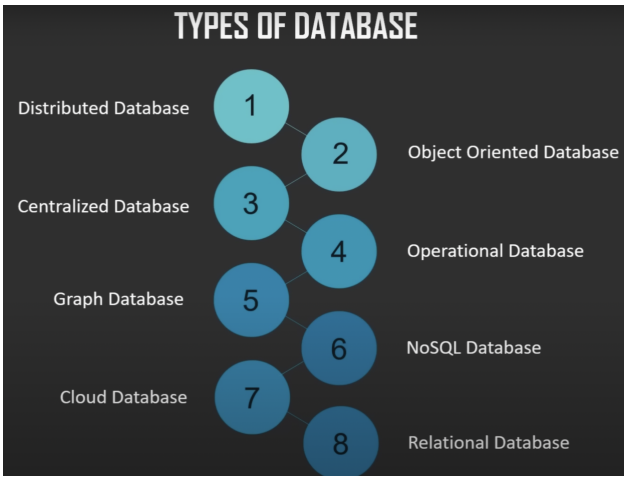
SQL

Types of Databases



https://www.w3schools.com/sql/default.asp
Complete this course from w3 schools for basics.

CHAPTER - 1 : Oracle Basics

1. **select distinct**
2. **where**
3. **and, or, not**
4. **order by**
5. **insert into**
6. **INSERT ALL INTO** -- The Oracle INSERT ALL statement is used to add multiple rows with a single INSERT statement. The rows can be inserted into one table or multiple tables using only one SQL command.

```
Ex: INSERT ALL
      INTO suppliers (supplier_id, supplier_name) VALUES (1000, 'IBM')
      INTO suppliers (supplier_id, supplier_name) VALUES (2000,
    'Microsoft')
      INTO suppliers (supplier_id, supplier_name) VALUES (3000,
    'Google')
    SELECT * FROM dual;
    INSERT ALL
      INTO suppliers (supplier_id, supplier_name) VALUES (1000, 'IBM')
      INTO suppliers (supplier_id, supplier_name) VALUES (2000,
    'Microsoft')
```

INTO customers (customer_id, customer_name, city) VALUES (999999, 'Anderson Construction', 'New York')
SELECT * FROM dual;

In the below example customers and orders are 2 tables and based on the data in these 2 tables, if we want to insert the data in to 3 more tables (at a time) small_orders, medium_orders , large_orders and special_orders based on different conditions then we can define the insert all in to as follows.

```
INSERT ALL
   WHEN ottl < 100000 THEN
      INTO small_orders
         VALUES(oid, ottl, sid, cid)
   WHEN ottl > 100000 and ottl < 200000 THEN
      INTO medium_orders
         VALUES(oid, ottl, sid, cid)
   WHEN ottl > 200000 THEN
      into large_orders
         VALUES(oid, ottl, sid, cid)
   WHEN ottl > 290000 THEN
      INTO special_orders
   SELECT o.order_id oid, o.customer_id cid, o.order_total ottl,
      o.sales_rep_id sid, c.credit_limit cl, c.cust_email cem
      FROM orders o, customers c
      WHERE o.customer_id = c.customer_id;
```

7. **null values, is null, is not null**
8. **update** --- The Oracle UPDATE statement is used to update existing records in a table in an Oracle database. There are 2 syntaxes for an update query in Oracle depending on whether you are performing a traditional update or updating one table with data from another table.

   (A) UPDATE statement when updating one table in Oracle/PLSQL

   update customers
   set first_name = 'John', salary,
   salary = 10,000
   where last_name = 'Stirgus';

   (B) UPDATE statement when updating one table with data from another table

   UPDATE customers
   SET c_details = (SELECT contract_date
   FROM suppliers
   WHERE suppliers.supplier_name = customers.customer_name)

```
WHERE customer_id < 1000;
```

9. **delete** --- delete from customers
   where first_name = 'John' and last_name = 'Stirgus';
10. **fetch first 10 rows only**
11. **min, max**
12. **Aggregate functions (count, avg, sum etc)**
13. **like** --- where first_name like 'a%', '%a', '%a%', '_a%', '%_a%' etc.
14. **wildcard characters** -- A wildcard character is used to substitute one or more characters in a string.

