Syntex

SQL 구문은 (;)으로 끝난다.

대소문자 구분이 없다.

키워드는 대문자, 테이블명은 첫 문자만 대문다, 그 외 컬럼명 등은 소문자 관례

Table에 저장한 문자열의 경우는 대문자, 소문자를 구별. 'ABC'와 'abc'

SQL 구문은 작성하는 순서가 있어 순서대로 기재해야 한다.

SQL에 직접 기술하는 문자열, 날짜, 숫자 등을 상수라고 한다.

문자열, 날짜 상수은 ('), 숫자 상수는 숫자 그대로 사용

한글 별명은 (") 사용

단어와 단어 사이는 공백 또는 줄바꿈으로 구분

1행 주석은 (--) 뒤에

복수행 주석은 (/* */) 사이에

database, table, column 이름은 영문자, 숫자, 언더바 만 사용 가능.

이름 첫 글자는 '영문자'로 해야 함.

가능하면 집계함수(집약함수)를 사용하는 것이 성능향상에 좋다. 성능차 많이 난다.

기술 순서 : SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY

실행 순서 : FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

쿼리문을 만들 때 열 이름에는 가공을 하지 말아라. Index가 있어도 가공을 하면 전체 Scan을 하여 속도가 느려진다.

Summary

<< Backup >>	
BACKUP	**USE** tempDB; **BACKUP DATABASE** ShopDB **TO DISK** = 'D:\sqlDB2019.bak' **WITH INIT**;
RESTORE	**USE** tempDB; **RESTORE DATABASE** ShopDB **FROM DISK =** 'D:\ShopDB.bak' **WITH REPLACE**; **USE** ShopDB;
<< Database >>	
CREATE	USE tempdb; CREATE DATABASE shopDB;
EXCUTE	EXEC sp_helpdb;
USE	USE shopDB;
DROP	USE tempdb; DROP DATABASE shopDB; -- Table과 Data 있어도 삭제 됨.
<< Schema >>	CREATE SCHEMA userSchema;
	CREATE TABLE userSchema.userTbl (id CHAR(8));
<< Table >>	
EXECUTE	EXEC sp_tables @table_type = " ' TABLE' ";

Summary (cont)

CREATE	USE shopDB; CREATE TABLE userTbl (userID CHAR(8) NOT NULL PRIMARY KEY, -- CONSTRAINT PF_name PRIMARY KEY CLUSTERED name CHAR(8) NOT NULL UNIQUE, birthYear SMALLINT NOT NULL DEFAULT YEAR(GETDATE( )) CHECK (birthYear>1900), hobby NVARCHAR(10) SPARSE NULL);
	CREATE TABLE buyTbl (num INT NOT NULL IDENTITY(1, 1), userID CHAR(8) NOT NULL FOREIGN KEY REFERENCES userTbl(userID), prodName NVARCHAR(20) NOTNULL, amount INT, CONSTRAINT PK_num PRIMARY KEY CLUSTERED (userID)), CONSTRAINT CK_birthYear CHECK (birthYear>1900);
DROP	DROP TABLE userTbl;
RENAME	sp_rename 'userTbl', 'userTbl1';
<< Alter - Column >>	
EXEC	EXEC sp_help buyTbl;
ADD *colname*	ALTER TABLE userTbl ADD weight SMALLINT NULL;
ALTER COLUMN	ALTER TABLE userTbl ALTER COLUMN weight INT NULL; -- NOT NULL로 변경 시 이전 값 체크함.
DROP COLUMN	ALTER TABLE userTbl DROP COLUMN weight; -- Data 있어도 삭제 됨.
RENAME	EXEC sp_rename 'userTbl.weight', 'myWeight', 'COLUMN';
<< Alter - Constraint >>	
ADD CONSTRAINT	-- DEFAULT(이전 값 체크 안함), CHECK / UNIQUE (이전 값 체크),
PRIMARY KEY	ALTER TABLE userTbl ADD CONSTRAINT PK_userTbl_userID PRIMARY KEY (userID, name);
FOREIGN KEY	ALTER TABLE buyTbl ADD CONSTRAINT FK_buyTbl_userTbl FOREIGN KEY (userID) REFERENCES userTbl(userID);
ON UPDATE ON DELETE	ALTER TABLE buyTbl ADD CONSTRAINT FK_buyTbl_userTbl FOREIGN KEY (userID) REFERENCES userTbl(userID) ON UPDATE CASCADE ON DELETE RESTRICT;
UNIQUE	ALTER TABLE userTbl ADD CONSTRAINT UN_name UNIQUE(name);
CHECK	ALTER TABLE userTbl ADD CONSTRAINT CK_birth CHECK (birthYear >= 1900 AND birthYear <= YEAR(GETDATE-()));

Summary (cont)	
DEFAULT	ALTER TABLE userTbl ADD CONSTRAINT DF_birthYear DEFAULT YEAR(GETDATE()) FOR birthYear; -- INSERT INTO userTbl VALUES (2016, DEFAULT, 195); -- 컬럼 지정 안하면 DEFAULT 입력. -- INSERT INTO userTbl(birtyYear, height) VALUES (2016, 195); -- 아니면 컬럼 지정하여 입력. -- INSERT INTO userTbl VALUES (2016, N'서울', 195); -- DEFAULT에 다른 값 입력 가능.
DROP CONSTRAINT	ALTER TABLE buyTbl DROP CONSTRAINT FK_buyTbl_userTbl;
	ALTER TABLE userTbl DROP CONSTRAINT PK_userTbl_userID;
변경	변경은 기존 Constraint DROP 후 새로 ADD CONSTRAINT 한다.
NOCHECK CONSTRAINT	ALTER TABLE userTbl NOCHECK CONSTRAINT CK_mobile; -- 제약사항 임시 해제. 사용 후 다시 체크 시작할 것.
CHECK CONSTRAINT	ALTER TABLE userTbl CHECK CONSTRAINT CK_mobile;
<< Index >>	
CREATE INDEX - ON	CREATE INDEX IDX_userTbl_addr ON userTbl(addr);
	CREATE UNIQUE INDEX IDX_userTbl_addr ON userTbl(addr);
DROP INDEX - ON	DROP INDEX IDX_userTbl_addr ON userTbl; -- CREAT TABLE에서 PRIMARY KEY / UNIQUE로 생성된 것은 삭제 안됨
EXEC	EXEC sp_helpindex userTbl;
<< Identity >>	
SET IDENTITY INSERT - ON	CREATE TABLE identTbl (num INT NOT NULL IDENTITY, name CHAR(3)); SET IDENTITY_INSERT identTbl ON; INSERT INTO identTbl(num, name) VALUES (10, 'CCC'); -- 'num' 열 이름을 기재해야 함.
SET IDENTITY INSERT - OFF	SET IDENTITY_INSERT identTbl OFF;
SELECT IDENT_CURRENT( )	SELECT IDENT_CURRENT('identTbl'); -- (') 필요
SELECT @@IDENTITY	SELECT @@IDENTITY; -- 현재 쿼리 창 기준
<< Insert >>	
BEGIM TRAN	BEGIM TRANSACTION;
INSERT INTO - VALUES	INSERT INTO Goods VALUES ('0001', '의류', NULL, DEFAULT); -- NULL 직접 기술 가능. DEFAULT 기술 추천.

