## Uvod

SQL(structured query language) je jezik koji se koristi za slanje upita ka RDBMS(relational database management system) koji skladisti podatke po relacionom modelu. Relacioni model je semanticki model za prezentovanje podataka koji se bazira iz dve grane matematike – set theory(teorija skupova) i predicate logic.

T-SQL je dijalekt SQL-a.

### SQL

SQL je ANSI i ISO standard language za queryovanje I storovanje podataka RDBMS-a.

SQL se sastoji iz nekoliko kategorija izraza:  
- **DML** (data manipulation language) (select, update, insert… tipovi izraza za upit i manipulaciju podataka)  
- **DDL** (data definition language) (create, alter, drop tipovi izraza, za manip. definicija podataka)  
- **DCL** (data control language) (grant, revoke… tipovi izraza za manipulaciju dozvola)

Ova knjiga se fokusira na DML tipove izraza, znaci SELECT, INSERT, UPDATE, DELETE, TRUNCATE, and MERGE

Common misunderstanding is that TRUNCATE is a DDL statement, but in fact it is a DML statement

### Set Theory

*By a set we mean any collection M into a whole of definite, distinct objects “m” (which are called the “elements” of M) of our perception or of our thought.*

* *Georg Cantor*

//Mozes malo vise prouciti o skupovima kasnije ako te interesuje, ali osnovna skola

### Predicate logic

Koristi se za upravljanje podacima i izvrsavanje upita nad podacima. PREDICATE je u sustini izraz/svojstvo koje je tacno ili nije tacno(holds, or doesn’t hold)

Relacioni model se oslanja na predicate za zadrzanje logickog integriteta podataka i definisanje strukture istih. U sustini primer bi bio CONSTRAINT koji ce nad tabelom “employees” da postavi pravilo da plata mora da bude > 0 => salary > 0

### The relational model

Relacioni model je sematicki model za upravljanje podacima i manipulaciju koji se bazira na teoriji skupova i logici. Cilj ovog modela jeste omogucavanje konzistentne prezentacije podataka uz minimalnu ili bez imalo redundantnosti gde ne moras da se oslanjas na intuiciju jer jelte, bazira se na matematici.

Relacioni model ukljucuje koncepte poput “propositions, predicates, relations, tuples, attributes & more”.

### Propositions, predicates & relations

Da deo “relacioni” u izrazu “relacioni model” stoji tu na osnovu koncepta “relacija” je netacan. Zapravo potice iz matematickog izraza relacija. U teoriji skupova, relacija je representation(predstava) skupa. U relacionom modelu, relacija je skup (related) informacija. U SQL-u taj skup se naziva tabela(not exact counterpart, but close).

Key point -> relacija jeste jedna tabela.

Operacije nad relacijama na osnovu relacione algebre rezultuju relacijama

//Relational model razlikue relaciju od relacione promenljive but to keep things simple, necemo o tome sada. Koristicemo izraz “relacija” za oba.

Relacija se sastoji iz headinga i bodyja.

Heading jeste spisak atributa(kolona u SQL-u) gde svaka kolona ima svoj naziv i tip podatka koje ce da sadrzi.

Body jeste skup tuple-a(row u SQL-u) gde se svaki element identifikuje kroz key. To KIS, reci cemo da je tabela skup redova.

A diagram of a workflow

AI-generated content may be incorrect.

Keep in mind da elementi u skupu(tuples) i kolone/atributi mozda mogu izgledati kao da imaju redosled. Oni nemaju redosled(order) i ne mogu imati redosled, po definiciji skupova.

To isto vazi i za recorde unutar tabele.

Kada formalizujes neku pojavu u prirodu tako da je mozes prezentovati kroz skup – ti ces u sustini koristiti razne predicates. Za opisivanje kolone/atributa, npr ti ces koristiti TIP(type).

**TYPE** je najjednostavniji oblik predikata. Npr int ogranicava kolonu da vrednost bude u opsegu –2,147,483,648 do 2,147,483,647. Tip nije samo integer/date. To su jednostavni tipovi.

Tip moze biti i kompleksan gde bi kompleksan bio npr neka enumeracija vrednosti.

