Database Fundamentals

Day 3

***Introduction to SQL & RDBMS :**

- عشان أقدر أتعامل مع الـ Database أنا محتاج أنزل Software Tool تساعدنى اللى هيا (RDBMS) إختصار لـ (Relational Database Management System) واللغة اللى بتتعامل بيها لعمل الـ Queries هيا الـ (SQL) (Structured Query Language) وأحد الـ RDBMS اللى هنشتغل عليها هيا (SQL Server)

- فى البداية هيتم تنزيل الـ SQL Server Developer Edition من موقع Microsoft الرسمى

- نعمل Install وبكدا بقا عندك Service فى الـ Background عندك هتلاقيها إسمها MSSQLSERVER ونتأكد إنها Running

- هتنزل الـ Environment اللى هتتعامل مع الـ Service دى وهى الـ SQL Server Management Studio ودى اللى هكتب عليها الـ Queries

- أول ما هتفتح هتحتاج تـ Connect على الـ Service اللى فى الـ Background

- لو الـ Service على نفس الجهاز بيبقى أولها (.) أما لو على جهاز تانى بنكتب الـ IP الخاص بالجهاز

ANSI SQL

- الـ ANSI SQL هو النسخة الأساسية والبدائية لـ SQL وهو إختصار لـ (American National Standard Institute SQL) هو عبارة عن Library بتحتوى على الـ Syntax of Queries of DB

- Microsoft عدلت على الـ ANSI SQL وسمته Transact SQL

- Oracle عدلت على الـ ANSI SQL وسمته PL SQL

- IBM عدلت على الـ ANSI SQL وسمته IBM-PL-SQL

- Open Source عدلوا على الـ ANSI SQL وسموه MySQL

Microsoft ANSI SQL

(Transact SQL)

- الـ Transact SQL عبارة عن 5 Categories of Queries وهم :

1- DDL Queries (Data Definition Language)

2- DML Queries (Data Manipulation Language)

3- DCL Queries (Data Control Language)

4 – DQL Queries (Date Query Language)

5- TCL Queries (Transactional Control Language)

*** DDL Queries :**

- هيا عبارة عن Queries بتتعامل على أساس تـ Create and Modify الـ Structure of Data وتـ Define Database Objects (Table , View , Index , ….etc) وبالتالى بتتعامل على مستوى الـ Metadata وليس الـ value of data

- أمثلة على الـ DDL :

Create \ Alter \ Drop \ Truncate \ Rename \ Select Into

*** CREATE DDL Command :**

- بيستخدم لـ Creation of Database Objects (Table , View , Index , Triggers)

- Syntax to Create a new Database :

```
CREATE Database Database_Name ;
```

- Syntax to Create a new Table :

```
CREATE TABLE table_name
(
column_Name1 data_type ( size of the column ) ,
column_Name2 data_type ( size of the column) ,
column_Name3 data_type ( size of the column) ,
column_NameN data_type ( size of the column )  ) ;
```

- Syntax to Create a new Index :

```
CREATE INDEX Name_of_Index ON Name_of_Table (column_name_1 , column_name_2 , ..............,,
column_name_N) ;
```

ـ فكرة الـ Index هو عبارة عن Pointer بيشاور على الـ Column\s وفايته إنه بيـ retrieve data أسرع (من غير وجوده) وبالتالى سهلت الـ Search

ـ بس العيب إنه لو جيت أعمل Update لـ Table هياخد وقت أطول (من غير وجوده برضو) ودا لإن الـ Index بتـ Update برضو

ـ فى العادى مسموح بتكرار الـ Indexes بس لو مش عاوز أكررهم بكتب UNIQUE INDEX مكان الـ INDEX فى الـ Syntax

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

- Syntax to Create a new Trigger :

```
CREATE TRIGGER Trigger_Name
[ BEFORE | AFTER | INSTEAD OF]  [ Insert | Update | Delete]
ON [Table_Name]
[ FOR EACH ROW | FOR EACH COLUMN ]
AS
Set of SQL Statement
```

ـ فكرة الـ Triggers نفس فكرة الـ Interruption (اللى هو لما يحصل Event معين أنفذ مجموعة من الـ Commands)

ـ الـ Trigger هو Special Stored Procedure (ليه هو Special عشان على عكس الـ Normal Stored Procedures بيتم تلقائياً بدون ما تعمل Invoke)

