~~

E-g.

```
select dept_name, loc from dept where
    loc not in('Chennai','Bangalore');
```

O/p:

dept_name	LOC
HR	Hyderabad

COMPARISON OPERATORS

(=, !=, <7, >=, <=, LIKE, BETWEEN, IN)

Between operator:

Used to search for values that are within a set of values.

E-g.

```
select f_name,deptno ~~where~~ from employee
where salary between 200 and 350;
```

IN Operator:

Fetches value from a set of literals. Used to test whether or not a value is 'in' the list of values provided by keyword IN. Can be used with any data type.

E.g.

```
select fname, deptno from employee
where job_id in ('J1', 'J2');
```

LIKE : Condition

Used to perform wild card searches of valid search string values.

→ search cond^n can contain either nos or characters

→ % denotes zero or many characters

→ _ denotes one character

E.g.:

① select dept_name, loc from dept where loc like 'c%';

o/p: dept_name	LOC
accounts	Chennai
marketing	— " —

② select dept_name, loc from dept where loc like 'chen__';

o/p: dept_name	LOC
accounts	Chennai
Marketing	— " —

Scanned with OKEN Scanner

CASE EXPRESSION:

Used as a type of if-else statement

Note: You cannot use simple case to test for null because it always uses equals operator (=). That That is because the cond" null = null is not true ~~consequently~~ consequently, a when NULL clause never applies. If the <common operand> is null, the else clause applies.

Syntax:

```
CASE col_name
WHEN condition1 THEN result1
-11- -11- 2 -11- -11- 2
ELSE result
END
FROM table_name
```

E.g.

①
```
DECLARE @intInput INT
SET @intInput = 2
SELECT CASE (@intInput)
WHEN 1 then 'One'
-11- 2 -11- 'Two'
-11- 3 -11- 'Three'
ELSE 'Your message'
END
```

Scanned with OKEN Scanner

Page No.:
Date. / /

② SELECT CASE(LOC)
WHEN 'Chennai' then 'TamilNadu'
—11— 'Bangalore'—11— 'KR'
ELSE 'NO IDEA'
END
FROM DEPT;

Joins:

→ Used to fetch data from 2 or more tables based cond".
→ It is used to combine rows from one (Self-Join) or more tables, based on related col. set" them.

→ Self Join
→ Cross —11— (Cartesian product)
→ Inner —11—
→ Left —11—
→ Right —11—
→ Full Outer join

Self-Join:

A table can be joined to itself on a self join.

Syntax:

```
SELECT <col_list>
FROM <table_name1> t1
JOIN <table_name1> t2
ON t1.col_name = t2.col_name;
```

E.g.

```
SELECT
t2.f_name as Employee
, t1.f_name as Manager
FROM
employee t1 JOIN employee t2
ON t1.emp_id = t2.mgr_id;
```

O/P:

Employee	Manager
Scott	Allen
Martin	—11—
Arthur	Jake
Quincy	Allen
Smith	Quincy
Miller	Allen

Inner Join

Fetches records that have matching values.

Page No.:
Date: / /

Syntax:

```
SELECT <column_list>
FROM <table_name> as t1
INNER JOIN <table_name2> as t2
ON t1.col_name = t2.col_name;
```

emp	Employee ID	First Name	Last Name	Address
1	101	Jack	Joe	Mumbai
2	102	Smith	Jane	Delhi
3	103	Jerg.	Samantha	Kolkata
4	104	Reyn.	Allen	Bang.
5	105	Ander.	Paige	NULL
6	106	John	Derek	Chennai

orders	Order ID	Employee ID	Order Date
1	1	104	2016-4-18
2	2	102	-11-
3	3	105	-11--19
4	4	101	-11--20
5	5	NULL	-11-05-20

E-g:

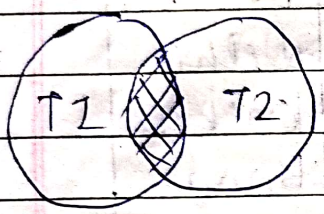
```
SELECT emp.EmployeeID, orders.OrderID,
       orders.OrderDate
FROM emp
INNER JOIN orders
ON emp.EmployeeID = orders.EmployeeID
ORDER BY emp.EmployeeID;
```