    LIKE '[2cb]%' --which means first character can any one of the 2, c, b.
    '[a-d]%' -- first character can be any one of the a,b,c,d
    '[!a-d]%' -- first character should not be any of a,b,c,d
    NOT like '[abcd]%' -- same as above (first character should not be any of a,b,c,d)

    Escape Characters:
    ```
    SELECT * FROM suppliers
    WHERE supplier_name LIKE 'Water!%' ESCAPE '!';
    ```
    -- It searches for the string Water%. Here the % symbol is treated as a character and not a wild character. This is how we escape a character.
    ```
    SELECT * FROM suppliers
    WHERE supplier_name LIKE 'H%!%' ESCAPE '!';
    ```
    -- It searches for the string that starts with H and then ends with %. The first % is the wildcharacter, whereas the second % is preceeded by an escape character symbol. So, that is treated as a character and not as a wild character. NOTE: the ESCAPE'!' is also important here.

15. **REGEXP_LIKE** -- use the below link to learn this.
    https://www.techonthenet.com/oracle/regexp_like.php
16. **IN/NOT IN** -- The IN operator allow you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
SELECT * FROM Customers
```

```
WHERE Country IN (SELECT Country FROM Suppliers);
```

17. **BETWEEN / NOT BETWEEN** -- The `BETWEEN` operator selects values within a given range. The values can be numbers, text, or dates. The `BETWEEN` operator is inclusive: begin and end values are included.

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND
'Mozzarella di Giovanni'
ORDER BY productid;
--In this example, all the records will be fetched where the
product name are alphabetically between the 'Carnarvon
Tigers' & 'Mozzarella di Giovanni'
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-
31';
```

18. **Aliases** -- SQL aliases are used to give a table, or a column in a table, a temporary name.

```
-- Aliases are often used to make column names more readable.

-- An alias only exists for the duration of that query.

-- An alias is created with the AS keyword.

SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Note: It requires double quotation marks or square brackets if the alias name contains spaces like below:

```
SELECT CustomerName AS Customer, ContactName AS [Contact
Person]
FROM Customers;
SELECT CustomerName, (Address || ', ' || PostalCode || ' ' || City || ', ' ||
Country) AS Address
FROM Customers;

SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
```

```
WHERE c.CustomerName='Around the Horn' AND
c.CustomerID=o.CustomerID;
```

19. **EXISTS** -- The Oracle EXISTS condition is used in combination with a subquery and is considered "to be met" if the subquery returns at least one row. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement.

Note: Oracle SQL statements that use the Oracle EXISTS condition are very inefficient since the sub-query is RE-RUN for EVERY row in the outer query's table. There are more efficient ways to write most queries, that do not use the EXISTS` condition.

This basically checks that for every record in the outer query if there is a record in the inner query or not.

Ex: Get the employee id and name of the professor who is an advisor for at least one female student