ـ ممكن يتواجد فى الـ DDL و الـ DML والـ LOGON

*** DROP DDL Command :**

ـ هو عبارة عن Command بيـ Delete \ Remove الـ Database Objects

ـ ممكن تـ Delete table \ Index \ View

- Syntax to remove a Database :

```
DROP DATABASE Database_Name;
```

- Syntax to remove a Table :

```
DROP TABLE Table_Name;
```

- Syntax to remove an Index :

```
DROP INDEX Index_Name;
```

*** ALTER DDL Command :**

- هو Command بتعدل فى الـ Database Structure و الـ Database objects Schema من خلاله

- ممكن تضيف معاه الـ add \ Drop Constraints

- Syntax to Add a new field in table :

```
ALTER TABLE name_of_table ADD column_name column_definition;
```

- Syntax to remove a column from table :

```
ALTER TABLE name_of_table DROP Column_Name_1 , column_Name_2 , ....., column_Name_N;
```

- Syntax to Modify a column of table :

```
1. ALTER TABLE table_name MODIFY ( column_name column_datatype(size));
```

*** TRUNCATE DDL Command :**

- هو Command بيـ Remove all records فى الـ Table وكمان الـ Storing Space للـ Table Records

- Syntax of TRUNCATE Command :

```
TRUNCATE TABLE Table_Name;
```

* RENAME DDL Command:

- هو Command بيستخدم لـ Table Name Change

- Syntax of RENAME Command :

```
RENAME TABLE Old_Table_Name TO New_Table_Name;
```

---

* DML Queries :

- فى الـ DML Queries تقدر تـ Add new Data و تـ Update \ Delete Existing Data

- بتتعامل مع الـ Data Values

* INSERT DML Command :

- يستخدم لإضافة New Data Record

- Syntax of INSERT Command :

```
INSERT INTO TABLE_NAME ( column_Name1 , column_Name2 , .... column_NameN )
 VALUES (value_1, value_2, .... value_N ) ;
```

---

* UPDATE DML Command :

- يستخدم لتعديل An Existing Record

- Syntax of UPDATE Command:

```
UPDATE Table_name
SET [column_name1= value_1, ....., column_nameN = value_N]
WHERE CONDITION;
```

---

* DELETE DML Command :

- بيستخدم لـ Delete Record\s من الـ Database table

- Syntax of DELETE Command :

```
DELETE FROM Table_Name WHERE condition;
```

*** DQL Queries :**

- عبارة عن Command بيـ Fetch Data From Table

- هو SELECT Command

- Syntax of SELECT DQL Command :

```
SELECT Column_Name_1, Column_Name_2, ....., Column_Name_N FROM Table_Name;

SELECT * FROM table_name;
```

- ممكن تضيف كذا Clause مع الـ Select Statement ودا بيحدد تفاصيل الـ Data المطلوبة أكتر لكن الـ Functionality واحدة

- أمثلة على الـ Clauses :

- Syntax of SELECT Statement with WHERE clause :

```
SELECT * FROM Name_of_Table WHERE [condition];
```

فى أمثلة كتيرة على الـ Clauses زى :

- SELECT UNIQUE

- SELECT DISTINCT

- SELECT COUNT

- SELECT TOP

- SELECT FIRST

- SELECT LAST

- SELECT RANDOM

- SELECT IN

-SELECT MULTIPLE

- SELECT DATE

- SELECT SUM

- SELECT NULL

*** JOINS :**

- فكرة الـ Joins هى إن أعمل Query بيتعامل مع كذا Table فى نفس الوقت

*** Microsoft JOINS Types :**

1) Cross JOIN (Cartesian Product)

2) Inner JOIN (Equi JOIN)

3) Outer JOIN (Left – Right – Full)

4) Self JOIN (Unary Relationship)

---

1) Cross JOIN (Cartesian Product) :

- طريقة لعمل الـ Join بين Two or more Tables وفيها بيتم عمل Cartesian Product (ضرب كل Row من الـ Table فى كل الـ Rows فى الـ Other Table)

- طبعاً الناتج مش Applicable فى الـ Real DB

- Syntax of Cross Join :

```
SELECT Col1, Col2
FROM table1 , table 2 ;
```

```
SELECT Col1 , Col2
FROM table1 CROSS JOIN  table2 ;
```

- ملحوظة : أنا حر أختار أى عدد من الـ Columns من Different Tables بس لازم أكتب الـ Tables بعد الـ FROM

---

2) Inner JOIN (Equi Join) :

- فى الـ Inner Join أنا بحصل على الـ Actual Match Rows اللى بناء على شرط أنا بشترطه

Syntax of Inner Join :

```
SELECT Col1 , Col2
FROM table1 , table2
WHERE Pk = Fk ;
```

```
SELECT Col1 , Col2
FROM table1 INNER JOIN table2
ON Pk = Fk ;
```

