DDL, DML, DCL and TCL Commands in Sql Server

DML

- DML is abbreviation of Data Manipulation Language. It is used to retrieve, store, modify, delete, insert and update data in database.
- SELECT – Retrieves data from a table
- INSERT - Inserts data into a table
- UPDATE – Updates existing data into a table
- DELETE – Deletes all records from a table

DDL

- DDL is abbreviation of Data Definition Language. It is used to create and modify the structure of database objects in database.
- CREATE – Creates objects in the database
- ALTER – Alters objects of the database
- DROP – Deletes objects of the database
- TRUNCATE – Deletes all records from a table and resets table identity to initial value.

DCL

DCL is abbreviation of Data Control Language. It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

GRANT – Gives user's access privileges to database
REVOKE – Withdraws user's access privileges to database given with the GRANT command

TCL

TCL is abbreviation of Transactional Control Language. It is used to manage different transactions occurring within a database.

COMMIT – Saves work done in transactions
ROLLBACK – Restores database to original state since the last COMMIT command in transactions
SAVE TRANSACTION – Sets a savepoint within a transaction

SQL Server IDENTITY example

- To create an identity column for a table, you use the IDENTITY property as follows:

  IDENTITY[(seed,increment)]

- In this syntax:

  The seed is the value of the first row loaded into the table.

  The increment is the incremental value added to the identity value of the previous row.

  The default value of seed and increment is 1 i.e., (1,1). It means that the first row, which was loaded into the table, will have the value of one, the second row will have the value of 2 and so on.

- The following statement creates a new table using the IDENTITY property for the personal identification number column:

```
CREATE TABLE hr.person (
   person_id INT IDENTITY(1,1) PRIMARY KEY,
   first_name VARCHAR(50) NOT NULL,
   last_name VARCHAR(50) NOT NULL,
   gender CHAR(1) NOT NULL
);
```

INSERT INTO T-SQL Statement in SQL Server

- The INSERT INTO T-SQL statement is used mainly to add one or more rows to the target table or view in SQL Server. This can be done by providing constant values in the INSERT INTO statement or provide the source table or view from which we will copy the rows.
- Syntax

  INSERT INTO table (column1, column2, …)
  VALUES (expression1, expression2, …);

  And the INSERT INTO statement syntax that is used to insert multiple rows from a source database table is like:

  INSERT INTO table (column1, column2, …)
  SELECT expression1, expression2, …
  FROM source_tables
  [WHERE conditions];

Update query

- An UPDATE query is used to change an existing row or rows in the database. UPDATE queries can change all tables' rows, or we can limit the update statement affects for certain rows with the help of the WHERE clause. Mostly, we use constant values to change the data, such as the following structures.

  UPDATE table

  SET col1 = constant_value1 , col2 = constant_value2 , colN = constant_valueN

- Example

```
update employees
set full_name = first_name + ' ' + last_name
```

Delete Query

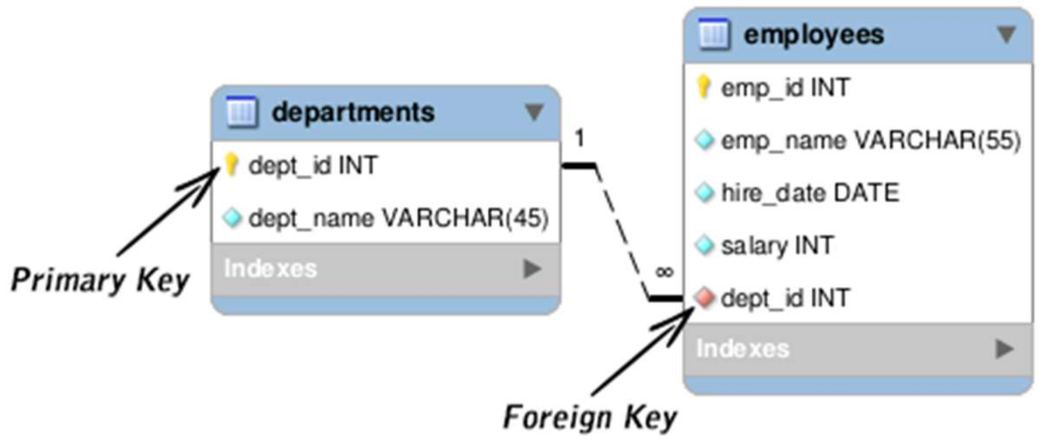
- The SQL Server (Transact-SQL) DELETE statement is used to delete a single record or multiple records from a table in SQL Server.
- Syntax

```
DELETE FROM table
[WHERE conditions];
```

- Example

```
DELETE FROM employees
WHERE first_name = 'Sarah';
```

Primary key (PK) and foreign key (FK)



The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

SQL Server joins

- In a relational database, data is distributed in multiple logical tables. To get a complete meaningful set of data, you need to query data from these tables using joins.
- SQL Server supports many kinds of joins, including
  - inner join,
  - left join,
  - right join,
  - full outer join,
  - cross join.
- Each join type specifies how SQL Server uses data from one table to select rows in another table.