Scanned with OKEN Scanner

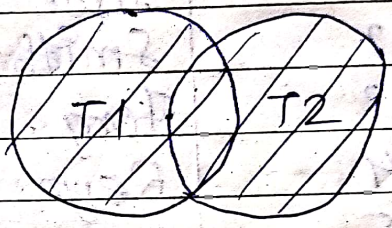
o/p

	Employee ID	Order ID	Order Date
1	101	4	2016-4-20
2	102	2	-11 — -18
3	104	1	-11 —
4	105	3	41 — -19

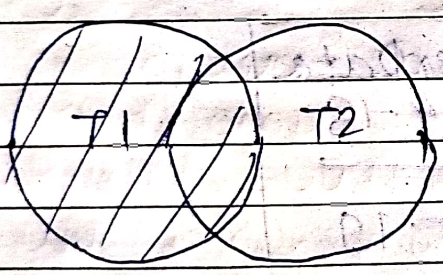
Inner Join



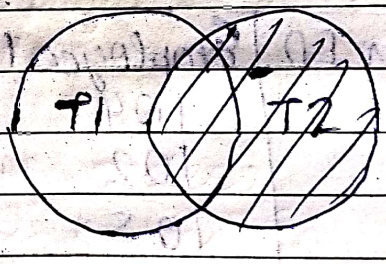
Full Join



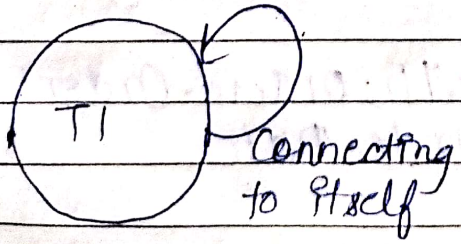
Left Join



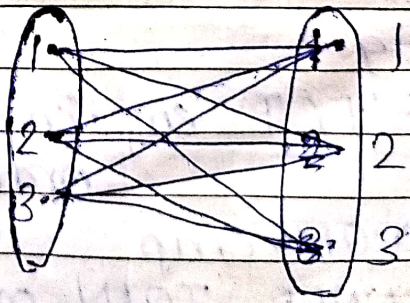
Right Join



Self Join



Cross Join



LEFT OUTER JOIN

→ Returns rows to the left (t1) even if there are no rows on the right (t2) of the join clause.
→ The result is NULL for rows on the right talele, when there is not match

Syntax:

```
SELECT <col-list>
FROM <table-name> as t1
LEFT OUTER JOIN <table-name2> as t2
ON t1.col_name = t2.col_name;
```

E.g.

```
Select emp.EmployeeID, orders.OrderID,
       orders.OrderDate
FROM emp
LEFT OUTER JOIN orders
ON Emp.EmployeeID = orders.EmployeeID
ORDER BY emp.EmployeeID;
```

o/p:

	EmployeeID	OrderID	OrderDate
1	101	4	2016-4-20
2	102	2	2016-4-18
3	103	NULL	NULL
4	104	1	2016-4-18
5	105	3	—11——19
6	106	NULL	NULL

Scanned with OKEN Scanner

RIGHT OUTER JOIN

→ Returns rows to the right (t2) rel^n even if there are no matching rows on the left (t1) rel^n (table)
→ result is NULL for rows on LEFT table, when there is no match

Syntax:
```
SELECT <column_list>
FROM <table_name1> as t1
RIGHT OUTER JOIN <table_name2> as t2
ON t1.col_name = t2.col_name;
```

E.g.:
```
SELECT Emp.EmployeeID, orders.OrderID,
       orders.OrderDate
FROM emp
RIGHT OUTER JOIN orders
ON Emp.EmployeeID = orders.EmployeeID
ORDER BY OrderID desc;
```

	EmployeeID	OrderID	OrderDate
1	NULL	5	2016-05-01
2	101	4	2016-04-20
3	105	3	—11— 19
4	102	2	—11— 18
5	104	1	—11—

Scanned with OKEN Scanner

FULL OUTER JOIN

→ Returns all records when there is a match on either left (t1) or right (t2) table records.