- الناتج من الـ Queries دى هو الـ Columns اللى بتـ Match الشرط بتاع PK = FK

- الـ Rows اللى مش بتـ Match مش هتظهر

- لو حابب أظهر الـ Unmatched Rows دى بلجأ لـ Outer Join

3) Outer Join :

- طريقة لعمل الـ Join بين الـ Tables وفيها بيتم إختيار الـ Matched \ Unmatched Rows

- الـ Outer Join عبارة عن 3 Types وهم :

a) Left Outer Join :

- ودا بياخد الـ Matched \ Unmatched Rows من الـ Left Table اللى هو (Table 1)

- Syntax of Left Outer Join :

```
SELECT Col1 , Col2
FROM table1 LEFT OUTER JOIN table2
ON Pk = Fk ;
```

---

b) Right Outer Join :

- ودا بياخد الـ Matched \ Unmatched Rows من الـ Righ Table اللى هو (Table 2)

- Syntax of Right Outer Join :

```
SELECT Col1 , Col2
FROM table1 RIGHT OUTER JOIN table2
ON Pk = Fk ;
```

---

c) Full Outer Join :

- ودا بياخد الـ Matched \ Unmatched Rows من الـ two Tables

- Syntax of Full Outer Join :

```
SELECT Col1 , Col2
FROM table1 FULL OUTER JOIN table2
ON Pk = Fk ;
```

- (INNER) JOIN : Returns records that have matching values in both tables
- LEFT (OUTER) JOIN : Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN : Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN : Returns all records when there is a match in either left or right table

4) Self Join (Unary Relationship) :

- بيجى فى العلاقة الـ Unary أو الـ Self

- بستخدم Alias Names لـ Columns و الـ Table فى الـ Syntax

- الـ Table الأساسى الموجود معايا فى الـ Database بيبقى الـ Child والـ Table اللى بيتـ Create فى الـ Memory دا الـ parent

- الـ PK بيبقى خاص بالـ Parent Table

- الـ FK بيبقى خاص بالـ Child Table

- Syntax of Self Join :

```
SELECT X.ColumnName , Y.ColumnName
FROM TableName X ,  TableName Y
Where  Y.PK = X.FK ;
```

- الـ Self ممكن يخش معاها الـ Inner و الـ Outer عادى ، هيا الفكرة كلها تكمن فى أنك بتتعامل مع نفس الـ Table بس فى الـ Relation

---

Joins Multiple Tables :

- الـ Joins ممكن تكون بين Two or More Tables

- بيكون تحت شروط (عدد الشروط بتكون أقل من عدد الـ Tables بـ 1)

- نفترض إن عاوز أعمل Join لـ 2Table بس هما مش بينهم علاقة Direct ، الحل إن أعمل table as a Bridge يقدر يحقق الـ join

- فى كل 2 Tables أنا بحدد الـ PK والـ FK من الـ Relation فى الـ Diagram

- فأحدد الشرط بين الجدول الأول والـ Bridge و بعدها شرط الجدول التانى بالـ Bridge

- فيه طريقتين لكتابة الـ Multiples Joins:

1) الطريقة العادية أستخدم الـ , و الـ Where (ANSI SQL)

```
SELECT Column1_name ,  Column2_name , …….. etc
FROM Table1 , Table 2 , Table3
Where  PK1 = FK1  ( in relation between T1 &T2)  and
        PK2 = FK2 (in relation between T2 & T3) ;
```

2) أستخدم طريقة Microsoft (Transact SQL)

```
SELECT Column1_name ,  Column2_name , …….. etc
FROM Table_1 INNER JOIN Table_2
ON PK1 = FK1 ( in relation between T1 &T2)
INNER JOIN Table_3
ON PK2 = FK2 (in relation between T2 & T3) ;
```

- فى الطريقة دى أنا بعملها Step by Step بخلص أول join وبعدها الناتج أعمله Join مع اللى بعده وهكذا .....

Joins with DML :

- أنا ممكن أستخدم الـ joins مش بس مع الـ Select (مش بس أعرض Data)

- ممكن أستخدمها فى (Update \ Delete \ Insert)

- الفكرة فى الـ Join إنه من خلال From ومعاها كذا Table وتحت شرط فى الـ Where بيحددلى داتا معينة أنا ممكن أعملها Select أو أعمل Update فيها أو أعمل Delete ليها وهكذا

***Some of Guides in Joins :**

- لازم متنساش شروط الـ Joins عشان من غير شرط هيكون عبارة عن Cross Join

- شروط الـ Joins بحدد الـ PK والـ FK بتوع الـ Relationship بين أى Two Tables من خلال الـ Diagram (لو بتعمل Restore لـ Database ممكن الـ Diagramميظهرش معاك فتخلى نفسك الـ Owner of Database عشان تقدر تشوف)

