SQL

Structured Query Language (SQL) is categorized into:

1. DML Data Manipulation Language
   INSERT, DELETE, UPDATE,

2. Query (DML)
   SELECT

3. DDL Data Definition Language
   CREATE, DROP, ALTER, RENAME, TRUNCATE

4. DCL Data Definition Language
   GRANT, DENY, REVOKE

5. Transaction
   COMMIT, ROLLBACK, SAVEPOINT

DML

Example:

```
SELECT    "Field (Attributes)"
FROM      "TABLE"
```

SELECT: Chooses the field
FROM: Determines the table
DISTINCT: Eliminates duplication in the resulted query
ALL: Allows duplication in the resulted query
AS: Shows the field name as the name that you put after it.

Example:

```
SELECT    ALL Emp-name As Name,
          DISTINCT emp-address, Salary,
          Salary * 12 AS Yearly
FROM      Employee
```

Math Query:
You can use the math operators (+ - * /) with the SELECT statement

Example:

```
SELECT    Amount, Amount x 0.07 AS Discount,
          Amount – Amount x 0.07 AS Total
FROM      Loan
```

***** :	Retrieving all fields of a table
WHERE:	Displays requested data with some terms and conditions using:
	a. Comparison Operators **(< <= > >= <>),**
	b. Logical Operators AND, OR, NOT
	c. String Operators LIKE, NOT LIKE, BETWEEN, NOT BETWEEN, IN, NOT IN, (% = String, _ = Character)
LIKE:	For detailed string comparison.
NOT LIKE:	For detailed string comparison exception.
BETWEEN:	Specifies the area of constraint
NOT BETWEEN:	Specifies the area of "Except"
IN:	Chooses the specified records from the parentheses after it.
NOT IN:	Chooses the specified records from anything but the parentheses after it.

Example:

```
SELECT    *
FROM      Loan
WHERE     Amount BETWEEN 0 AND 500
          AND Name LIKE "Ba_h%"
```

|| :	Concatenation
. :	Object Indicator
ORDER BY:	Orders the records according to a specified field (**ASC**: Ascending is default– **DESC**: Descending)

Example:

```
SELECT    EmpName || ' ' || Sal || ' ' || Employee.EID
FROM      Employee, Department
WHERE     EmpName IN ('Ali', 'Bassem', 'Ahmad')
          AND Employee.EID = Department.EID
ORDER BY  EmpName ASC
```

Set Operations

EXCEPT:	to differentiate queries
EXCEPT ALL:	to differentiate queries (allowing duplication)
UNION:	to union queries
UNION ALL:	to union queries (allowing duplication)
INTERSECT:	to intersect queries
INTERSECT ALL:	to intersect queries (allowing duplication)

Example:

```
(SELECT   DISTINCT Customer-Name
FROM      Depositor)
EXCEPT
(SELECT   Customer-Name
FROM      Borrower)
```

Aggregate Functions (Group Functions)
Functions that take a group of values and returns one value
SUM – AVG – MAX – MIN – COUNT

SUM	Gets the sum of record values (numeric only)
AVG	Gets the average of record values (numeric only)
MAX	Gets the maximum of record values
MIN	Gets the minimum of record values
COUNT	Counts records

Example:

```
SELECT    SUM (Balance)
FROM      Account
```

GROUP BY: Groups the fields in a particular field name
HAVING: Used with aggregation functions. Works like WHERE.

Note:
1. HAVING always comes after GROUP BY if necessary, never before.
2. HAVING always comes with GROUP BY, not WHERE.
3. With aggregate functions, we always use HAVING, not WHERE.