Summary (cont)	
	INSERT INTO Goods(id, name) VALUES(0001, '의류'); -- NOT NULL 컬럼을 제외하고 특정 컬럼 지정하여 입력 가능.
	INSERT INTO Goods(id, name) VALUES (0001, '의류'), (0002, '사무');;
INSERT INTO - SELECT	CREATE TABLE newTbl (e INT, f INT, g INT); -- 입력할 테이블이 만들어져 있어야 한다. INSERT INTO newTbl SELECT a, b, c FROM oldTbl;
	INSERT INTO newTbl(e, f) SELECT b, a FROM oldTbl;
	INSERT INTO targetTbl SELECT * FROM updateTbl WHERE NOT EXITS (SELECT a FROM targetTbl WHERE target-Tbl.a = updateTbl.a);
SAVE TRAN	SAVE TRANSACTION;
ROLLBACK TRAN	ROLLBACK TRANSACTION; COMMIT 되기 전 취소. CHECK 제약 조건 때문에 에러 발생한 경우 자동 ROLL BACK 되지는 않는다.
COMMIT TRAN	COMMIT TRANSACTION;
<< *Update* >>	
UPDATE - SET	BEGIN TRY BEGIN TRAN UPDATE userTbl SET mData = '0000-00-00'; -- 전체 바뀜 UPDATE userTbl SET height = height * 0.01, mDate = NULL WHERE addr = '경북'; -- NULL CLEAR : NOT NULL 조건 없어야 함. COMMIT TRAN END TRY BEGIN CATCH ROLLBACK TRAN SELECT ERROR_NUM( ) SELECT ERROR_MESSAGE( ) END CATCH
<< *Delete* >>	
TRUNCATE TABLE	TRUNCATE TABLE buyTbl; -- 빠름.
DELETE FROM	DELETE FROM userTbl; --한 줄씩 삭제. 느림. 테이블은 남음.
	DELETE FROM userTbl WHERE addr = '서울';
	DELETE TOP(3) FROM userTbl WHERE addr = '서울'; -- 무작위로 3명 삭제됨.

Summary (cont)

DELETE FROM userTbl WHERE userID IN (SELECT TOP(2) userID FROM userTbl WHERE addr = '서울' ORDER BY height);
-- 우선 순위 삭제 방법. DELETE -WHERE문과 하위 SELECT 문에서 기본 KEY를 지정해야 함.

DataBase

< BACKUP >	----------	
BACKUP	**USE** tempDB; **BACKUP DATABASE** ShopDB **TO DISK** = 'D:\sqlDB2019.bak' **WITH INIT**;	DATABASE 단위로 백업함. 폴더 미리 생성.
RESTORE	**USE** tempDB; **RESTORE DATABASE** ShopDB **FROM DISK =** 'D:\ShopDB.bak' **WITH REPLACE**; **USE** ShopDB;	데이터베이스를 일단 다른 곳으로 바꾸고 진행할 것.
< EXECUTE>	----------	
	EXECUTE sp_helpdb;	SQLServer
	EXECUTE sp_help Goods;	테이블 정보
	EXECUTE sp_depends userTbl;	참조 Table 확인
	EXECCUTE sp_helpindex userTbl;	index 확인
	SELECT OBJECT_NAME(object_id) as [뷰 이름], definition **FROM sys.sql_modules**;	View / Procedure 확인
SHOW	**SHWO DATABASES**;	MySQL
	SHOW TABLES;	MySQL
기타	**EXECUTE sp_tables @table_type = " ' TABLE' "**;	
	EXECUTE sp_columns @table_name = 'Department', **@table_owner** = 'HumanResources';	
USE	**USE** shop;	
< Database >	----------	
CREATE DATABASE	**USE** tempdb; **CREATE DATABASE** shop;	
USE	**USE** shopDB;	
DROP	**USE** tempdb; **DROP DATABASE** shopDB;	안에 있는 테이블도 모두 삭제됨.

DataBase (cont)

< Schema >	----------
CREATE SCHEMA	**CREATE SCHEMA** userSchema;
CREATE TABLE	**CREATE TABLE** userSchema.userTbl (id INT);
SELECT	**SELECT** * **FROM** userSchema.userTbl;
< View >	----------
Syntex	원 Table에 NOT NULL이 있는 Column이 있으면 INSERT가 안된다.
	JOIN 문으로 만들어진 View는 원칙적으로는 INSERT 가 안된다. INSTEAD OF TRIGGER를 사용해야 한다.
	VIEW 생성 시 WITH CHECH OPTION을 사용하야 조건에 맞지 않는 data 입력 시 Error를 발생시켜준다.
	VIEW 가 참조하고 있는 Table도 그냥 삭제가 된다. 삭제되면 VIEW는 더 이상 사용할 수 없다. 따라서 삭제 전 EXEC sp_depends 로 먼저 확인하는 것이 좋다.
CREATE VIEW - AS	**CREATE VIEW** v_userTbl **AS SELECT** userID, name, addr **FROM** userTbl
	CREATE VIEW v_sum **AS SELECT** userID, **SUM**(price*amount) AS [total] **FROM** buyTbl **GROUP BY** userID;
- WHTI CHECK OPTION	**CREATE VIEW** v_height177 **AS SELECT** * **FROM** userTbl **WHERE** height >= 177 **WHTI CHECK OPTION**;
	-- CAUTION : WITH CHECH OPTION이 없으면 INSERT로 height 177 이상의 data를 입력해도 입력이 된다. 입력후 조회는 안된다. 이를 방지하기 위해 사용.
- WITH ENCRIPTION	**ALTER VIEW** v_userTbl **WHTI ENCRIPTION AS SELECT** userID, name, addr **FROM** userTbl 암호화(복구 안됨)
SELECT	**SELECT** * **FROM** v_userTbl;

DataBase (cont)

	읽기 전용으로 사용할자.
	복잡한 쿼리문을 View로 만들어 두면 간단하게 SELECT 문으로 조회할 수 있다.
ALTER VIEW - AS	**ALTER VIEW** v_suerTbl **AS SELECT** userID AS [아이디], name AS [이름], addr **FROM** userTbl
DROP VIEW	**DROP VIEW** v_userTbl;
내용 확인	**SELECT OBJECT_NAME(object_id) as** [뷰 이름], definition **FROM sys.sql_modules**;
참조관계 확인	**EXEC sp_depends** userTbl;
UPDATE	**UPDATE** v_userTbl **SET** addr = N'서울' **WHERE** userID = 'JKW';
INSERT	**INSERT INTO** v_userTbl (userID, name, addr) **VALUES** ('KBM', '김병만', '충북');
	-- 원 userTbl의 birthYear이 NOT NULL로 되어 있어 Error이 발행함. 편법적으로 NOT NULL을 NULL로 변경 후 삽입하여야 한다. 원칙적으로 이 경우 INSERT 불가 함.

DDL - Data Definition Language

< Create >

Syntex	NULL도 습관적으로 적어주자. NULL이 자주 되는 곳은 SPARSE NULL로 메모리 관리 할 것. NULL이 별로 없는 경우에 지정하면 더 않좋음. 최소 60%는 되어야 함. SELECT 검색 속도는 오히려 느려짐 Table 압축도 안된다.
	테이블 이름 앞에 '#'을 붙이면 tempdb에 임시 테이블이 생성 된다. 생성한 쿼리창에서만 인식되고 쿼리창을 닫으면 삭제된다. 다른 쿼리창에서도 사용하려면 '##'을 붙여서 만들면 된다.