### Missing values

Jelte, postoji debata o tome da li bi predikat trebao podrzavati dvo-vrednostne ili trovrednosne vrednosti. Dvovrednosni predicate bi bio npr true/false.

Trovrednosni bi bio true/false/null tj nepostojanje vrednosti. Dvovrednosni sistem podrazumeva “the law of excluded middle” sto je matematicko pravilo koje kaze da “nema sredine”. Ili jesi, ili nisi. Sa Trovrednosnim sistemom ti kazes “unknown/ne znam” da je takodje vrednost.

Neki tvrde da se NULL/unknown i trovrednosni predicate logic na ubrajaju u relacioni model.

SQL supports three value predicate logic.  
Postoji “Codd” koji tvrdi da treba da postoji cetvorovrednosni predicate sistem u kom kada postoji nepoznata vrednost(null), ona se deli na “primenjivu” i “neprimenjivu”. Neprimenjiva bi bila kada employee nema mobilni telefon pa onda postojanje atributa mobilnog telefona nema primenu tj non-applicable.

Ukoliko employee ima mobilni telefon onda je “primenjiva” nepoznata vrednost jer jelte, ima telefon samo mi ne znamo koji je. SQL ne podrzava cetvorovrednosni predicate sistem.

### Constraints

Jedna od prednosti relacionog modela jeste mogucnost za definisanjem integriteta podataka kao deo modela. Constraints su pravila koja se jelte definisu u modelu, a koje sprovodi RDBMS. Jedan od najcescih jeste da li atribut dozvoljava ili ne dozvoljava null marker.

Constraints takodje se namecu kroz sam model – npr ako imas tabelu sa 3 kolone, onda neces imati 4 atributa :)

Za sprovodjenje referential integrity koriste se candidate keys i foreign keys. Na osnovu candidate key mozes uniquely identifikovati record u tabeli npr. Tipicno je da se jedan od candidate keys koristi kao primary key.   
U slucaju ljudi to ce biti JMBG npr, a mozes i “id”. Svi drugi candidate keys pored tvog primary key-a se nazivaju alternate keys.

Foreign keys are used to enforce referential integrity. Definise se nad jedan ili vise atributa u relaciji i referencira candidate key u toj relaciji ili nekoj drugoj relaciji.

### Normalization

Relacioni model poseduje pravila normalizacije, kao i u matematici. Naime, sluzi da se smanji redundantnost podataka.

Postoje x3 nominalne forme - 1NF, 2NF, 3NF

1st nominal form(1NF) kaze da svaki record(tuple) u tabeli(relaciji) mora biti unique i da svaka kolona(atribut) mora biti atomican.

2nd nominal form(2NF) ima dva pravila. Prvo kaze da podaci moraju da zadrze njihov prvu normalnu formu. Drugo je vise vezano za kljuceve(candidate & foreign).

Svaki atribut u relaciji sa candidate key, teba da bude funkcionalno zavisan od tog candidate key-a. To znaci da atribut ne moze da bude parcijalno zavisan od candidate key-a.

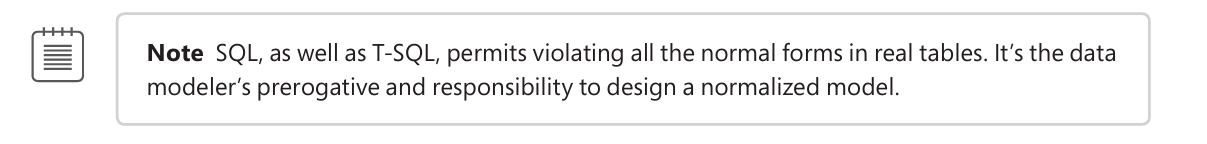
To put it more informally, if you need to obtain any nonkey attribute value, you need to provide the values of all attributes of a candidate key from the same tuple

A screenshot of a computer

AI-generated content may be incorrect.

TLDR ,ne mozes da imas dva primary key-a gde ces na osnovu jednog ili na osnovu drugog raditi pretrage.

3rd nominal form(3NF) takodje ima dva pravila. Prvo je da mora se ispostovati 2nd nominal form. Drugo je da svi non-key atributi moraju biti individualni i nezavisni jedni od drugih(must be dependent on candidate keys nontransitively.)