- ممكن أضيف شروط مع شرط الـ PK = FK (ربط الـ Tables) (يعنى شرط ربط الـ Tableأساسى وممكن أضيف شروط أنا بـ and)

ـ قبل كدا كنت لما بحتاج إنى معرضش قيمة Column بـ NULL كنت بستخدم مع Where clause إما is null أو Is not null (Null is not A value عشان أستخدم الـ Equi Operator معاها)

ـ لو عاوز أعرض الـ Column بس أعمل Replace لقيمة الـ NULL بقيمة معينة ممكن تكون قيمة Column تانى أو قيمة كـ Default value بستخدم is null Function

Is null (column1_name , column2_name or Value)

ـلو عاوز أزود عدد الـ Columns بستخدم Coalesce Function ودى عبارة عن إنها بتـ Check لو كان أول Column بـ NULL أعرض تانى Column ولو التانى بـ NULL إستخدم التالت ..وهكذا وممكن أحط فى الأخر Default Value

Coalesce (column1_name , column2_name , …., columnN_name , Value)

ـ بستخدم الـ Convert Function لما يكون عاوز أعمل Concatenation بين Columns من أنواع Data types مختلفة فبوحد الـ Data type

Convert (data type (size) , column_name)

ـ فى حالة إن الـ Column فيه Null (أى قيمة هتتحط جنب الـ Null هيبقى بـ Null) فممكن أستخدم الـ Is null Function مع الـ Convert

Convert (data type (size) , is null (column_name , Value))

فى حالة إستخدام الـ Is null مع الـ Convert ممكن أستبدلهم بـ Concat Function ودى بتعمل نفس الـ Functionality ولكن خليت الـ Query مفيهوش Functions كتير وبالتالى أسرع

Concat (column1_name , column2_name)

Concat_ws ('Seperator' , column1_name , column2_name)

الفرق بين الـ = والـ like :

ـ الـ = : القيمة اللى عاوزها بتساوى القيمة اللى بعد الـ =

ـ الـ like : هى Keyword بستخدمها لما أكون عاوز أبحث عن Pattern معين (قيمة أنا مش فاكرها بالظبط ولكن فاكر جزء منها أو شكل الـ Pattern بتاعها) مع العلم ان لازم أكون على علم بإن :

Underscore (_) : one character

Percentage (%) : zero or more characters

Some patterns :

(a%) : أسماء بتبدأ بحرف الـ a وبعده zero or more char

(%a) : أسماء بتنتهى بحرف الـ a وقبله zero or more char

(% a %) : أسماء بتحتوى على حرف الـ a وقبله وبعده zero or more char

(_ a %) : أسماء تانى حرف فيها a وقبله one char وبعده zero or more char

(% a _) : أسماء الحرف القبل الأخير a و قبله zero or more char وبعده one char

(a % h) : أسماء أولها حرف a وأخرها حرف h وبينهم zero or more char

(ahm %) : أسماء بدايتها ahm و بعدها zero or more char

([ahm] %) : أسماء بدايتها (a أو h أو m) وبعده zero or more char

([^ahm] %) : أسماء بدايتها مش (a أو h أو m) وبعده zero or more char

([a – h] %) : أسماء بدايتها من حرف الـ a حتى h وبعده zero or more char

([^ a – h] %) : أسماء بدايتها مش من حرف الـ a حتى h وبعده zero or more char

([%] %) : أسماء أخرها حرف الـ % وقبله zero or more char

([_] %) : أسماء بتحتوى على الـ _ وقبلها وبعدها zero or more char

([_] % [_]) : أسماء بدايتها _ ونهايتها _ وبينهم zero or more char

*** Order by Keyword :**

- بتستخدم فى حالة الترتيب فى عرض الـ Data

- ممكن أرتب بإستخدام أى Column من الـ Table (حتى لو مش ضمن الـ Columns اللى هتتعرض)

```
SELECT col_1 , col_2, col_3
FROM table
Order by by col_1
Order by by col_4
```

- ممكن أكتب رقم بعد الـ Order by ودا بيبقى معناه رقم الـ column فى الـ Query وهستخدم الـ Column دا فى الـ Ordering

```
SELECT col_1 , col_2, col_3
FROM table
Order by 1 (refers to col_1)
Order by 2 (refers to col_2)
Order by 3 (refers to col_3)
```

- الـ Multiple Ordering ممكنة بإستخدام أكتر من Column

```
ELECT col_1 , col_2, col_3
FROM table
Order by col_3 , col_4
```