DDL - Data Definition Language (cont)

	데이터 무결성을 위한 제약조건으로 NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK 5가지가 있다.
CREATE TABLE	**CREATE TABLE** Goods (id **CHAR**(4) NOT NULL **PRIMARY KEY**, name **VARCHAR**(100) NOT NULL, price **INTEGER** NULL **DEFAULT** 0, date **DATE** NULL);
	CREATE TABLE Goods (num **INT IDENTITY** NOT NULL, id **CHAR(8)** NOT NULL **FOREIGN KEY REFERENCES** Goods**(id)**, amount **SMALLINT** NULL);
- SPARSE NULL	**CREATE TABLE** userTbl (name NCHAR(8) NOT NULL, hobby NVARCHAR(10) **SPARSE NULL**);
< PRIMARY KEY >	
CREATE	중복 값이 없는 열. NULL은 안됨. 키본 키는 하나(두 개 열을 합쳐서 설정 가능)
- 바로 지정	**CREATE TABLE** userTbl (userID CHAR(8) NOT NULL **PRIMARY KEY**);
- 이름 생성	**CREATE TABLE** userTbl (userID CHAR(8) NOT NULL **CONSTRAINT** *PK_useTbl_userID* **PRIMARY KEY**);
- 마지막에 지정	**CREATE TABLE** userTbl (userID CHAR(8) NOT NULL**, CONSTRAINT** *PK_useTbl_userID* **PRIMARY KEY(** *userID* **)**);
	, 로 구분하여 마지막에 지정. 지정할 컬럼명을 넣어줘야함.
ALTER - ADD	**ALTER TABLE** userTbl **ADD CONSTRAINT** *PK_useTbl_userID* **PRIMARY KEY**(userID));
복수열 지정	복수의 열을 합쳐서 하나의 값으로 지정, 합쳐진 값이 중복되면 안됨.
	CREATE TABLE prodTbl (prodCode CHAR(3) NOT NULL, prodID CHAR(4) NOT NULL**,** **CONSTRAINT** *PK_prodTbl_prodCode_prodID* **PRIMARY KEY(** *prodCode, prodID* **)**)

DDL - Data Definition Language (cont)

	ALTER TABLE prodTbl **ADD CONSTRAINT** *PK_prodTbl_prodCode_prodID* **PRIMARY KEY**(prodCode, prodID));
< FOREIGN KEY >	
CREATE	PRIMARY / UNIQUE KEY에만 지정 가능.
- 바로 지정	**CREATE TABLE** buyTbl (userID CHAR(8) NOT NULL **FOREIGN KEY REFERENCES** userTbl(userID));
- 이름 생성	**CREATE TABLE** buyTbl (userID CHAR(8) NOT NULL **CONSTRAINT** *FK_userTbl_buyTbl***FOREIGN KEY REFERENCES** userTbl(userID));
- 마지막에	**CREATE TABLE** buyTbl (userID CHAR(8) NOT NULL, **CONSTRAINT** *FK_userTbl_buyTbl***FOREIGN KEY**(userID) **REFERENCES** userTbl(userID));
ALTER - ADD	**ALTER TABLE** buyTbl **ADD CONSTRAINT** *FK_userTbl_buyTbl***FOREIGN KEY**(userID) **REFERENCES** userTbl(userID);
- ON UPDATE CASCADE	**ALTER TABLE** buyTbl **ADD CONSTRAINT** FK_userTbl_buyTbl **FOREIGN KEY** (userID) **REFERENCE** userTbl(userID) **ON UPDATE CASCADE**; PK 변경 자동 반영
- ON DELETE RESTRICT	**ALTER TABLE** buyTbl **DROP CONSTRAINT** FK_userTbl_buyTbl; **ALTER TABLE** buyTbl **ADD CONSTRAINT** FK_userTbl_buyTbl **FOREIGN KEY** (userID) **REFERENCE** userTbl(userID) **ON UPDATE CASCADE ON DELETE RESTRICT**; 제약조건 삭제 후 다시 지정
	--CASCASE / SET NULL / SET DEFAULT / NO ACTION / RESTRICT
	SSMS - Table 우클릭 - 디자인 - Table sheet 바탕화면 우클릭 - 관계 - 추가 - 테이블 및 열 사양 - ...클릭 - (기본 키 테이블 / 지정할 열) & (외래 키 테이블 / 지정할 열) 선책
< UNIQUE >	
CREATE	중복되지 않는 유일한 값. NULL 허용

DDL - Data Definition Language (cont)	
- 바로 지정	**CREATE TABLE** userTbl (userID CHAR(8) NOT NULL **PRIMARY KEY**, email CHAR(30) NULL **UNIQUE**);
- 이름 생성	**CREATE TABLE** userTbl (userID CHAR(8) NOT NULL **PRIMARY KEY**, email CHAR(30) NULL **CONSTRAINT** AK_email **UNIQUE**);
- 마지막에 지정	**CREATE TABLE** userTbl (userID CHAR(8) NOT NULL **PRIMARY KEY**, email CHAR(30) NULL, **CONSTRAINT** AK_email **UNIQUE(**email**)**);
ALTET - ADD	**ALTER TABLE** userTbl **ADD CONSTRAINT** UN_name **UNIQUE**(name);
< CHECK >	
CREATE	**CREATE TABLE** userID (birthYear NULL **CHECK** (birthYear>1900));
	CREATE TABLE userID (birthYear NULL, **CONSTRAINT** CK_birthYear **CHECK** (birthYear>1900));
ALTER - ADD	**ALTER TABLE** userTbl **ADD CONSTRAINT** CK_birth **CHECK** (birthYear >= 1900 AND birthYear <= YEAR(GETDATE( )));
	ALTER TABLE userTbl **ADD CONSTRAINT** CK_mobile1 **CHECK** (mobile1 IN('010', '011', '016'));
	ALTER TABLE DocExc **ADD** ColumnD INT NULL **CONSTRAINT** CHK_ColumnD_DocExc **CHECK** (ColumnD > 10 AND ColumnD < 50);
ALTER - WITH NOCHECK ADD	**ALTER TABLE** userTbl **WITH NOCHECK ADD CONSTRAINT** CK_mobile **CHECK** (mobile1 IN('010', '011', '016'));
	-- WITH NOCHECK는 모든 CONSTRAINT 설정 시 사용 가능.
< DEFALUT >	
CREATE	**CREATE TABLE** userTbl (birthYear INT NOT NULL **DEFAULT** YEAR(GETDATE( ), addr NCHAR(2) NOT NULL **DEFALUT** N'서울', height SMALLINT **DEFAULT**170);
ALTER	**ALTER TABLE** userTbl **ADD CONSTRAINT** DF_birthYear **DEFAULT** YEAR(GETDATE( )) **FOR** birthYear;
INSERT	**INSERT INTO** userTbl **VALUES** (2016, DEFALUT, 195);

DDL - Data Definition Language (cont)

	INSERT INTO userTbl(birthYear, height) **VALUES** (2016, 195);	
	INSERT INTO userTbl **VALUES** (2016, N'인천', 195);	
< IDENTITY >		
	자동 순번 생성(Server) .INSERT 시 값 입력하면 안됨. 자동으로 NOT NULL 지정 됨.	
	INT **IDENTITY**	
	INT **IDENTITY(1, 2)**	초기 값 1, 증가 값2
	SSMS TABLE 생성 - 열 속성 - ID 사양을 '예'로 변경하면 동일한 효과	PRIMARY KEY로 지정되 것이 ID임.(num)
- SEQUENCE	IDENTITY 대신 SEQUENCE 객체를 사용할 수 있다. 오라클과의 호완성을 위해 만듬. Transaxt-SQL 기본(5) 20:00	
< 데이터 형 >	----------	
- 문자형	CHAR(5)/VARCHAR(10) / VARCHAR(max)	영어(기호) 고정 5자/가변 10자 / 최대 8천자
	NCHAR(5) / NVARCHAR(10) / NVARCHAR(max)	한글 고정 5자 / 가변 10자 /최대 4천자
	VARBINARY(max)	이미지, 동영상 저장에 사용
	CAST(@MovieScript **AS NVARCHAR**(**MAX**))	로 변형하면 최대 2GB까지 저장 가능
- 숫자형	BIT	0 or 1
	TINYINT / SMALLINT / INT / BIGINT	+255 / ±3.2만(+ 6.5만) / ±21억(+42억) / 이상
	DECIMAL(5,2)	전체 5자리 중 소수점 이하 2자리 사용
	FLOAT / DOUBLE	작은 / 큰 부동소수점
- 날짜형	DATE/TIME/DATETIME2	
- 기타	CURSOR	T-SQL 커스를 변수로 처리
	TABLE	테이블 자체를 저장. 임시 테이블과 비슷한 기능.
	XML	XML 데이터 형식 저장. 최대 2GB
< Rename >	----------	