→ If there are rows on "t1" that don't match on "t2" or vice-versa, these rows would be listed as well

Syntax:

```
SELECT <column_list>
FROM <table_name1> as t1
FULL OUTER JOIN <table_name2> as t2
ON t1.col_name = t2.col_name;
```

e.g.

```
SELECT Emp.EmployeeID, orders.OrderID,
       orders.OrderDate
From emp
FULL OUTER JOIN orders
ON Emp.EmployeeID = orders.EmployeeID
ORDER BY Emp.EmployeeID;
```

o/p:

	Employee ID	Order ID	order date
1	NULL	5	2016-05-01
2	101	4	11--4-20
3	102	2	11 ---18
4	103	NULL	NULL
5	104	1	2016-4-18
6	105	3	11 ---19
7	106	NULL	NULL

CROSS JOIN

→ Displays all rows & columns of all tables
→ Also called Cartesian product

Syntax:

```
SELECT <col_list>
FROM <table_name1> as t1
CROSS JOIN <table_name2> as t2;
```

E.g.

```
SELECT emp.EmployeeID, orders.OrderID,
       orders.orderdate
FROM emp
CROSS JOIN orders
```

Scanned with OKEN Scanner

SUBQUERIES

A subquery is a select query that is enclosed inside another query. The inner select query is used to determine results of outer select query.

E.g.

```
                    Outer query
Select Dept from Employees
where salary = (select Max(Salary) from
                    Employees);
                    Inner query
```

Subqueries with select statement

Syntax:

SELECT

```
SELECT column_name
FROM table WHERE
column_name OPERATOR
(SELECT column_name FROM table [WHERE])
```

INSERT

The insert statement uses the data returned from the subquery to insert into another table

Syntax:

INSERT INTO table_name
SELECT * FROM table
WHERE VALUE OPERATOR
(Inner Query)

For e.g.

INSERT INTO orders SELECT product_id,
p_name, sell_price, FROM products
WHERE product_id IN (SELECT product_id
FROM products WHERE sell_price > 1000);

Products Table

prod. id	p_name	sell_price	p_type
101	Laptop	1400.5	Luxury
102	Camera	60.4	Non Luxury
103	Necklace	1200.6	"
104	Sofa	800.3	"

O/P (Orders Table)

order_id	product_sold	selling_price
101	Laptop	1400.5
103	Necklace	1200.6

Page No.:
Date. / /

UPDATE

Any Syntax:

UPDATE table SET
column_name = new value
WHERE OPERATOR VALUE
(SELECT COLUMN_NAME FROM ~~TABLE~~
TABLE_NAME) WHERE)

For e.g.

UPDATE EMPLOYEES SET SALARY = SALARY * 0.35
WHERE AGE IN (SELECT AGE FROM
EMPLOYEES_B WHERE AGE >= 27);

DELETE

Syntax:

~~DELETE FROM TALE_1~~
DELETE FROM TABLE_NANE WHERE
OPERTOR VALUE(INNER QUERY)

BUILT-IN Functions

Conversion → Data type casting & conv.
&
Logical — Aggregate
Math — String
~~Data~~ Date

Conversion:

~~CAST~~

~~⇒ Returns~~

⇒ Throws error on failure:

1) CAST(expr AS ddatatype)
2) CONVERT(datatype, expr)
2) PARSE(value AS datatype)

⇒ Returns null on failure

1) TRY_CAST(expr AS datatype)
2) TRY_CONVERT(d-type, expr)
3) TRY_PARSE(value AS datatype)

E.g.

```
SELECT CAST('10' AS INT) * 20 AS CAST_VER,
    CONVERT(int, '10') * 20 AS CONVERT_VER
```

Note: Both functions can be used interchangeably. In most situations CAST is ANSI-SQL complaint while CONVERT is not

```
SELECT TRY_CAST('A100' AS INT) * 20 AS
    TRYCAST_VER,
TRY_CONVERT(INT, 'X100') * 20 AS
    TRYCONV_VER
```

→ Displays result as null : conv. not possible

```
SELECT CAST('A100' AS INT) * 20 AS
    TRYCAST_VER
```