```
select emp_id, name from professor as p
where exists (select roll_no from students as s
            where s.advisor = p.emp_id
            and s.sex = 'F');
```

20. **GROUP BY** -- The Oracle GROUP BY clause is used in a SELECT statement to collect data across multiple records and group the results by one or more columns. 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.

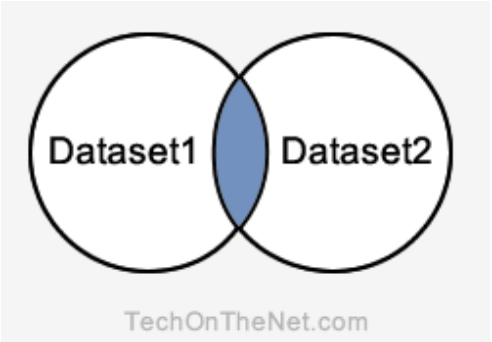
EX:
```
SELECT product, SUM(sale) AS "Total sales"
    FROM order_details
    GROUP BY product;
    SELECT category, COUNT(*) AS "Number of suppliers"
    FROM suppliers
    WHERE available_products > 45
    GROUP BY category;
```

This GROUP BY example uses the COUNT function to return the category and the number of suppliers (in that category) that have over 45 available_products.

NOTE: GROUP BY always comes after the where clause

SELECT department, MAX(salary) AS "Highest salary"
FROM employees
GROUP BY department;
This GROUP BY example uses the MAX function to return the name of each department and the maximum salary in the department.

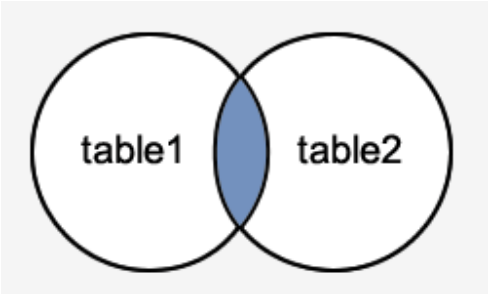
21. **HAVING** -- The Oracle HAVING clause is used in combination with the GROUP BY clause to restrict the groups of returned rows to only those whose the condition is TRUE. The `HAVING` clause was added to SQL because the `WHERE` keyword cannot be used with aggregate functions. This is a filter on top of the group by result set. This is a further condition applied only to the aggregated results to restrict the groups of returned rows. Only those groups whose condition evaluates to TRUE will be included in the result set.
    Ex: SELECT department, SUM(sales) AS "Total sales"
    FROM order_details
    GROUP BY department
    HAVING SUM(sales) > 25000;
    The Oracle HAVING clause will filter the results so that only departments with sales greater than $25,000 will be returned.
    SELECT department, COUNT(*) AS "Number of employees"
    FROM employees
    WHERE salary < 49500
    GROUP BY department
    HAVING COUNT(*) > 10;
    NOTE: GROUP BY always comes after the where clause
22. **INTERSECT** -- The Oracle INTERSECT operator is used to return the results of 2 or more SELECT statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.



The INTERSECT query will return the records in the blue shaded area. These are the records that exist in both Dataset1 and Dataset2. Each SELECT statement within the INTERSECT must have the same number of fields in the result sets with similar data types.

```
EX: SELECT supplier_id
    FROM suppliers
    WHERE supplier_id <= 99
    INTERSECT
    SELECT supplier_id
    FROM orders
    WHERE quantity > 25;
```

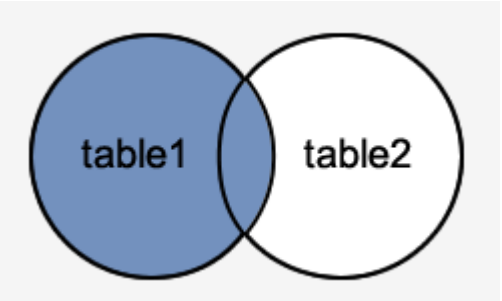
23. **INNER JOIN --** There are 4 different types of Oracle joins:
    Oracle INNER JOIN (or sometimes called simple join)
    Oracle LEFT OUTER JOIN (or sometimes called LEFT JOIN)
    Oracle RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)
    Oracle FULL OUTER JOIN (or sometimes called FULL JOIN)
    There is one more type called SELF JOIN.



The Oracle INNER JOIN would return the records where *table1* and *table2* intersect.