DDL - Data Definition Language (cont)

RENAME TABLE	**sp_rename** 'Gods', 'Goods';	SQL Server
	-- RENAME TABLE Gods to Goods;	MySQL
DROP TABLE	**DROP TABLE** Goods;	Table 자체 삭제, FK 테이블부터 삭제해야 함.
< ALTER TABLE - Column >		
ADD *colname*	**ALTER TABLE** Goods **ADD** name_eng VARCHAR(100);	SQL Server Only
	-- SSMS에서 drag해서 열 순서를 바꿀 수 있다.	
	-- ALTER TABLE Goods ADD COLUMN name_eng VARCHAR(100);	MySQL
ALTER COLUMN	**ALTER TABLE** userTbl **ALTER COLUMN** hobby NVARCHAR(10) NOT NULL;	
	-- 데이터 유형, 크기 변경. 숫자는 문자로 가능, 반대는 에러. 크기가 작아도 에러. NULL 있는 데이터를 NOT NULL로 바꿔도 ERROR. 아래와 같이 업데이트 후 사용해야 함. **UPDATE** userTbl **SET** hobby = ' ' **WHERE** hobby IS NULL;	
DROP COLUMN	**ALTER TABLE** Goods **DROP COLUMN** name_eng;	제약조건 걸려있으면 제약조건 먼저 삭제해야 함.
EXEC	**EXEC sp_rename 'userTbl.**uesrID', 'ID', **'COLUMN';**	컬럼명 변경
< ALTER TABLE - Constraint >		
ADD CONSTRAINT	**ALTER TABLE** userTbl **ADD CONSTRAINT** DF_birthYear **DEFAULT** YEAR(GETDATE( )) **FOR** birthYear;	
DROP CONSTRAINT	**ALTER TABLE** userTbl **DROP CONSTRAINT** DF_birthYear;	
NOCHECK CONSTRAINT	**ALTER TABLE** buyTbl **NOCHECK CONSTRAINT** FK_userTbl_buyTbl;	FK 제약을 중지 시킨다.(임시로 사용 가능)
CHECK CONSTRAINT	**ALTER TABLE** buyTbl **CHECK CONSTRAINT** FK_userTbl_buyTbl;	다시 CONSTRAINT 실행
변경	기존 Constraint DROP 후 새로 ADD CONSTRAINT 한다.	

Index

Syntex	과용하면 성능 저하가 발생한다.

Index (cont)

WHERE 절에 Index를 생성한 열의 이름이 나와야 Index 검색을 한다. 따라서 WHERE 절에 사용되는 열에 INDEX를 만들어라.

WHERE 절에 사용하더라도 자주 사용하는 것만 만들어라. 가끔사용하는 것은 차라리 한 번 느려지는 것이 낫다. 인덱스가 있으면 INSERT / UPDATE /DELETE 등에 더 느려진다. INSERT 를 자주 사용하면 안 만드는 것이 좋다.

데이터의 종류가 몇가지 않되는 컬럼에는 INDEX를 만들어도 SQL이 자동으로 사용하지 않는다. 따라서 용량만 커지므로 만들지 않는 것이 좋다. (핸드폰 국번 등)

사용하지 않는 Nonclustered Index는 제거하는 것이 좋다. 검색 시간이 더 걸릴 수 있다.

외래 키가 사용되는 열에는 인덱스를 되도록 생성해주는 것이 좋다.

JOIN에 자주 사용되는 열에는 인덱스를 생성해주는 것이 좋다.

Index는 Clusteder 형과 NonClustered 형으로 만들 수 있다.

Clustered 형은 db를 index별로 정렬하여 별도로 저장하고, NonClustered 형은 Index Page만 만들고 db는 순차적으로 저장한다.

Clustered 형 Index는 table당 하나만 만들 수 있다. PRIMARY KEY 또는 UNIQUE 제약으로 만들 수 있다.

Clustered Index를 생성하는 열은 범위나 집계함수, ORDER BY절에 자주 사용하는 열의 경우에 효과적이다. 이미 정렬되어 있기 때문이다.

Index (cont)	
	Insert가 대량으로 일어나는 경우 Clustered index가 있는 경우 성능 저하가 심할 수 있어 무조건 clustered index가 좋은 것은 아니다. PRIMARY KEY NONCLUSTERED 로 지정하여 Nonclustered형으로 만드는 것이 좋다.
	Clustered형은 검색은 빠르나 Insert, Alter에는 Nonclustered형이 더 빠르다.

	Index는 하나 또는 여러 열로 만들 수 있다.
	Index는 제약조건 없이 만들 수 없다.
	Primary Key와 Unique 제약조건을 걸면 자동으로 Index가 생성된다.
	여러 걸럼에 걸쳐 제약조건을 걸어 Index를 만들려면 CONSTRAINTS 제약조건면 (col1, col2) 로 만든다.
	PRIMARY KEY를 지정하면 CLUSTERED 형 index가 자동 생성된다. NONCLUSTERED 형으로 별도 지정하여 바꿀 수 있다.
	PRIMARY KET를 NONCLUSTERED로 지정하면 UNIQUE를 CLUSTERED 형으로 지정할 수 있다.
	PRIMARY KEY로 Clustered Index를 만들어도 UNIQUE 는 Nonclustered Index로 여러 개 만들 수 있다.
	PRIMARY KEY와 UNIQUE 제약 조건으로 자동 생성된 index는 DROP INDEX로 제거할 수 없다. ALTER TABLE - DROP CONSTRAINT로 제거해야 여기에 자동 생성된 index가 제거 된다.

CREATE INDEX	따로 지정하지 않으면 NONCLUSTERED 형으로 만들어 진다.

Index (cont)

CREATE INDEX - ON	**CREATE INDEX** idx_userTbl_addr **ON** userTbl(addr);	
	CREATE UNIQUE INDEX idx_userTbl_name **ON** userTbl(name);	
DROP INDEX	PRIMARY KEY와 UNIQUE 제약 조건으로 자동 생성된 index는 DROP INDEX로 제거할 수 없다. ALTER TABLE - DROP CONSTRAINT로 제거해야 여기에 자동 생성된 index가 제거 된다.	
	DROP INDEX idx_userTbl_addr **ON** userTbl;	
	DROP INDEX userTbl. idx_userTbl_addr;	테이블 명을 적어줘야 한다.
ALTER INDEX	**ALTER INDEX**	
확인	**EXEC sp_helpindex** userTbl;	
	SSMS - 도구 옵션 - 쿼리 실행 - SQL Server - 고급 - SET STATISTICS IO	참조 페이지 수 확인

Memory Table

준비	기본 키 및 비클러스터형 인덱스 필요. NONCLUSTERED 예약어를 PRIMARY KEY와 함께 사용해야 함.
	1. dataBase 생성
	2. dataBase - 속성 - 파일 그룹 - 메모리 액세스에 최적화된 데이터 - 파일 그룹 추가
	3. dataBase - 속성 - 파일 - 추가 - 논리적 이름 만들기(임의) - '파일 형식'은 DILESTREAM 데이터로 지정 - '파일 그룹'은 2번 그룹과 동일해야 함.
CREATE TABLE	**CREATE TABLE** *memoryTbl* (a INT **PRIMARY KEY NONCLUSTERED**, b NCHAR(100)) **WHTH (MEMORY_OPTIMIZED=ON)**;
CREATE PROCEDURE	**CREATE PROCEDURE** usp_diskInsert @data NCHAR(100) **AS** **DECLARE** @I INT = 1; **WHILE** @i <= 500 **BEGIN** **INSERT INTO** dbo.diskTable(a, b) **VALUES** (@i, @data); **SET** @I += 1; **END**