-- Throws error as conversion failed

Logical functions:

Func^n.	Desc.
→ CHOOSE(index, val_1, val_2, [val_n])	Return spec. index from list, else null if ind = 0 or greater than size
→ IIF(boolean_expr, true_value, false_value)	Return one of 2 val. based on boolean expression

Note: In choose func" index starts from 1

e.g. ①
SELECT CHOOSE(3, 'Test', 'Zest', 'Rest', 'Nest')

o/p: Rest

② Considers integer value only & returns A

~~SELECT~~
SELECT CHOOSE(1.4, 'A', 'B', 'C')

o/p: A

③
SELECT IIF(1>10, 'TRUE', 'FALSE')

o/p: FALSE

Maths functions

1) ~~ABSS~~
1) ABS(num)

2) RAND(seed) → returns pseudo random float value from 0 to 1(excl.)

3) EXP(float_expr)

4) FLOOR(num) → returns greatest integer less than or equal

5) ROUND(num, length)

→ SELECT ABS(-10) as 'ABS'

o/p: 10

→ SELECT RAND(0) as 'RAND'

o/p: 0.117169

Aggregate Functions

Performs calc. on a set of values.

Function:

Syntax	Desc.
AVG(expr)	null val. are ignored
MIN(expr)	used with num. col. only
SUM(expr)	
COUNT()	returns no. of items in grp
MAX(expr)	

Eg.

Get Avg, Min, Max, Sum of Salary from emp table

SELECT AVG(SAL) as AVGSAL, MIN(SAL) as MINSAL, MAX(SAL) as MAXSAL, SUM(SAL) as SUMSAL from Employee

-- Get emp count for each dept

```
SELECT Deptno,
COUNT (*) FROM Employee
GROUP BY Deptno
```

--- dept no. wise avg, min, max ---

```
SELECT DEPTNO, AVG(SAL) AS AVGSAL,
MIN(SAL) as MINSAL, MAX(SAL) as MAXSAL,
SUM(SAL) as SUMSAL from EMPLOYEE
GROUP BY DEPTNO
HAVING SUM(SAL) > 3000
```

1) LTRIM(char-exp.) → Removes leading spaces

2) RTRIM(char-exp) → Removes trailing spaces

3) CHARINDEX (exprToFind, exprToSearch, [start_location])
(substring)

→ Checks whether substring is present or not & returns starting loc. of found

4) CHAR(int_expr).

→ Converts int ASCII code to character

5) STR(expr.)

→ Returns char. data conv. from num. data

6) CONCAT(str1, str2, str3...)

→ Returns a string by concatenating 2 or more strings

Scanned with OKEN Scanner

E.g.

```
SELECT REPLACE('abcdefghicde', 'cde',
                                'xxx') as
'REPLACE', SUBSTRING('abcdef', 2, 3) as
SUBSTR, LEFT('abcdefg', 2) as 'LEFT',
RIGHT('abcdefg', 2) as 'RIGHT'
```

REPLACE	SUBSTR	LEFT	RIGHT
abxxxfghixxx	bcd	ab	fg

Date & Time Functions

Function	Description
1) SYSDATETIME()	Returns a datetime2(7) val. that contains date & time.
2) CURRENT_TIMESTAMP	Returns a datetime value
3) DATEPART(datepart, date)	Returns a integer that represents the specified datepart of the specified date
4) DAY(date)/MONTH (date)/YEAR(date)	Returns an integer that represents day/month/year part of specified date

Function	
5) DATEDIFF(datepart, startdate, enddate)	Returns the no. of date or time datepart boundaries that are crossed betⁿ 2 spec. dates
6) DATEADD(datepart, number, date)	Returns a new datetime value by adding an interval to the spec. datepart of specified date

E.g.

```
SELECT
SYSDATETIME() as 'SYSDATETIME',
CURRENT_TIMESTAMP as 'TIMESTAMP',
DATEPART(year, '12-DEC-2017') AS 'DATEPART',
DATEDIFF(mm, '12/31/2015', '10/23/2016') as
'DATEDIFF',
DATEADD(mm, 2, '12/31/2015') as 'DATEADD'
```

Res:

SYSDATETIME	TIMESTAMP
2017-12-24 09:27:23.1501314	2017-12-24 09:27:23.150

DATEPART	DATEDIFF	DATEADD
2017	10	2016-02-29 00:00:00.000

Scanned with OKEN Scanner

STORED PROCEDURE :

- A stored procedure is a set of sql statements with a name, that has been created & stored on the database.