## Types of database workloads

Dva najcesca workloada su za OLTP(Online transaction processing) i DW(data warehousing).

Za DW to mozes sa AzureSQL/SqlServer koji koristi SMP(symmetric multiprocessing) architecture ILI za malo zahtevnije workloade Synapse Azure Analytics koji koristi MPP(Massive parallel processing)

A diagram of a diagram of a database

AI-generated content may be incorrect.

Here’s a quick description of what each acronym represents:

■ OLTP: online transactional processing   
■ DSA: data-staging area   
■ DW: data warehouse   
■ ETL / ELT: extract, transform, and load; or extract, load, and transform

### OLTP – Online Transaction Processing

Relacioni model jeste primarno “optimizovan” za OLTP workloads koji predstavljaju neki bazicni CRUD kada ti imas online transakcije koje na taj nacin rade sa podacima, znaci primarni cilj ti je data entry bez da imas neki reporting.

Ovde su tabele cesto i normalizovane, ali upravo zbog te normalizacije ovaj use case ne odgovara reportingu jer u OLTP mi cemo imati puno entiteta sa kompleksnim relationshipima.

Ako bi to pokusali na reporting scenariu, mi cemo imati mnogo kompleksnih queryja sa losim performansom.

### DW – Data Warehousing

Ovo je okruzenje koje je optimizovano za pribavljanje podataka i reporting purposes. Ukoliko opsluzuje celu organizaciju onda se to naziva data warehouse. Ako opsluzuje samo departman unutar organizacije ili jedan mali specifican context onda se naziva data mart.

U ovim slucajevima model podataka je optimizovan primarno za retrieval needs.

The model has intentional redundancy, fewer tables, and simpler relationships, ultimately resulting in simpler and more efficient queries than an OLTP environment.

Najjednostavniji data warehouse dizajn se naziva **star schema**. Ona ukljucuje nekoliko dimenzija schema i fact table(tabelu cinjenica). Svaka dimenzija predstavlja jednu temu na osnovu koje zelis da radis reportove – na primer sistem koji radi sa “orders & sales” verovatno ce zahtevati sledece dimenzije: customers, products, employees & time.

Svaka dimenzija je implementirana kao jedna tabela sa redundantnim podacima.

single ProductDim(product dimension) table instead of three normal ized tables: Products, ProductSubCategories, and ProductCategories

Ukoliko uzmes i normalizujes tabelu dimenzije(u vise tabela jelte), onda dobijas nesto sto se naziva **snowflake dimension**. Takva schema koja podrzava normalizovane dimenzije se naziva **snowflake schema**(znaci imas jedan ili vise snowflake, ne mora sve u snowflake).

**Fact table** tj tabela cinjenica sadrzi “facts and measures”(cinjenice i mere) poput kolicina, vrednost za svaku relevantnu kombinaciju kljuceva dimenzija.

Npr za svaku relevantnu kombinaciju customer, product, employee & day – tabela cinjenica ce imati red koji sadrzi quantity & value.

Note – podaci u dimenzijama su cesto “pre agregirani” za razliku od OLTP podataka koji su skladisteni cesto “po transakciji”.

### ETL/ELT

Proces koji povlaci podatke sa izvornih sistema(OLTP & drugi), manipulise im i ucitava ih u datat warehouse se naziva **Extract, transform & load ETL.**

Neki sistemi prvo povlace podatke sa izvornih sistema, ucitavaju ih u data warehouse i na kraju ih transformisu se nazivaju **Extract Load Transform – ELT**

**Microsoft provides an on-premises tool called Microsoft SQL Server Inte gration Services (SSIS) to handle ETL/ELT needs which comes with SQL Server license.**

Microsoft also provides a serverless cloud service for ETL/ELT solutions called **Azure Data Factory**

Cesto procesi koji vrse OLTP -> DW i koji sadrze neki ETL/ELT posao, imaju i DSA**(Data-Staging-Area)** koja moze biti u nekoj npr relacionoj bazi i sluzi kao data cleansing area. Zona gde ce se podaci “preciscavati”.