Memory Table (cont)

CREATE PROCEDURE	**CREATE PROCEDURE** usp_memoryInsert @data NCHAR(100) **WITH NATIVE_COMPLATION, SCHEMABINDING** **AS** **BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE=N'Korea')** **DECLARE** @I INT = 1; **WHILE** @i <= 500 **BEGIN** **INSERT INTO** dbo.diskTable(a, b) **VALUES** (@i, @data); **SET** @I += 1; **END** **END**
	DECLARE @sendData nchar(100) = REPLICATE(N'가', 100); **EXECUTE** usp_diskIsert @sendDatal

DML - Data Manipulation Language

BEGIN TRANSACTION	**BEGIN TRANSACTION**; / START TRANSACTION;	SQL Server / MySQL Only
INSERT INTO - VALUES	**INSERT INTO** Goods **VALUES (**'0001', '의류', 1000, '2019-09-20**)**	열 리스트 생략 가능
	INSERT INTO Goods **VALUES (**'0001', '의류', NULL, '2019-09-20**)**	NULL 직접 기술
	INSERT INTO Goods **VALUES (**'0001', '의류', 1000, '2019-09-20**)**, **(**'0002', '사무', 500, '2009-02–03'**)**;	복수 행 삽입
	INSERT INTO Goods **VALUES (**'0001', '의류', DEFAULT, '2019-09-20**)**	DEFALUT 직접 기술 추천
	-- 자동 생성 순번은 입력하면 안된다.	
	INSERT INTO Goods(id, name) **VALUES(**0001, '의류');	
	-- NOT NULL 컬럼을 제외하고 특정 컬럼 지정하여 값을 넣을 수 있다.	
INSERT INTO - SELECT - FROM	**INSERT INTO** NewTable **(**id, name, price**) SELECT** id, name, price **FROM** Goods; -- 입력할 테이블이 미리 만들어져 있어야 한다.	다른 테이블에서 데이터 복사
	INSERT INTO GoodClassify **(**classify, sum_price**) SELECT** classify, SUM(price) **FROM** Goods **GROUP BY** classify;	

DML - Data Manipulation Language (cont)

	-- INSERT 내 SELECT 문에는 WHERE, GROUP BY 등 모두 사용 가는. ORDER BY는 효과 없음	
	SELECT classify, sum_price **INTO** NewTable **FROM** Goods; -- 새로 생성할 테이블 자동으로 만들어 진다.	위와 동일함
SET IDENTUTY_INSERT - OFF	**SET IDENTITY_INSERT** Tbl1 **ON**; **INSERT INTO** Goods(num, id, name) **VALUE**(11, 0001, '의류');	강제로 바꾸로 싶을 때 사용 num을 지정하지 않으면 error 발생함.
SET IDENTUTY_INSERT - ON	**SET IDENTITY_INSERT** Tbl1 **OFF**;	끝나면 다시 OFF. 12부터 시작 함.
SELECT IDENT_-CURRENT	**SELECT IDENT_CURRENT**('Goods');	현재의 IDENTITY 값, (') 필요
SELECT @@IDENTITY	SELECT @@IDENTITY	현재 쿼리창의 가장 최근 IDENTITY 값
COMMIT TRAN	**COMMIT TRAN**;	SQL Server
	COMMIT;	
UPDATE - SET	**UPDATE** Goods **SET** date = '2009-1-2';	주의. 선택된 컬럼 전체 변경. 잘 사용 안함.
UPDATE - SET - WHERE	**UPDATE** Goods **SET** peice = price * 10 **WHERE** classify = '사무';	
	UPDATE Goods **SET** date = **NULL WHERE** id = '0008';	NULL 클리어. NOT NULL 제약 없어야 함.
	UPDATE Goods **SET** peicd = price * 10, date = '2019-01-02' **WHERE** classify = '사무';	복수열 갱신
TRUNCATE TABLE	**TRUNCATE TABLE** Goods;	Table 내 모든 레토드(만) 삭제, DELETE 보다 빠름.
DELETE FROM	**DELETE FROM** Goods;	<주의> Table 내 레코드만 삭제
DELETE FROM - WHERE	**DELETE FROM** Goods **WHERE** price >= 4000;	WHERE 구문만 가능
DELETE TOP( ) FROM - WHERE	**DELETE TOP**(10) **FROM** Goods **WHERE** name = 'Kim';	임의 순서로 삭제됨.
	DELETE FROM Goods **WHERE** *id* **IN** (**SELECT TOP**(10) *id* **FROM** Goods **ORDER BY** price);	기본 key를 지정해야 함.
MERGE	**MERGE** memberTBL AS M	변경될 테이블
USING	**USING** changeTbl AS C	변경할 기준이 되는 테이블
ON	**ON** M.userID = C.userID	두 테이블 비교할 기준

DML - Data Manipulation Language (cont)

WHEN	**WHEN NOT MATCHED AND** changeType = '신규가입' **THEN**
	.....**INSERT**(userID, name, addr) **VALUES**(C.userID, C.name, C.addr)
	WHEN MATCHED AND changeType = '주소변경' **THEN**
	.....**UPDATE SET** M.addr = C.addr
	WHEN MATCJED AND changeType = '회원탈퇴' **THEN**
	.....**DELETE**;

WITH - Common Table Expression

Syntex	임시 테이블을 만들어서 기 테이블의 값을 다시 SELECT로 사용.
WITH - AS	**WITH** cte_Tbl1(addr, maxHeight) **AS (SELECT** addr, **MAX**(height) **FROM** Tbl1 **GROUP BY** addr**)**
	.....**SELECT AVG**(maxHeight*1.0) **AS** '지역별 최고 키' **FROM** cte_Tbl1;

SELECT

Syntex	반환하는 결과값은 무작위 순이다. 메모리에 임시 생성하고 버린다.	
	SELECT 열 이름 **FROM** 테이블 이름 **WHERE** 행 필터;	
SELECT	**SELECT * FROM** Goods;	
	SELECT id, name FROM Goods;	
-- AS	**SELECT** id, name **AS** nm **FROM** Goods;	
	SELECT id **AS "**아이디**"**, name AS **"**이름**" FROM** Goods;	한글 별명
	SELECT '상품**' AS "**구분**"**, 10 **AS** num, **'**2009-02-24**' AS** date **FROM** Goods;	COLUMN에 상수 할당
	- 구분, num, date 컬럼 생성하고 값으로 '상물, 10, '209-02-24'를 모든 행에 추가한다.	
-- DISTINCT	**SELECT DISTINCT** classify, date **FROM** Goods;	해당 열에서 중복 제거하고 표시
-- TOP()	**SELECT TOP(10)** CreditCardID **FROM** Sales **WHERE** Type = 'Vista' **ORDER BY** ExpYear;	
	SELECT TOP(**SELECT** COUNT(*)/100 **FROM** Sales) CreditCardID FROM Sales **WHERE** Type = 'Vista' **ORDER BY** ExpYear;	

SELECT (cont)