Example:

```
SELECT    Branch-name, SUM (balance)
FROM      Account
GROUP BY  Branch-Name
HAVING    SUM (Balance) >= 10,000
```

IS NULL	To check if the value of a record is Null.
IS NOT NULL	To check if the value of a record is not Null

Usual Query Statements Succession:

SELECT	**Fields name**
FROM	**Table Name**
WHERE	**Condition**
GROUP BY	**Fields**
HAVING	**Condition**
ORDER BY	**Fields**

Example:

```
SELECT    LID, Amount
FROM      Loan
WHERE     Amount IS Null
```

Nested SQL:

```
SELECT    CName
FROM      Depositor
WHERE     CName IN  ( SELECT    CName
                      FROM      Borrower )
```

(Very much like the INTERSECT. If we use NOT IN, it becomes like EXCEPT)

```
INSERT INTO     Student (SID, SName, Address)
                Values (13, "Nadim", "Abdoon")

DELETE FROM     Depositor
WHERE           CID = 43

UPDATE          Account
SET             Balance = 1.06 x balance
WHERE           Balance > 10,000
```

INSERT: It is applied over one relation to insert only one row
DELETE: It is applied over one relation to insert one or more rows
UPDATE: It is applied over one relation to update one or more fields
SET: Specifies the data to be updated by the UPDATE statement

DDL

Domain Types:

- char (n) fixed length
- varchar2 (n) variable length
- number (p,d)
- Int, TinyInt, SmallInt, BigInt
- date
- Money
- Raw
- Long Raw
- BLOB

Integrity Constraints:

- Primary Key
- Not Null
- Check (P)

Example:

Check (balance >= 0): withdrawing from the account must be greater than $0

CREATE	Creates tables, with the help of the TABLE statement
DROP	Drops tables or fields
ALTER	Alters table structure, with the help of the TABLE statement
MODIFY	Modifies fields in a table after using ALTER
ADD	Adds fields to a table after using ALTER

PRIMARY KEY	Sets the primary key of the table with the help of REFERENCE
FOREIGN KEY	Sets the foreign key of the table with the help of REFERENCE
UNIQUE	Makes one of the fields a unique key
CHECK	Sets a condition on a certain field when creating a table
CONSTRAINT	Sets a constraint on a certain field when creating a table
REFERENCE	To refer to a particular table
NOT NULL	To force the user to enter a value

TRUNCATE	Frees the hard disk from an existing useless disk space.

Note:
The difference between the Primary Key and Unique is that both must not contain repetition

```
CREATE TABLE    Old-Sales (SalesID BigInt, SalesAmount Money)
```

```
CREATE TABLE    Student
(SID            Number          Primary Key,
 Name           char (50)       NOT Null,
 Address        varchar2 (80)                   )
```

Example:

```
CREATE TABLE    Branch
(Branch-name    char(20),
Branch-city     varchar(20),
Asset           INTEGER,
CONSTRAINT      C1      PRIMARY KEY (Branch-name),
Check (Asset >= 0                   )
```

```
CREATE TABLE Loan
(LID            Number (4)      Primary key,
Amount          Number (5,2)    NOT NULL,
Branch-name     varchar (25)    NOT NULL,
FOREIGN KEY (Branch-name)       REFERENCE Branch )
```

Referential Integrity Constraints:

PRIMARY KEY		Can not be Null
FOREIGN KEY	REFERENCE table-name	Can be Null
UNIQUE		Can be Null
CHECK	CHECK (attribute-list & Condition)	

```
DROP TABLE      Student
```

(Must delete all relationships with the table before deleting it)

```
ALTER TABLE     Branch
(ADD            Asset               Number (10, 2)          NOT Null
MODIFY          Branch-name         varchar2 (155)
DROP            Location
ENABLE          CONSTRAINT          Banking-branch-pk                )
```

```
CONSTRAINT      Constraint-name     PRIMARY KEY (Attribute List)
```

```
TRUNCATE TABLE      Student
```

```
FOREIGN KEY (SID)       REFERENCE Student
[ON DELETE CASCADE]     Delete data in related PK & FK
[ON UPDATE CASCADE]     Update data in related PK & FK
```

TABLE It is used to create, alter or drop tables
VIEW It is a special way to view data in a specific method
SYNONYM To make a copy from a certain table with a different name
SEQUENCE Gives an instance of a table for a certain field a specific value.