```
EX: SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
    FROM suppliers
    INNER JOIN orders
    ON suppliers.supplier_id = orders.supplier_id;
```

24. **LEFT OUTER JOIN --** This type of join returns all rows from the LEFT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).



The Oracle LEFT OUTER JOIN would return the all records from *table1* and only those records from *table2* that intersect with *table1*.

EX: SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date

FROM suppliers

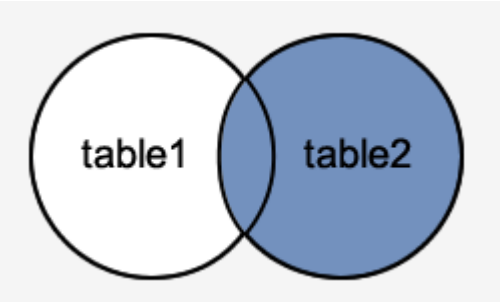
LEFT OUTER JOIN orders

ON suppliers.supplier_id = orders.supplier_id;

This LEFT OUTER JOIN example would return all rows from the suppliers table and only those rows from the orders table where the joined fields are equal.

If a supplier_id value in the suppliers table does not exist in the orders table, all fields in the orders table will display as <null> in the result set.

25. **RIGHT OUTER JOIN**-- exactly opposite to the left outer join



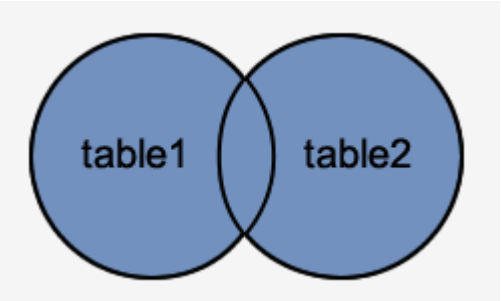
Ex: SELECT orders.order_id, orders.order_date, suppliers.supplier_name

FROM suppliers

RIGHT OUTER JOIN orders

ON suppliers.supplier_id = orders.supplier_id;

26. **FULL OUTER JOIN** -- This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.



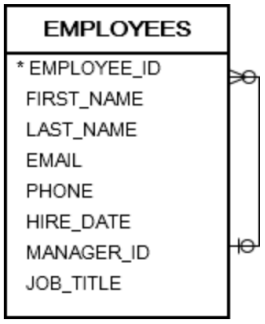
EX: SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
FULL OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;

27. **Self Join** -- A self join is a join that joins a table with itself. A self join is useful for comparing rows within a table or querying hierarchical data. A self join uses other joins such as inner join and left join. In addition, it uses the table alias to assign the table different names in the same query.

NOTE: Referencing the same table more than once in a query without using table aliases cause an error.

A) Using Oracle self join to query hierarchical data example

See the following `employees` table in the sample database.



The `employees` table stores personal information such as id, name, job title. In addition, it has the `manager_id` column that stores the reporting lines between employees.

The President of the company, who does not report to anyone, has a NULL value in the `manager_id` column. Other employees, who have a manager, have a numeric value in the `manager_id` column, which indicates the `id` of the manager.

To retrieve the employee and manager data from the `employees` table, you use a self join as shown in the following statement:

```
select  (e.first_name || ' ' || e.last_name) as employee,
(m.first_name || ' '|| m.last_name) as manager, e.job_title
from employees e
inner join employees m
on m.employee_id = e.manager_id
order by manager;
```

EMPLOYEE	MANAGER	JOB_TITLE
Tommy Bailey		President
Evie Harrison	Ava Sullivan	Sales Representative
Grace Ellis	Ava Sullivan	Sales Representative
Lily Fisher	Ava Sullivan	Sales Representative
Sophia Reynolds	Ava Sullivan	Sales Representative
Sophie Owens	Ava Sullivan	Sales Representative
Poppy Jordan	Ava Sullivan	Sales Representative
Louie Richardson	Blake Cooper	Programmer
Georgia Mills	Callum Jenkins	Shipping Clerk
Maisie Nichols	Callum Jenkins	Shipping Clerk
Eleanor Grant	Callum Jenkins	Shipping Clerk
Hannah Knight	Callum Jenkins	Shipping Clerk
Connor Haves	Callum Jenkins	Stock Clerk

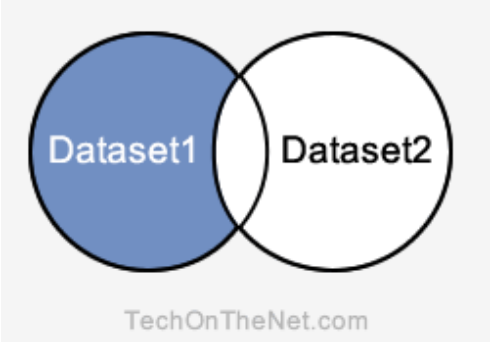
B) Using Oracle self join to compare rows within the same table example

Ex: The following statement finds all employees who have the same hire dates:

```
select e1. hire_date,
(e1.first_name || ' ' || e1.last_name) as Employee1,
(e2.first_name || ' ' || e2.last_name) as Employee2
from employees e1
inner join employees e2
on e1.employee_id > e2.employee_id
and e1.hire_date = e2.hire_date
order by e1.hire_date desc, Employee1, Employee2;
```

HIRE_DATE	EMPLOYEE1	EMPLOYEE2
07-DEC-16	Rory Kelly	Elliot Brooks
28-SEP-16	Kai Long	Tyler Ramirez
20-AUG-16	Sophia Reynolds	Austin Flores
17-AUG-16	Amelie Hudson	Mohammad Peterson
21-JUN-16	Bella Stone	Ivy Burns
14-JUN-16	Jasmine Hunt	Seth Foster
07-JUN-16	Gracie Gardner	Harper Spencer
07-JUN-16	Rose Stephens	Gracie Gardner
07-JUN-16	Rose Stephens	Harper Spencer
07-JUN-16	Summer Payne	Gracie Gardner
07-JUN-16	Summer Payne	Harper Spencer
07-JUN-16	Summer Payne	Rose Stephens
21-APR-16	Elsie Henry	Matilda Stevens
10-APR-16	Reggie Simmons	Liam Henderson
24-MAR-16	Lucy Crawford	Sienna Simpson
24-MAR-16	Lucy Crawford	Sophie Owens
24-MAR-16	Rosie Morales	Lucy Crawford
24-MAR-16	Rosie Morales	Sienna Simpson
24-MAR-16	Rosie Morales	Sophie Owens
24-MAR-16	Sienna Simpson	Sophie Owens

28. **MINUS** -- The Oracle MINUS operator is used to return all rows in the first SELECT statement that are not returned by the second SELECT statement. Each SELECT statement will define a dataset. The MINUS operator will retrieve all records from the first dataset and then remove from the results all records from the second dataset.



NOTE: There must be same number of expressions in both SELECT statements and have similar data types.

```
EX: SELECT supplier_id, supplier_name
    FROM suppliers
    WHERE state = 'Florida'
    MINUS
    SELECT company_id, company_name
    FROM companies
    WHERE company_id <= 400
```

```
ORDER BY 2;
```

29. **PIVOT** -- Use the link to read this topic.
    https://www.techonthenet.com/oracle/pivot.php
30. **Subqueries** -- In Oracle, a subquery is a query within a query. You can create subqueries within your SQL statements. These subqueries can reside in the WHERE clause, the FROM clause, or the SELECT clause.

WHERE clause

Most often, the subquery will be found in the WHERE clause. These subqueries are also called nested subqueries.
For example:

```
SELECT *
FROM all_tables tabs
WHERE tabs.table_name IN (SELECT cols.table_name
                          FROM all_tab_columns cols
                          WHERE cols.column_name =
'SUPPLIER_ID');
```

Limitation: Oracle allows up to 255 levels of subqueries in the WHERE clause.

FROM clause

A subquery can also be found in the FROM clause. These are called **inline views**.
For example:

```
SELECT suppliers.name, subquery1.total_amt
FROM suppliers,   (SELECT supplier_id, SUM(orders.amount) AS total_amt
                   FROM orders
                   GROUP BY supplier_id) subquery1
WHERE subquery1.supplier_id = suppliers.supplier_id;
```

In this example, we've created a subquery in the FROM clause
This subquery has been aliased with the name *subquery1*. This will be the name used to reference this subquery or any of its fields.

Limitations

Oracle allows an unlimited number of subqueries in the FROM clause.

SELECT clause

A subquery can also be found in the SELECT clause.
For example:

```
SELECT tbls.owner, tbls.table_name,  (SELECT COUNT(column_name) AS
total_columns
                                        FROM all_tab_columns cols
                                       WHERE cols.owner = tbls.owner
                                       AND cols.table_name =
                              tbls.table_name) subquery2
FROM all_tables tbls;
```

31. **TRUNCATE TABLE** -- The TRUNCATE TABLE statement is used to remove all records from a table in Oracle. It performs the same function as a DELETE statement without a WHERE clause.

    NOTE & WARNING: If you truncate a table, the TRUNCATE TABLE statement can not be rolled back. The main difference between DELETE and TRUNCATE TABLE is that you can roll back the DELETE statement if you choose, but you can't roll back the TRUNCATE TABLE statement.

    EX: TRUNCATE TABLE customers;

32. **UNION --** The Oracle UNION operator is used to combine the result sets of 2 or more Oracle SELECT statements. **It removes duplicate rows between the various SELECT statements (means only one copy of the common records will present in the result)**. Each SELECT statement within the UNION operator must have the same number of fields in the result sets with similar data types.

    Ex:
    ```
    SELECT supplier_id, supplier_name
        FROM suppliers
        WHERE supplier_id <= 500
        UNION
        SELECT company_id, company_name
        FROM companies
        WHERE company_name = 'Apple'
        ORDER BY 2;
    ```

    Union operator helps us to implement two queries with two criteria on the same set of tables and join their results.

For example when you want to see results of no.of records which are updated in the last 13 days and no.of records which are updated 13 days before. We can answer it with the following query.

```
SELECT a.code AS Code, a.name AS Name, COUNT(b.Ncode)
FROM cdmaster a, nmmaster b
WHERE a.code = b.code
AND a.status = 1
AND b.status = 1
AND b.Ncode <> 'a10'
AND TRUNC(a.last_updated_date) <= TRUNC(sysdate - 13)
GROUP BY a.code, a.name
UNION
SELECT a.code AS Code, a.name AS Name, COUNT(b.Ncode)
FROM cdmaster a, nmmaster b
WHERE a.code = b.code
AND a.status = 1
AND b.status = 1
AND b.Ncode <> 'a10'
AND TRUNC(a.last_updated_date) > TRUNC(sysdate - 13)
GROUP BY a.code, a.name;
```

The Oracle UNION operator allows you to perform a count based on one set of criteria.
TRUNC(a.last_updated_date) <= TRUNC(sysdate - 13)

As well as perform a count based on another set of criteria.
TRUNC(a.last_updated_date) > TRUNC(sysdate - 13)

33. **UNION ALL** -- The Oracle UNION ALL operator is used to combine the result sets of 2 or more SELECT statements. It returns all rows from the query and it does not remove duplicate rows between the various SELECT statements. In the UNION operator duplicates will be removed, whereas in the UNION ALL duplicates will be present in the result set.

CHAPTER - 2 : Oracle Advanced
(https://www.techonthenet.com/oracle/tables/alter_table.php)

1. **ALTER TABLE** -- The Oracle ALTER TABLE statement is used to add, modify, or drop/delete columns in a table. The Oracle ALTER TABLE statement is also used to rename a table.