-- TOP( ) PERCENT	**SELECT TOP(0.1) PERCENT** CreditCardID **FROM** Sales **WHERE** Type = 'Vista' **ORDER BY** ExpYear;	동률 출력 안 함
-- TOP( ) WITH TIES	**SELECT TOP(0.1) PERCENT WITH TIES** CreditCardID **FROM** Sales **WHERE** Type = 'Vista' **ORDER BY** ExpYear;	동률 전부 출력
-- INTO	TABLE을 새로 생성한다. Primary Key, Foreign Key는 복사되지 않는다.	
	SELECT * **INTO** Tbl2 **FROM** Tbl1;	
	SELECT ID, name **INTO** Tbl2 **FROM** Tbl1;	
FROM	**SLEECT** * **FROM** Sales **TABLESAMPLE(5 ROWS)**;	샘플 무작위 생성
-- TABLESAMPLE(5 ROWS)	**SLEECT TOP(500)** * **FROM** Sales **TABLESAMPLES(5 PERCENT)**;	

WHERE 술어

LIKE	**SELECT** * **FROM** Sample **WHERE** srt **LIKE** '__ddd**%**';	
	-- str 컬럼에 '두 글자 + 중간에 ddd + 마지막 임의의 문자열'이 있는 문자열	
	<CAUTION> SELECT * **FROM** Sample **WHERE** srt = 'ddd**%**';	특수문자 사용하는 곳에 '=' 사용하면 결과가 없는 것으로 나옴.
IS NULL	**SELECT** * **FROM** Goods **WHERE** price **IS NULL**;	price가 NULL인 column
IS NOT NULL	**SELECT** * **FROM** Goods **WHERE** price **IS NOT NULL**;	price가 NULL이 아닌 column
AND	**SELECT** name, classify **FROM** Goods **WHERE** classfy **!=** '의류' **AND** price >= 1000;	FROM 뒤
OR	**SELECT name, classify** FROM **Goods** WHERE **classfy = '의류'** OR** price >= 1000;	
BETWEEN	**SELECT** * **FROM** Goods **WHERE** price **BETWEEN** 100 **AND** 1000	처음과 끝 포함
IN(=or)	**SELECT** * **FROM** Goods **WHERE** price **IN(**320, 500**)**	320 or 500이 있는 column. NULL 안됨
NOR IN	**SELECT** * **FROM** Goods **WHERE** price **NOT IN(**320, 500**)**	320이나 500이 없는 column. NULL 안됨.
Subscript	SELECT name, Height FROM userTBL **WHERE** height > **(SELECT** height FROM userTBL **WHERE** Name = '김경호**)**;	

WHERE 술어 (cont)

-- ANY	**SELECT** name, height **FROM** userTBL **WHERE** height = **ANY (SELECT** height FROM userTBL **WHERE** add = '경남**')**;
-- ALL	**SELECT** name, height **FROM** userTBL **WHERE** height > **ALL (SELECT** height FROM userTBL **WHERE** add = '경남**')**;
-- IN	**SELECT** name, price **FROM** Goods **WHERE** id **IN(SELECT** id **FROM** Store **WHERE** store_id = '부산');
- EXISTS 술어	**SELECT** name, price FROM Goods AS S **WHERE EXISTS (SELECT** * **FROM** Store AS TS **WHERE** TS.id = '부산' **AND** TS.id = S.id**)**;
	-- EXISTS 인수는 항상 상관 서브쿼리로 작성해야 함.

GROUP BY

Syntex	집약 KEY에 NULL이 포함되면 결과에도 불명(공란)으로 존재한다.	
	SELECT 구에 상수, 집약 함수, GROUP BY에서 사용한 집약 키 외의 열명을 쓸 수 없다. 집약키와 1:1 대응 가능해야 한다.	
	SELECT 구에서 지정한 별명을 사용할 수 없다. 별명 순서가 나중 이므로.	
	결과가 sort된 것이 아니다. order by 사용해야 한다.	
	WHERE 구문에서 집약함수로 조건지은 후 GROUP BY을 사용할 수 없다. - HAVING 사용	
GROUP BY	**SELECT** userID, **SUM**(amount) **AS** '총 구매 개수' **FROM** Goods **GROUP BY** userID;	NULL 값의 개수도 집계됨
	SELECT userID, **SUM**(amount*price) **AS** '총 구매액' **FROM** Goods **GROUP BY** userID;	
	SELECT userID, **SUM**(amount*price) **AS** '총 구매액' **FROM** Goods **WHERE** classify = '의류' **GROUP BY** userID;	
-- ROLLUP	**SELECT** classity, **SUM**(amount*price) **AS** '총 구매액' **FROM** Goods **GROUP BY ROLLUP**(classify);	소합계, 총합계만
	SELECT userID, classify, **SUM**(amount*price) **AS** '총 구매액' **FROM** Goods **GROUP BY ROLLUP**(classify, userID);	

GROUP BY (cont)

	-- classify별 각 userID의 '총 구매액'과 classify 별 소합계, 마지막 총합계
-- CUBE	**SELECT** productName, color, **SUM**(amount) **AS** '수량합계' **FROM** Tbl1 **GROPU BY CUBE**(color, productName);
	-- 제품별 소합계, 총합계, 색상별 소합계
-- GROUPI-NG_ID	**SELECT** groupName, **SUM**(price*amount) **AS** '비용', **GROUPING_ID**(groupName) **AS** '추가행 여부' **FROM** Tbl1 **GROUP BY ROLLUP**(gropuName);
	-- 총합계 표시를 위해 추가된 행에 '1' 표시

HAVING

작성 순서	SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY
Syntex	WHERE 구는 '행'에 대한 조건을 지정, HAVING 구는 '그룹'에 대한 조건을 지정.
	집약 KEY에 대한 조건은 HAVING이 아닌 WHERE 구에 작성한다.
	HAVING 구에는 상수, 집약 함수, GROUP BY에서 사용한 집약 KEY만 사용 가능.
HAVING	**SELECT** classify, COUNT(*) **FROM** Goods **GROUP BY** classify **HAVING** COUNT(*) = 2;
	SELECT classify, AVG(price***1.0**) **FROM** Goods **GROUP BY** classify **HAVING** AVG(price***1.0**) >= 2500;
	SELECT classify, AVG(price***1.0**) **FROM** Goods **WHERE** store = '부산' **GROUP BY** classify **HAVING** AVG(price***1.0**) >= 2500;

ORDER BY

Syntex	SELECT 구문의 가장 마지막에 기술	
	<CAUTION> 성능이 떨어지므로 가급적 사용하지 말 것.	
	소트 KEY에 NULL이 포함되어 있는 경우 제일 처음 또는 제일 마지막에 모아서 표시된다.	
	SELECT 구에 부여한 별명 사용 가능.	
	SELECT 구에 포함되지 않는 열이나 집약합수도 사용 가능. 별명도 가능.	
ORDER BY	SELECT id, name, price FROM Goods **ORDER BY** price **DESC**;	내림차순

ORDER BY (cont)		
	SELECT id, name, price FROM Goods **ORDER BY** price DESC, id;	복수 정렬
	SELECT classfy, COUNT(*) FROM Goods **GROUP BY** classify **ORDER BY COUNT**(*);	집약함수 사용 가능
	SELECT namd **AS nm** FROM Goods **ORDER BY nm**;	별명 가능.
OFFSET / FETCH NEXT	**SELECT** ID, nsme **FROM** userTBL **ORDER BY** birth **OFFST** 4 **ROWS FETCH NEXT** 3 **ROWS ONLY**;	4줄 건너 뛰고 이 후 3줄만 출력