Example:

```
CREATE VIEW     V1
SELECT          OName, OCode
FROM            Operation
WHERE           Cost < 1000

SELECT          *
FROM            V1

SELECT          OName
FROM            V1

DROP VIEW       V1
```

Example:

```
CREATE SYNONYM Oper  FOR    Operation
```

```
SELECT          DName
FROM            Doctor
WHERE           D.Sal > 600

DROP SYNONYM    Oper
```

Example:

```
CREATE SEQUENCE Seq1        START WITH 56

INSERT INTO     Doctor
```

VALUES (Seq1.nextval, 'Dr Hisham', '07512', 300)

SELECT Seq1.curval, Seq1.nextval
FROM Dual

DCL

GRANT: Grants access to a user to specific data
DENY: Denies access of a user to specific data
REVOKE: Removes any GRANT's or DENY's from a specific user.

GRANT CREATE ROLE TO Public

GRANT INSERT ON Operation TO R1

REVOKE INSERT ON Operation TO R1

GRANT EXECUTE ON Get_Name TO Student1

TL

COMMIT Saves the current situation of the data (all data)
ROLLBACK Retrieves the situation of all data from the last COMMIT done
SAVEPOINT Saves the situation of the data as a point for recovery purposes

SQL Language Basics

Naming Objects

- Avoid a name that matches a reserved keyword
- When naming a name with a space, use the square brackets (e.g. [Order Details]) in SQL statements (whereas "Order" is a reserved keyword)

Comment
Block Comment
/*
…….
*/

Inline Comment
-- This is an inline comment

Variables
DECLARE @MyVar TinyInt

```
SET       @MyVar = 5
SELECT    @MyVar    AS    MyVariableValue
```

Batches

```
CREATE    VIEW      SalesAug
AS
SELECT    *
FROM      Orders
WHERE     Month (Order Date) = 8

GO

CREATE    VIEW      SalesSept
AS
SELECT    *
FROM      Orders
WHERE     Month (Order Date) = 9
```

NOTE: GO statement prevents an error from occurring by executing the first CREATE then the second CREATE, instead of both of them at once.

SQL Advanced Features

Expressions

```
+ - * / AND OR NOT = < > LIKE +

SELECT    OrderID, (Unit-Price * Quantity)    AS    Total-Price
FROM      [Order Details]

SELECT    *
FROM      Customers
WHERE     City = 'London'      OR    Country = 'Germany'

SELECT    *
FROM      Customers
WHERE     Company-Name      LIKE  'A%'
```

Exec

```
DECLARE   @MyString    varchar(20)
SET       @ MyString = 'Customer'
EXEC      ('SELECT * FROM' + @MyString )
```

Built-in Functions

```
RIGHT, LEFT, SUBSTRING
AVG, MIN, MAX, SUM, COUNT
@@version, DB_Name ()
```

```
SELECT      Company-Name, LEFT (Compane-Name, 4) AS CoID
FROM        Customers

SELECT      Company-Name, RIGHT (Compane-Name, 4) AS CoID
FROM        Customers

SELECT      Company-Name, SUBSTRING (Compane-Name, 2, 2) AS CoID
FROM        Customers

SELECT      @@VERSION

SELECT      DB_Name ()
```

Control Flow

```
IF..ELSE    WHILE       CASE
```

If..Else

```
DECLARE     @MyVar      SmallInt
SET         @MyVar = 3

IF      @MyVar = (9/3)
        BEGIN
            SELECT *    FROM Products
        END

ELSE
        BEGIN
            SELECT *    FROM Categories
        END
```

While

```
WHILE (SELECT MIN (Unit-Price)    FROM Products) > 1
        BEGIN
            UPDATE      Products        SET Unit-Price = (Unit-Price / 2)
            PRINT       'New prices have been set'
        END
```

Case

```
SELECT      Product-Name, Supplier-ID,  CASE Discontinued
                                            WHEN 1 THEN 'Yes'
                                            WHEN 0 THEN 'No'
                                        END AS Discontinued
FROM Products
```