DDL Commands and Constraints, Indexes

**********What	**********Why	**********How

**********	**********DDL**********	**********
Create Table	To store data for objects, following a certain format. Allows retrieval, and updating with other SQL commands.	CREATE TABLE Pets (petId INT PRIMARY KEY IDENTITY(1,1), petName VARCHAR (70) NOT NULL, petColor VARCHAR (70) NOT NULL, petBreed VARCHAR (70) NOT NULL, petAge INT);
Alter Table	To add and/or modify existing table columns.	ALTER TABLE Pets ADD weight INT NOT NULL; ALTER TABLE Pets ALTER COLUMN PetName VARCHAR(55);
Drop Table / Database	Removes an existing table or database.	Drop table Pets; Drop database PetsCo;
**********	******CONSTRAINTS******	**********
NOT NULL	Prevents creation of a table row without giving a value for the column.	USE PetsCo go CREATE TABLE catList (catNumber INT NOT NULL, catName VARCHAR(90) NULL)
UNIQUE	No row's column will have an equal value in the same column of any other row.	CREATE TABLE dogList (dogNumber INT UNIQUE, dogName VARCHAR(35) NULL)
PRIMARY KEY	Has to be unique and also as-well NOT NULL, and can contain one or more columns. This allows to specifically identify each of a table's rows apart from another.	USE dogBase GO CREATE TABLE theDogs (dogNumber INT PRIMARY KEY, dogSurname VARCHAR(25) NULL)
CHECK	Permits only certain values for a row.	CREATE TABLE reptiles (speciesID VARCHAR(50),

		reptilianWeight INT, CONSTRAINT check_reptilian_weight CHECK (reptilianWeight BETWEEN 1 and 10000));
FOREIGN KEY	The key in the referenced table must exist and be the same value as this foreign key.	CREATE TABLE petHome (petNumber int, petSurname VARCHAR(25), CONSTRAINT PK_petHome PRIMARY KEY NONCLUSTERED (petNumber), CONSTRAINT FK_PETHOME FOREIGN KEY (petNumber) REFERENCES pets(petid) ON DELETE CASCADE, qty_ordered int);
DEFAULT	Sets a value range that must be entered for a column's data.	Create Table PetsCoWild (petWeight int default(35), surName Nvarchar(50) Constraint surName_Default Default('Poodle Pooch'), petAge Int, maxPetAge Int Default(25))
**********	**********INDEXES*********	**********
UNIQUE	The index key contains non duplicate entries, the table column(s) it refers to also will be a collection of unique data as well. There may be 1 or more columns. Created automatically for table primary or unique keys.	`CREATE UNIQUE NONCLUSTERED INDEX IX_petIDIndex ON Pets (petID)`
CLUSTERED	The most prominent index SQL Server has . Tables w/o a	`CREATE CLUSTERED INDEX CLU_PetsIndex ON PetsCoWild (petweight);`

	clustered index are on the heap not in any particular order. Table data stored in B-tree leaf node, index key in a node, eliminates heap usage. The non-keys are stored in leaf-nodes making possible only one of this type index. Queries are faster. For inserts, updates and deletes it can slow things down	

NONCLUSTERED	Same as clustered except a pointer to the data is stored in the leaf node not the real-data. Could point to the clustered index area or, the heap! Multiple indexes allowed, on different columns, in WHERE, and any ORDER BY sort is not now needed. Overhead less than for CLUSTERED. Too many indexes slow it down, right amount speeds up queries.	`CREATE INDEX IndexPetsAUST ON PetsAUST(PetID);`
		
		`ALTER INDEX IndexPetco ON PetCo REBUILD WITH (ONLINE = ON);`
		
		`DROP INDEX IndexPetco on PetCo;`

FILTERED	Indexes a part of all the table rows, to help with queries that only need rows with certain values. So, if you select on a column from a 100k row table, and only 15k rows have a value in that column, it helps out ! Has a where clause.	`CREATE INDEX FIL_petsCoAUST ON PetsAUST (petID) WHERE petID IS NOT NULL`

SPATIAL	Index on spatial type data only! Geometry data type columns are geometric, points, lines and polygons. The geometry type has geographic objects . Must already have a clustered primary key.	`go`
		`CREATE TABLE PetsJAPAN`
		`(`
		`petNumber INT IDENTITY (1,1) PRIMARY KEY,`
		`petSurName Varchar(50) ,`
		`petPicture geometry`
		`)`
		`go`
		`Has column with a spatial data type and clustered prim key.`
		
		`CREATE SPATIAL INDEX PetsJapanSpatialIndex`
		`ON PetsJapan(petPicture)`
		`WITH ( BOUNDING_BOX = ( 0, 0, 500, 200 ) );`

XML Primary Secondary	XML BLOB, binary large objects!
	Makes a query faster by figuring on the tags, values and paths for instances of XML in that column and indexing them.
	The Primary clustered on the table, clustering table and xml node. Once created, it can create Secondary, like PATH, VALUE PROPERTY. PATH XML, is from path and node values of primary helps finding paths. PROPERTY XML has base table prim key and path, node value of the primary XML index.
		CREATE TABLE PetsEURO
		(petID INT IDENTITY (1,1) PRIMARY KEY,
		  surName Varchar(50) ,
		petAddress XML
		)
		
		CREATE PRIMARY XML INDEX PrimaryXML_PetsEURO_petAddress
		  ON PetsEURO (petAddress);
		GO
		
		Now, a PATH secondary index,
		
		CREATE XML INDEX SecondaryXML_PetsEURO_petAddress_Path
		  ON PetsEURO (petAddress)
		  USING XML INDEX PrimaryXML_PetsEURO_petAddress FOR PATH;
		GO