연산자 / 형 변환 / 기타	
컬럼 연산	SELECT name, price * 2 AS "price_x2" FROM Goods;
+=	누적
10%7	나머지 -> 3
<>	SELECT name, prcice FROM Goods WHERE price **<>** 1000;
NOT	SELECT name, prcice FROM Goods WHERE **NOT** price = 1000;
'1-3' < '2'	문자열 대소 비교는 사전식 원칙
IS NULL	SELECT name, price FROM Goods WHERE price IS NULL;
IS NOT NULL	SELECT name, price FROM Goods WHERE price IS NOT NULL;
NULL	NULL을 포함한 계산은 무조건 NULL이 된다.
	NULL/0 도 NULL이 된다.
	비교 연산에서는 NULL 값 행은 반환되지 않는다.
< 형 변환 >	
CAST( )	**SELECT** price, amount, **CAST(CAST**(price **AS** FLOAT)/amount **as** DECIMAL(10,2)) **AS** '단가/수량' **FROM** Tbl1;
	-- 가격을 실수로 바꾼 후 나눠야 정수가 안된다. 이 후 소수점 2자리까지만 표시
CONVERT( )	**SELEC** ANG(**CONVERT**(FLOAT, amount)) **AS** '평균구매개수' **FROM** Tbl1;
STR( )	**SELECT** STR(123);
PARSE( )	**SELECT PARSE**('2019년 9월 9일' **AS** DATE);
TRY_PARSE( )	**SELECT TRY_PARSE**('3.14' **AS** INT);
	-- PARSE( ) 는 형이 틀리면 오류를 발생시키나 TRY_PARSE는 Null을 반환하고 계속 진행한다.
문자 + 문자	'100'+'200' -> '100200'
문자 + 숫자	'100' + 200 -> 300
	'100' + 200.0 -> 300.0
실수 -> 정수 / DECIMAL	소수점 자리수가 잘린다.
숫자 -> 문자	문자 자리수가 작으면 ERROR 발생
< 기타 >	
STATISTICS	SET STATISTICS TIME ON;

변수

syntex	마지막 결과 라인까지 한 번에 실행해야 한다. 한 번 실행되고 없어진다.
DECLARE - SET	**DECLARE @**myVar1 INT, **@**myVar2 DECIMAL(5,2), **@**myVar3 NCHAR(20); **DECLAER @**point INT = 77; **SET @**myVar1 = 5; **SET @**myVar2 = 4.52; **SET @**myVar3 = '가수 이름 => '; **SET @** myVar1 = **GETDATE( )**; **SELECT** @myVar1 = HireDate **FROM** Tbl1 **WHERE** ID = 111; **SELECT @**myVar1 + **@**myVar2; **SELECT @**myVar3, Name **from** Tbl1 **WHERE** height > 180; **SELECT TOP**(**@**myVar1) Name, height **FROM** Tbl1 **ORDER BY** height;
Table 변수	Table 변수로 생성하면 한 번 실행하고 Table이 없어짐. CREATE TABLE # 으로 만든 임시 테이블은 쿼리 창 종료 시 까지는 존재 함.
	DECLARE @tblVar TABLE (id CHAR(8), name NVARCHAR(10), addr NCHAR(2)); INSERT INTO @tblVar SELECT userID, name, addr FROM userTbl WHERE birthYear >= 1970; SELECT * FROM @tblVar;
CREATE TABEL #	CREATE TABLE #tempTbl (id CHAR(8), name NVARCHAR(10), addr NCHAR(2)); INSERT INTO #tempTbl SELECT userID, name, addr FROM userTbl WHERE birthYear >= 1970; SELECT * FROM #tempTbl;

DateTime

GETDATE( )	SELECT GETDATE();	now
SYSDATETIME ( )	SELECT SYSDATETIME();	now(동일)
DATEADD	SELECT DATEADD(day, 100, '2019/01/01');	100 일 더하기
	SELECT DATEADD(hour, 100, '2019/01/01');	100 시간 더하기
DATEDIFF	SELECT DATEDIFF(week, GETDATE(), '2027/10/9');	DayOfWeek
DATEPART	SELECT DATEPART(year, GETDATE());	년도만
MONTH	SELECT YEAR(GETDATE());	년도만(동일)
	-- year / month / week / hour / minute / second	
DATENAME	SELECT DATENAME(weekday, GETDATE());	DayOfWeek
DATEFROMPARTS	SELECT DATEFROMPARTS('2019', '10', '09');	문자열로 날짜 생성
	-- TIMEFROMPARTS() / DATETIME2FROMPARTS()	
EOMONTH	SELECT EOMONTH('2019-01-03');	1월의 마자막 날
	SELECT EOMONTH(GETDATE(), 3);	현재 + 3개월의 마지막 날

수치 / 논리 / Str 연산

< 수치 연산 >		
ROUND	SELECT ROUND(153.246, 2);	소수점 반올림(153.250)
	SELECT ROUND(153.246, -2);	소수점 반올림(200)
RAND	SELECT RAND();	0~1까지의 임의의 수
FLOOR	SELECT FLOOR(3.14);	정수 내림(3)
CEILING	SELECT CEILING(3.14);	정수 올림(4)
ABS	SELECT ABS(-100);	
SQRT	SELECT SQRT(10);	
POWER	SELECT POWER(3, 2);	
< 논리 연산 >		
CHOOSE	SELECT CHOOSE(2, 'a', 'b', 'c', 'd');	
IIF	SELECT IIF(1>2, 'TRUE', 'FALSE');	조건, 참 값, 거짓 값
< String 연산 >		
CHARINDEX	SELECT CHARINDEX('Server', 'SQL Server 2017');	공백포함 시작 위치 -> 5
RIGHT / LEFT	SELECT RIGHT('SQL Server 2017', 4);	오른쪽에서 3 글자 -> '2017'
SUBSTRING	SELECT SUBSTRING('대한민국만세', 3, 2);	3번째부터 2글자 -> '민국'
LEN	SELECT LEN('SQL Server 2017');	공백 포함 글자 수 -> 15
LOWER / UPPER	SELECT LOWER('ABDdef');	모두 소문자로
LTRIM / RTRIM	SELECT LTRIM(' 공백앞뒤두개 ');	왼쪽 공백 제거 -> '공백앞뒤두개 '
REPLACE	SELECT REPLACE('SQL Server 2017', 'Server', '서버');	'SQL 서버 2017'
REPLICATE	SELECT REPLICATE('SQL, 5);	'SQL' 다섯번 반복
REVERSE	SELECT REVERSE('SQL Server 2017');	7102 revreS LQS
SPACE	SELECT SPACE(5);	공백 5개 반환
STR	SELECT STR(123);	'123'
STUFF	SELECT STUFF('SQL 서버 2017', 5, 2, 'Server');	5번째부터 2글자 삭제 후'Server'집어 넣기 -> 'SQL Server 2017'
FORMAT	SELECT FORMAT(GETDATE(), 'dd/MM/yyyy');	'16-09-2017'

집약 함수

집약 함수	복수의 행을 입력받아 하나의 행을 출력
	모든 집약 함수는 계산 전 NULL을 제외시킨다. 단 COUNT(*)만 예외적으로 NULL 포함.
	집약함수는 SELECT, HAVING, ORDER BY에서만 사용 가능하다.