## SQL Server Architecture

### On-Prem & Cloud flavors of Sql Server.

Inicijalno je tu bio samo jedan RDBMS – Microsoft Sql Server koji je sluzio za on premises baze. Danas unutar svoje “data platforme”, MSFT nudi onprem, box solutions i service based cloud solutions. Mnogo vise :)

### On Premises

Ova nijansa RDBMS-a se naziva SQL Server koji je najtradicionalnija verzija sto msft nudi. Ovde customer vodi racuna o svemu – hardver, patchovi, disaster recovery, data retention i slicno.

Customer moze da instalira i vise instanci SQL servera na istom backendu i radi sta hoce u sustini. Max kontrola.

### Cloud

Cloud computing varijanta nudi cloud compute & cloud storage opcije “po zahtevu” sa deljenog “pool of resources”. MSFT RDBMS mogu biti ponudjene i sa public cloud i private cloud varijanti.

Za private cloud infrastrukturu se misli na cloud infrastrukturu koju jedna organizacija koristi putem tehnologije virtuelizacije. Cesto je hostovana lokalno.

Za public cloud infrastrukturu – ovde microsoft sve resava za tebe tako sto svi servisi public cloud infrastrukture su ti dostupni preko “network” tj mreze.

Ovde imamo x2 opcije, a to su IaaS i PaaS.

**IaaS** je kada hostujes SQL Server na VM. Ovde je na tebi da podesavas sve osim underlying hardware, jel tako. It’s essentially like maintaining your own SQL Server installation—one that happens to reside on Microsoft’s hardware.

**PaaS** je ukoliko koristimo microsoft managed sql server instance. Hardware, software installation and maintenance, high availability and disaster recovery, and patching are all responsibilities of Microsoft.   
Customer jedino o cemu treba da brine je index & query tuning.

### SQL Server Instances

U sustini ovo su instalacije SQL Servera na jednom on prem servisu. Svaka instalacija(proces) ne deli nista drugo izmedju sebe osim hardvera. Kompletno je nezavisna.

A diagram of a server

AI-generated content may be incorrect.

Kada imas vise instanci na istom serveru, samo jedna instanca moze biti default instanca. Sve ostale moraju biti named instances. O tome koja instanca je default, odlucuje se na kreiranju same instance i ne moze se kasnije promeniti.

Da bi se client povezao na instancu:  
**(default)** – **{pc\_ip/pc\_name} (Server1)**  
**(named\_instance)** – **{pc\_ip/pc\_name}\{instance\_name} (Server1\Inst2)**

### Databases

Baze(databases) su kao containeri koji sadrze razlicite objekte poput tabela, viewova, stored procedura i drugih objekata. Svaka instanca SQL servera moze sadrzati vise baza.

Setup proces takodje automatski postavi nekoliko sistemskih baza za interne mssql potrebe.

Ti naravno mozes postaviti svoje user databases koje ce sadrzati podatke koje ti zelis.

A diagram of several columns

AI-generated content may be incorrect.

Sistemske baze:

* **master** – Sadrzi “instance wide” metadata informacije, konfiguracije servera, informacije o svim bazama unutar instance i informacije inicijalizacije.
* **model** – Ova baza se koristi kao template za nove baze. Svaka nova baza koju kreiras incijalno je kreirana kao kopija model baze, pa ako zelis neke objekte poput user defined data types da budu dostupni u svim novokreiranim bazama ili da novokreirane baze budu unapred konfigurisane na neki nacin, to ovde mozes da podesis. Note, dodatne izmene u “model” bazi se nece odraziti na vec postojece baze
* **tempdb** – ovde SQL Server skladisti privremene podatke poput work tables, sort & hash table data kada treba da se persistuju, row versioning information i slicno. SQL Server ti dozvoljava da ovde kreiras i svoje tabele, ali **tempdb** se unistava i re-kreira kao kopija **model** baze svaki put kada restartujes instancu SQL servera.
* **msdb** – Ovu bazu koristi najcesce SQL Server Agent servis da skladisti svoje podatke. Njegov zadatak je automacija u koje spadaju “jobs, schedules, and alerts”. SQL Server Agent takodje vrsi replikacije. Pored toga, msdb sadrzi podatke o SQL Server features poput database mail, service broker, backups, and more.
* **resource** – resource baza je skrivena, read only db koja cuva definicije o svim sistemskim objektima. Kada queryujes system objekte u bazi, ispada kao da su oni u **sys schema** lokalne baze, ali u stvarnosti sve te definicije se nalaze u resource bazi.