فى الحالة دى هيرتب الـ Data بـ Col_3 وبعدها المتشابه فى الـ Col_3 هيرتبهم بإستخدام Col_4

- ممكن أستخدم الـ Keywords asc \ desc فى تحديد الـ Ordering Type كـ Ascending أو Descending

Normalization

- الـ Database Design بيتم عن طريق :

إما ERD Creation ودا بيكون لـ System From Scratch

أو إنه نعمل تعديلات (Modifications) على System موجود قبل كدا ولكن فيه بعض الـ Features عاوز أضيفها أو تعديلات لـ Schema

- هدف الـ ERD أو الـ Normalization هو إنى أوصل لـ Mapping سليم

- الـ Normalization : هو طريقة لإعادة هيكلة الـ Tables عشان أقدر أوصل لـ Database سليمة

- بعد ما تعمل الـ Schema عن طريق الـ ERD طبق عليها قواعد الـ Normalization برضو عشان تتأكد إن الـ Mapping سليم

---

* هدف الـ Normalization :

- حل مشاكل الـ Duplication و الـ Inconsistency و الـ DML Anomalies

---

- الـ Normalization معتمد على فكرة الـ Functional Dependency

- الـ Functional Dependency هى relation بين الـ attributes والـ PK

- بتتمثل الـ Functional Dependency فى مدى إعتمادية الـ Column على الـ Primary Key

- Functional Dependency Representation :

$A \rightarrow B$

- الطرف اللى ع الشمال (A) بيسمى الـ Determinant و اللى على اليمين بيسمى الـ Dependent

- العلاقة بتتقرى إما

B is Functionally Dependent on A

أو

A is a determinant of B

- معناها إن مع أى Unique value of A بيقى فيه Value of B ليها

- مثال مثلا إن لو أنا معايا الـ SSN أقدر أجيب الـ Name

- الـ Table بيتقال عليه سليم لو كان

$A \rightarrow B , C , D$

$PK \rightarrow Columns$

- لو عرفت الـ Primary Key أقدر أجيب أى Column تانى (مش العكس)

$H , M \rightarrow Z$

- ممكن يكون الـ PK عبارة عن Composite Key

*** Functional Dependency Types :**

1) Full Functional Dependency :

- بيكون كل الـ Columns معتمدة على الـ PK

2) Partial Functional Dependency :

- فى حالة إن فيه Column بقدر أجيبه من جزء من الـ CK

3) Transitive Functional Dependency :

- فى حالة إن فيه Column بقدر أجيب قيمته من Non PK(من Column غير الـ Primary Key)

Normalization Rules :

- موجود 6 أنواع من الـ Normal Form (الـ Form اللى بيبقى عليها الـ Table بعد تطبيق الـ Rule الخاص بيها)

- فكرة الـ Normalization إنه بيـ Divide الـ Table اللى بيـ Violate الـ rule فى كل Normal Form إلى Tables بينهم Relations

1) First Normal Form (1NF)

2) Second Normal Form (2NF)

3) Third Normal Form (3NF)

4) Boyce Codd's Normal Form (BCNF)

5) Fourth Normal Form (4NF)

6) Fifth Normal Form (5NF)

- فى الكورس هنتعرف على أول 3 أنواع

- لما بستلم الـ Table بعتبر إنه فى الـ Zero Normal Form وببدأ أطبق Rule كل Normal Form لحد م أوصل لـ 5NF

Normal Forms Rule :

1) 1NF :

- بشوف الـ Multivalued columns وبعمل جدول ليها مع الـ PK

- الـ New Table :

الـ PK هو عبارة عن Composite Key (الـ PK والـ Multivalued Column)

الـ FK عبارة عن (الـ PK) فى الجدول الأساسى

- الـ Rule دى بتمنع إن يكون فى الـ attribute الواحد Multiple Values

2) 2NF :

- بركز ع العلاقة الـ Partial Functional Dependency

- بشوف الـ Columns اللى بتعتمد على جزء من الـ CK وباخد الجزء والـ Columns فى جدول تانى

الـ PK فى الجدول الجديد (بيكون جزء الـ CK)

والـ FK فى الجدول الجديد (بيكون نفس جزء الـ CK) بيشاور على الـ PK فى الجدول الأساسى

3) 3NF :

- بركز على العلاقة الـ Transitive Functional Dependency

- بشوف لو فيه Two Non-Key Columns لو فيه Dependency Relation بينهم

بعمل جدول فيه الـ Columns ويكون

الـ PK عبارة عن الـ Determinant

والـ FK عبارة عن الـ Determinant فى الجدول الأساسى

Steps in normalization