Data creation for joins

```
CREATE SCHEMA hr;
GO

CREATE TABLE hr.candidates(
    id INT PRIMARY KEY IDENTITY,
    fullname VARCHAR(100) NOT NULL
);

CREATE TABLE hr.employees(
    id INT PRIMARY KEY IDENTITY,
    fullname VARCHAR(100) NOT NULL
);
```

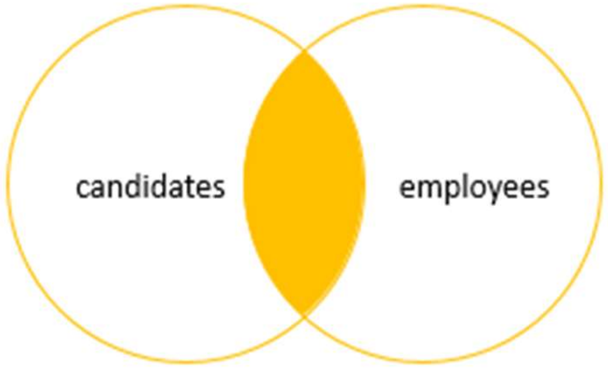
```
INSERT INTO
    hr.candidates(fullname)
VALUES
    ('John Doe'),
    ('Lily Bush'),
    ('Peter Drucker'),
    ('Jane Doe');


INSERT INTO
    hr.employees(fullname)
VALUES
    ('John Doe'),
    ('Jane Doe'),
    ('Michael Scott'),
    ('Jack Sparrow');
```

SQL Server Inner Join

- Inner join produces a data set that includes rows from the left table, matching rows from the right table.
- The following example uses the inner join clause to get the rows from the candidates table that has the corresponding rows with the same values in the fullname column of the employees table:

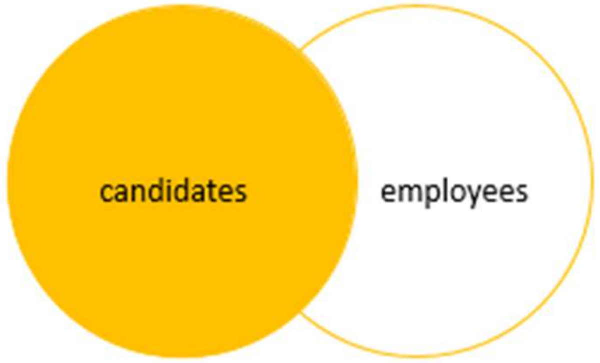
```
SELECT
   c.id candidate_id,
   c.fullname candidate_name,
   e.id employee_id,
   e.fullname employee_name
FROM
   hr.candidates c
   INNER JOIN hr.employees e
      ON e.fullname = c.fullname;
```



SQL Server Left Join

- Left join selects data starting from the left table and matching rows in the right table. The left join returns all rows from the left table and the matching rows from the right table. If a row in the left table does not have a matching row in the right table, the columns of the right table will have nulls.
- The left join is also known as the left outer join. The outer keyword is optional.
- The following statement joins the candidates table with the employees table using left join:

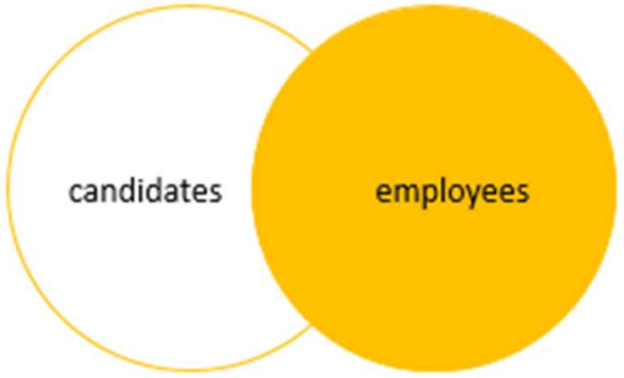
```
SELECT
        c.id candidate_id,
        c.fullname candidate_name,
        e.id employee_id,
        e.fullname employee_name
FROM
        hr.candidates c
        LEFT JOIN hr.employees e
                ON e.fullname = c.fullname;
```



SQL Server Right Join

- The right join or right outer join selects data starting from the right table. It is a reversed version of the left join.
- The right join returns a result set that contains all rows from the right table and the matching rows in the left table. If a row in the right table does not have a matching row in the left table, all columns in the left table will contain nulls.
- The following example uses the right join to query rows from candidates and employees tables:

```
SELECT
   c.id candidate_id,
   c.fullname candidate_name,
   e.id employee_id,
   e.fullname employee_name
FROM
   hr.candidates c
   RIGHT JOIN hr.employees e
      ON e.fullname = c.fullname;
```



Introduction to SQL Server GROUP BY clause

- The GROUP BY clause allows you to arrange the rows of a query in groups. The groups are determined by the columns that you specify in the GROUP BY clause.
- The following illustrates the GROUP BY clause syntax:
- Syntax

```
SELECT
   select_list
FROM
   table_name
GROUP BY
   column_name1,
   column_name2 ,...;
```

- If you want to reference a column or expression that is not listed in the GROUP BY clause, you must use that column as the input of an aggregate function.

```
SELECT
job.job_title,
count(emp.employee_id) as 'Number of employees'
FROM
[employees] emp
inner join
jobs job
on
emp.job_id = job.job_id
group by
job.job_title
```

Introduction to SQL Server HAVING clause

- The HAVING clause is often used with the GROUP BY clause to filter groups based on a specified list of conditions.
- The following illustrates the HAVING clause syntax:

```
SELECT
   select_list
FROM
   table_name
GROUP BY
   group_list
HAVING
   conditions;
```

- In this syntax, the GROUP BY clause summarizes the rows into groups and the HAVING clause applies one or more conditions to these groups. Only groups that make the conditions evaluate to TRUE are included in the result

```
SELECT
    sub.Name,
    avg(ListPrice) as 'Average List Price'
FROM
    production.product prod
INNER JOIN
production.ProductSubcategory sub
ON
sub.ProductSubcategoryID =
prod.ProductSubcategoryID

GROUP BY
    sub.Name
having
avg(ListPrice) > 100
ORDER BY
    sub.Name;
```