Mozes kreirati 32.767 user baza unutar instance. User database skladisti objekte i podatke aplikacije.

Postavljanjem “collation” propertya na db level ce ti odluciti o default language support, case sensitivity & sort order za character data u bazi. Ako ne specificiras, uzimaju se default collation instance.

* Security concepts

Naime kako bi pristupio instanci moras imati pristup ti kao user. To je login.   
Potom kada zelis da pristupis nekoj bazi, db admin mora da namapira tvoj login na database usera.

Database user je entitet na kojeg se nadodaju permisije ka db objektima. SQL Server supportuje **contained databases** feature koji “breaks the connection” izmedju db usera i instance level logina. User(Windows/SQL Authenticated) je fully contained unutar odredjene baze i nije povezan u login na instance level.

To znaci da kada se povezujes na SQL Server kao user, moras da specificira bazu na koju se povezujes. Ne mozes kasnije switchovati baze.

Kada radis sa Azure SQL Managed instance, ti se brines jedino o logickom sloju. Ako radis sa SQL Serverom na VM ili on-prem, onda moras da radis i sa fizickim slojem baze. To znaci da radis i sa sledecim:

A diagram of a computer system

AI-generated content may be incorrect.

Baza se sastoji iz data files, transaction log files, opciono checkpoint files koji sadrze memory optimized data(In-Memory OLTP feature).

Svaka baza mora da ima barem jedan data file i barem jedan log file. Data files cuvaju object data, a log files informacije koje SQL Serveru trebaju da bi odrzao transakcije.

Pri kreiranju baze mozes definisati razlicita svojstva za data i log files ukljucujuci i file name, location, initial size, max size i autogrowth increment.

Iako SQL Server moze da upisuje u vise data files u paraleli, sa log fileovima moze samo jedan(one at a time) u sekvencijalnom pristupu.

**Datafiles** su organizovani u **filegroups** koje su u sustini logicke grupacije. Filegroup je target za kreiranje objekta poput table/index. Informacije o objektu ce biti rasporedjene kroz fileove koji pripadaju target file groupi.

Filegroups su tvoj nacin za kontrolisanje fizickih lokacija tvojih objekata. Database mora posedovati barem jednu filegroup koja se naziva **PRIMARY** i opciono moze imati druge filegroups.

Primarni data file(**.mdf** ekstenzija) se nalazi u primary filegroup i to je pirmarni data file za bazu i system catalog. Opciono mozes dodati secondary data files(**.ndf** ekstenzija) unutar PRIMARY.

User filegroups sadrze samo secondary data files. Mozes odabrati koja filegroup je marked kao default filegroup.

Kada object creation statement ne govori preciznije o tome gde se kreira, on ce se kreirati u default filegroup.

.mdf -> master data file

.ldf -> log data file

.ndf -> not master data file

SQL Server db engine u sebi ima takodje i memory-optimized engine “In-Memory OLT”. Uz taj feature mozes integrisati memory optimized objects poput memory optimized tables i natively compiled modules(procedures, functions & triggers) u bazu. Da bi to uradio moras da kreiras filegroup u bazi koji ces markovati kao containing memory optimized data i unutar njega bar jedan path u folder. Sql server storeuje checkpoint files u memory optimized data u tom folderu i koristi ih da recoveruje data svaki put kad je sql server restartovan.

### Schemas & Objects

Da je baza container objekata je malo simplified verzija. Baza sadrzi schemas. Schemas sadrze objekte. Schema je container(namespace) objekata poput tabela, views, stored procedures & drugo.

A diagram of a database

AI-generated content may be incorrect.

Permisije mozes kontrolisati na schema nivou. Npr useru mozes dozvoliti SELECT permisija za samo jednu schemu.

Pri svojim upitima ukoliko ne ukljucis schema name, SQL Server ce apply process za resolve schema name poput proveravanja da li objekt postoji u **user default schema**, a ako ne postoji onda proverava da li postoji u **dbo** schema.

