Exception Handling

Exception handling in SQL Server is the process of dealing with errors or exceptions that occur during the execution of SQL code.

- ERROR_LINE()
- ERROR_MESSAGE()
- ERROR_PROCEDURE()
- ERROR_NUMBER()
- ERROR_SERVERITY()
- ERROR_STATE()

```
BEGIN TRY
 Select 100/0 as Result;
   END TRY
BEGIN CATCH
   PRINT ERROR_MESSAGE();
END CATCH;
```

Transaction

Transactions

Commit | Save Poi... | Roll Back

Commit :

It is used to permanently save the changes in the database during transaction.

```
Begin transaction
insert into table_name values (,...)
Commit
```

Rollback:

It is used to undo the changes made during the transaction.

```
Begin transaction
Update table_name set value=100
Rollback
```

Savepoint:

Save point is a point within a transaction and can be rolled back later.

```
Begin transaction
Delete from table_name where=condition
Savepoint savepoint_name

Rollback
```

```
Select dbo.functioname(@parameters)
```

Indexes

An index in a database is a data structure that enhances the speed of data retrieval operations on a database table by providing a quick and efficient way to locate and access rows based on the values in one or more columns

Rollback:

It is used to undo the changes made during the transaction.

Syntax

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Example:

```
CREATE INDEX idx_employee_salary
ON employees (salary);
```

Single column index

```
CREATE INDEX idx_employee_salary
ON employees (salary);
```

Composite index(multiple columns)

```
CREATE INDEX idx_employee_department
ON employees (department_id, salary);
```

Unique index

```
CREATE UNIQUE INDEX idx_employee_id
ON employees (employee_id);
```

Clustered index

```
CREATE CLUSTERED INDEX
idx_clustered_example
ON table_name (column1, column2,
...);
```

Nonclustered index

```
CREATE NONCLUSTERED INDEX
idx_nonclustered_employees_salary
ON employees (salary);
```

Dropping the index (delete)

```
DROP INDEX index_name
ON table_name;
```

Pivot & Un-pivot

PIVOT rotates rows into columns by aggregating values.

Basic PIVOT syntax

```
SELECT columns, pivoted_columns
FROM table
PIVOT
(
 aggregate_func(value_column)
 FOR pivot_column IN (values)
) AS pivot_table;
```

Name	Product	Amount
John	Apple	100
John	Orange	200

Pivot query:

```
SELECT Name, Apple, Orange
FROM table
PIVOT
(
 SUM(Amount)
 FOR Product IN (Apple, Orange)
) AS pivot_table;
```

Output:

Name	Apple	Orange
John	100	200

UNPIVOT rotates rows into columns by aggregating values.

Basic UNPIVOT syntax:

```
SELECT columns,
unpivoted_column,
value_column
FROM table
UNPIVOT
(
 value_column FOR
unpivoted_column IN
(original_columns)
) AS unpivot_table;
```

Example:

Input table:

Name	Apple	Orange
John	100	200

Unpivot query:

```
SELECT Name, Product, Amount
FROM table
UNPIVOT
(
 Amount FOR Product IN (Apple,
Orange)
) AS unpivot_table;
```

Output:

Name	Product	Amount
John	Apple	100
John	Orange	200

IntelliPaat

SQL Cheat Sheet