- Stored procedure can be defined as the set of logical group of SQL statements which are grouped to perform a specific task.

→ Accept input parameters

→ Return multiple values (out parameters)

→ Contain programming statements

→ can call other stored procedures / functions

→ Return status to indicate success or failure

Note: Use Alter instead of create to make changes in the procedure

Benifits:

Reduced Server/Client Network Traffic
→ All commands executed in the single batch of code

→ Stronger security
The Execute as can be specified to enable impersonating another user to perform certain db tasks without providing direct permission to procedure

→ Reuse of code
The code can be called from other procedures

→ Easier maintainance
Changes need to be done within stored procedure which handles all db operations & no changes are reqd in client appl.

→ Improved performance
All procedures by default are compiled on the db the first time it is executed & maintains the execution plan for subsequent calls.

Scanned with OKEN Scanner

For Execution:

```
EXEC procedure_name;
```

For e.g.

```
CREATE PROCEDURE SelectAllCustomers
  @City nvarchar(30), @PostalCode nvarchar(10)
AS
SELECT * FROM Customers where City = @City
  AND PostalCode = @PostalCode
  GO;
```

Execution:

```
EXEC SelectAllCustomers @City = 'London',
     @PostalCode = 'WA1 1DP';
```

Comments

```
-- Single line

/* Multi-line */
```

Scanned with OKEN Scanner

MySQL Stored Procedure

```
DELIMITER &&
CREATE PROCEDURE procedure_name
  [[IN|OUT|INOUT] parameter_name datatype
  [parameter datatype]]

BEGIN
    Declaration section
    Executable section
END &&
DDELIMITER ;
```

IN parameter:
Takes parameter as input. The parameter's value is always protected

OUT parameter:
Used to pass parameter as output.
It's value can be changed inside the procedure & new value is passed back to calling program.

INOUT parameters:
Calling program can pass the args. & procedure can modify it, & then new value is passed back to calling program.

CALL procedure

Syntax:

CALL procedure_name [parameter(s)]

Procedure without parameter

E.g. DELIMITER &&

Create Procedure get_merit_student()
Begin
Select * From student_info WHERE
marks > 70;
Select COUNT(stud_code) AS Total_student
From student_info;
End &&

DELIMITER;

O/P:

Original Table:

	stud_id	stud_code	stud_name	subject	marks
1		101			68
2		102			70
3		103			70
4					90
5					85
6					92
7					83
8		108			85
10		110			67
10					

CALL the procedure

CALL get_merit_student();

o/p:

stud_id	stud_code	stud_name	subject	marks
4	104			90
5	105			85
6				92
7				83
8	108			85

Total_student
9

Procedure with In parameter

```
Delimiter &&
Create Procedure get_student(IN var1 INT)
Begin
    Select * From student_info LIMIT var1;
    Select COUNT(stud_code) AS Total_student
        From student_info;
End &&
Delimiter ;
CALL get_student(4);
```

stud_id	stud_code	stud_name	subject	marks
1	101			
2	102			
3	103			
4	104			

Total_student
9

Procedure with OUT parameter :-

For e.g. ;

```
Delimiter &&

Create Procedure display_max_mark(
                    OUT highest_mark INT)

Begin
    select MAX(marks) into highest_mark
        from student_info;

End ; End &&

Delimiter ;

CALL display_max_mark(@M)

SELECT @M
```

o/p:

@M
92

Date. / /

Procedure with INOUT parameter:

```
Delimiter &&
Create Procedure display_marks(INOUT var1 INT)
    Begin
        Select marks into var1 from student_info
            where stud_id=1;
    End &&
Delimiter ;
SET @M='3';
CALL display_marks(@M);
SELECT @M;
```

@M
70

Scanned with OKEN Scanner