Ukoliko isti objekat postoji u vise schema onda je bolje da specificiras schema pre nego pokusas da ga referujes jer mozda dobijes razlicite rezultate

## Creating tables and defining data integrity

Ovi primeri ovde se odnose na **dbo** schema koja se automatski kreira u svakoj bazi i postavlja za default schema za usere osim ukoliko nije drugacije naznaceno.

USE TSQLV6; --> sets current db context

DROP TABLE IF EXISTS dbo.Employees; --> mozes i da ni ne kreiras ako postoji vec

CREATE TABLE dbo.Employees --> **dbo.Employees** two part name(msft recommended)  
(   
empid INT NOT NULL,   
firstname VARCHAR(30) NOT NULL,   
lastname VARCHAR(30) NOT NULL,   
hiredate DATE NOT NULL,   
mgrid INT NULL,   
ssn VARCHAR(20) NOT NULL,   
salary MONEY NOT NULL   
);

Ako ommitujes schema name onda ide u default schema.  
Pri create table specificiras <attribute\_name> <data\_type> <nullability>

### Defining data integrity

Imamo procedural data integrity gde se integritet podataka cuva kroz stored procedure i trigere i declarative data integrity gde se integritet cuva kroz samu definiciju modela.

declarative data integrity: primary key, unique, foreign key, check, default constraints, nullability, data type.

Takve constraintove mozes u toku CREATE TABLE definisati, a i dodati kasnije kroz ALTER TABLE. Svi constraintovi osim default constrainta se mogu definisati kao composite constraints.

### Primary key constraints

PK Constraint forsira uniqueness of rows i nedozvoljava NULL u constraint atributima. Svaka kombinacija vrednosti atributa se znaci moze pojaviti samo jednom(unique).

Ako pokusas da dozvolis null pod PK, dobijes error.

ALTER TABLE dbo.Employees  
 ADD CONSTRAINT PK\_Employees  
 PRIMARY KEY(employeeid)

Behind the scenes, sql server kreira index da bi naterao uniqueness logickog primary key constrainta. Unique index je fizicki objekat kojeg sql server koristi da enforceuje uniqueness.

Indexi takodje se koriste da bi se speed up queries da bi se izbegao sorting ili nepotrebni full table scans.

### Unique constraints

Enforceuje uniqueness row-a. Dozvoljava ti implementaciju koncepta alternate key-a.

Mozes imati vise unique constraintova na atributima tabele. Kolona moze biti i null, nije restricted samo na not null kolone.

ALTER TABLE dbo.Employees  
 ADD CONSTRAINT UNQ\_Employees\_ssn  
 UNIQUE(ssn)

Behind the scenes kreira se isto unique index.   
Jedna interesantna stvar, po SQL-u unique se definise kao “Unique constraint na T je zadovoljena akko ne postoje dva reda R1 i R2 od T takvih da R1 i R2 imaju istu ne null vrednost”.

Znaciu unique se ubraja samo za vrednosti. Kako je null nepostojanje vrednosti, mi trebali imati vise redova sa unique attributom gde je null “vrednost”.

To je kako SQL standard definise null handling unique constraintova.

Po SQL Serveru ti ako stavis unique na nullable atribut mozes imati samo jedan null od svih redova.

Ukoliko zelis unique po SQL standardu:

CREATE UNIQUE INDEX idx\_ssn\_notnull ON dbo.Employees(ssn) WHERE ssn IS NOT NULL

### Foreign key constraints

Foreign key razresafa referentni integritet. Poenta FK je da ogranici vrednosti u FK koloni na one koje postoje u referenced columns.  
  
CREATE TABLE dbo.Odrders  
(  
orderid INT NOT NULL,  
custid INT NOT NULL,  
desc VARCHAR(20),  
qty INT NOT NULL,  
CONSTRAINT PK\_Orders  
 PRIMARY KEY(orderid)  
)

I sad da ubacis FK:  
ALTER TABLE dbo.Orders  
 ADD CONSTRAINT FK\_Orders\_Employees  
 FOREIGN KEY(empid) REFERENCES dbo.Employees(empid)

Postoje “referencing” i “referenced” tabele u FK. Referencing je one koje pozivaju referenced tabelu. Referencing je orders. Referenced je employees.

Nulls su dozvoljeni u FK po defaultu. Cak iako PK ne dozvoljava NULL.

Ovako gore opisani FK je samo za enforceovanje referential action-a koji se naziva “NO ACTION”.

To znaci da ukoliko pokusas da obrises row iz referenced table ili update referenced candidate key attributes, to ce biti rejected ako related rows postoje u referencing tabelama.

Mozes definisati FK sa actions.

Postoje opcije ON DELETE i ON UPDATE sa akcijama poput CASCADE, SET DEFAULT i SET NULL.

CASCADE – operacije ce se cascadeovati u related rows. ON DELETE CASCADE znaci da ce se i delete preneti.

SET DELETE i SET NULL – operacije koje u referenced tabeli menjaju candidate key atribute, postavice vrednost u referencing tabelama na default/null vrednost. Ovde ce postojati potom “orphan rows”

Parent rows with no related child rows su ovek allowed.

### CHECK Constraints

Definises predicate koji row mora da ispostuje da bi bio dodat/modifikovan.

ALTER TABLE dbo.Employees  
 ADD CONSTRAINT CHK\_Employees\_salary  
 CHECK(salary > 0.00);

Modifikacije se priznaje samo kada je evaluacije TRUE ili UNKNOWN. Npr salary od -1000 ce biti rejected, ali 50000 i NULL bice accepted. Ako je null dozvoljen.

### Default constraint

Vezuje se za neki konkretni atribut. Koristi se kako bi osigurao vrednost nekog atributa ukoliko ne bude bio prosledjen pri kreiranjnu reda.

ALTER TABLE dbo.Orders  
ADD CONSTRAINT DFT\_Orders\_orderts  
DEFAULT(SYSDATETIME()) FOR orderts;

Default expression invokeuje SYSDATETIME fn koja uzvraca sa current date i time vrednosti.

## Single Table Queries

### SELECT Statement

Ovaj statement sluzi za vrsenje upita nad tabelom, primenjivanje logickih manipulacija i vracanje rezultata. Ovde ce biti opisan i logicki redosled po kom se razliciti query upiti procesiraju(logical query processing).

DB Engine ima svoj query optimizer koji cesto dodaje dosta transformacija i precica u “fizickom procesiranju queryja” kao deo optimizacije.

SELECT empid, YEAR(orderdate) as orderyear, COUNT(\*) AS numorders   
FROM orders  
WHERE custid = 71  
GROUP BY empid, YEAR(orderdate)  
HAVING COUNT(\*) > 1  
ORDER BY empid, orderyear

U vecini programskih jezika bi se stvari izvrsavale redosledom kako ih ispises, ali u SQL je malo drugacije. U gore queryju je redosled sledeci(logical order):  
1. FROM (queries the rows from the orders table)  
2. WHERE (filters only orders where custid is 71)  
3. GROUP BY (groups the orders by employee id and orderdate)  
4. HAVING ( filters only groups(emp id and ord yr) having more than one order)  
5. SELECT ( returns emp id, ord yr & num of orders for each group)  
6. ORDER BY (sorts the rows in the output)  
  
Iako sintaksno ide prvo “select”, on se skoro poslednji izvrsava

### FROM clause

U ovom clause specificiras imena tabela nad kojima zelis da vrsis upit.

**FROM Sales.Orders**

Kada selektujes sve recorde iz tabele, ne moras da delimitujes identifiere(kolone).  
Ovako nesto: **SELECT id, name, surname FROM users** je sasvim okej.

Medjutim ukoliko je tvoja kolona “reserved keyword”, pocinje s brojem, sadrzi razmak i slicno – onda koristis T-SQL specific formu za delimiting ili navodnike:  
**SELECT id, “Order Details”, [Full Name] from [dbo].[Users]**

### Where Clause

Definises predikat ili logicki izraz na osnovu kog se filtriraju rezultati iz FROM clause-a. Oni redovi koji su evaluirani sa “true” su vraceni, a ostali odbaceni. Pod ostali to su “FALSE” i “UNKNOWN”. Zapamti – three-valued predicate logic.

WHERE izraz moze mnogo da utice na performans, na primer ako u query-u uzmes izraz koji koristi indexe, onda ces mnogo brze filtrirati rezultate nego da se radi full table scan.

### GROUP BY Clause