집약 함수 (cont)

	집약함수와 별도의 이름 열을 함께 사용하려면 반드시 GROUP BY와 함께 사용해야 한다.	
COUNT( )	**SELECT COUNT**(*) **FROM** Goods;	NULL 값 포함 모든 행. 단순히 모든 Data 행의 개수 반환.
	SELECT **COUNT**(price) **FROM** Goods;	NULL 값 제외
-- COUNT(DISTINCT...)	**SELECT COUNT**(**DISTINCT** classify) **FROM** Goods;	COUNT 안에 사용해야 함.
COUNT_BIG( )	**SELECT COUNT_BIG**(**DISTINCT** classify) **FROM** Goods;	결과값이 BIGINT 형(21억개 이상인 경우 사용)
SUM( )	**SELECT SUM**(amount) **FROM** Goods;	
	SELECT classify, **SUM**(amount) **AS** '합계' **GROUP BY** classify;	
AVG( )	**SELECT AVG**(price) **FROM** Goods;	NULL 행은 분모 Count에서 제외
	<CAUTION> : 정수형으로 지정된 Data를 평균하면 결과 값도 정수형 소수점 때고 나온다. TRUNC와 같이.	
	SELECT AVG(price***1.0**) **FROM** Goods;	
	SELECT AVR(**CAST**(price **AS DECIMAL(10,6)**)) **AS** '평균가' **FROM** Tbl1;	총 10자리 중 소숫점 6자리로 표시
STDEV( )	**SELECT STDEV**(return) **FROM** Stock;	표준편차
VAR( )	**SELECT VAR**(return) **FROM** Stock;	분산
MAX( ). MIN( )	**SELECT MAX**(date), **MIN**(date) **FROM** Goods;	
	SELECT name, height **FROM** Tbl1 **WHERE** height = **(SELECT MAX**(height) **FROM** Tbl1**)** ;	
	-- Tbl1의 가장 큰 height을 가진 레코드의 name, height	

순위 / 분석 함수

ROW_NUMBER( )	SELECT **ROW_NUMBER( ) OVER** (**ORDER BY** height DESC, name ASC) '키큰순위', name, addr, height FROM Tbl;	1-2-3-4
	SELECT addr, **ROW_NUMBER( ) OVER** (**PARTITION BY** addr **ORDER BY** height DESC, name ASC)[지역별 키큰순위], name, height FROM Tbl;	

순위 / 분석 함수 (cont)

	SELECT top(10) * FROM (SELECT *, ROW_NUMBER( ) OVER(PARTITION BY stockCode ORDER BY logDate DESC, logTime DESC) AS rnum FROM LogMinute) AS l2 WHERE rnum = 1;	
RANK( )	SELECT **RANK( ) OVER** (**ORDER BY** height DESC)[키큰순위], name, addr, height FROM Tbl; 1-2-2-4	1-2-2-4
DENSE_-RANK( )	SELECT **DENSE_RANK( ) OVER** (**ORDER BY** height DESC)[키큰순위], name, addr, height FROM Tbl;	1-2-2-3
NTILE( )	SELECT **NTILE(4) OVER** (**ORDER BY** height DESC)[반번호], name, addr, height FROM Tbl;	키 순으로 4개의 그룹으로 나눔(1,2반은 3명씩)
LEAD( ) OVER	SELECT name, addr, height, height - (**LEAD**(height, 1, 0) **OVER** (**ORDER BY** height DESC)) '다음 사람과 키 차이' FROM Tbl1;	다음 값과의 차이(마지막은 자기 값)
LGR( ) OVER		이전 값과의 차이
FIRST_-VALUE OVER	SELECT addr, name, height, height - (**FIRST_VALUE**(height) **OVER** (**PARTITION BY** addr **ORDER BY** height DESC)) AS '지역별 가장 큰 키와의 차이' FROM Tbl1;	(지역별) 가장 큰 값
CUME_DIST	SELECT addr, name,height, (**CUME_DIST( ) OVER** (**PARTITION BY** addr **ORDER BY** height DESC)) * 100 AS '지역별 누적인원 백분율' FROM Tbl1;	누적함수 동률이면 나중 값
PERCEN-TILE_CONT	SELECT **DISTINCT** addr, **PERCENTILE_CONT(0.5) WITHIN GROUP** (**ORDER BY** height) **OVER** (**PARTITION BY** addr) AS '지역별 키의 중간값' FROM Tbl1;	WTHIN GROUP에 정렬할 열 지정. 적적한 값을 보간하여 표시적적한 값을 보간하여 표시.
PERCEN-TILE_DISC		있는 값 중에서만 추출.
PIVOT	SELECT * FROM pivotTest **PIVOT** (SUM(amount) **FOR** season **IN**('봄', '여름', '가을', '겨울')) AS resultPivot;	

JOIN		
< INNER JOIN >		
INNER JOIN	**SELECT B**.uesrID, **U**.name, **B**.prodName, U.addr **FROM** buyTbl **B**	구매한 사람만 출력
	.....**INNER JOIN** userTbl **U ON B**.userID = **U**.userID **WHERE B**.userid = 'JYP';	중목 안된 컬럼명은 U. 생략 가능
-- DISTINCT	**SELECT DISRINCT U**.uesrID, **U**.name, **U**.addr, **FROM** userTbl **U**	한 번이라도 구매한 사람 목록
	.....**INNER JOIN** buyTbl **B ON U**.userID = **B**.userID **ORDER BY U**.userid;	
-- WHERE EXISTS	**SELECT** U.userID, U.addr **FROM** userTbl U	동일한 결과
	.....**WHERE EXISTS** (**SELECT** * **FROM buyTbl** B WHERE U.**userID =** B.userID);	
< OUTER JOIN >		
-- LEFT OUTER JOIN	**SELECT U**.uesrID, **U**.name, **U**.prodName, U.addr **FROM** userTbl **U**	구매안한 사람도 출력
	.....**LEFT OUTER JOIN** buyTbl **B ON U**.userID = **B**.userID **ORDER BY** U.userID;	userTbl 다 나와.
	.....**WHERE** B.pridName = **NULL**;	한 번도 구매한 적 없는 고객
-- RIGHT OUTER JOIN	.....**FROM** buyTbl B **RIGHT OUTER JOIN** userTbl U **ON**...	테이블 순서 바꾸면 동일한 결과.
	.....**RIGHT OUTER JOIN** buyTbl **B ON U**.userID = **B**.userID **ORDER BY** U.userID;	
-- FULL OUTER JOIN		양쪽 다 나와.
< 다자 조인 >		
-- INNER JOIN	**SELECT S**.stdName, **S**.addr, **C**.clubName, **C**.roomNo **FROM** stdTbl **S**	가입된 사람만.
	.....**INNER JOIN** stdclubTbl **SC ON** S.stdname = SC.stdName	S와 SC를 먼저 JOIN하고
	.....**INNER JOIN** clubTbl **C ON** SC.clubName = C.clubName	그 결과를 다시 C와 JOIN
-- LEFT OUTER JOIN	**SELECT S**.stdName, **S**.addr, **C**.clubName, **C**.roomNo **FROM** stdTbl **S**	
	..... **LEFT OUTERJOIN** stdclubTbl **SC ON** S.stdname = SC.stdName	가입하지 않은 사람도 다 나와
	..... **LEFT OUTERJOIN** clubTbl **C ON** SC.clubName = C.clubName	
-- RIGHT OUTER JOIN	**SELECT S**.stdName, **S**.addr, **C**.clubName, **C**.roomNo **FROM** stdTbl **S**	

JOIN (cont)

	..... **LEFT OUTERJOIN** stdclubTbl **SC ON** S.stdname = SC.stdName	
	..... **RIGHT OUTERJOIN** clubTbl **C ON** SC.clubName = C.clubName	가입한 학생이 없는 동아리도 다 나와.
-- FULL OUTER JOIN	**SELECT S**.stdName, **S**.addr, **C**.clubName, **C**.roomNo **FROM** stdTbl **S**	
	..... **FULL T OUTERJOIN** stdclubTbl **SC ON** S.stdname = SC.stdName	가입하지 않은 학생도 나오고
	..... **FULL OUTERJOIN** clubTbl **C ON** SC.clubName = C.clubName	가입한 학생이 없는 동아리도 나와
CROSS JON	**SELECT** * **FROM** buyTbl **CROSS JOIN** userTbl;	
	-- 'ON' 없음. 카티션 곱. 대량 샘플 데이터 만들 대 사용.	
SELF JOIN	SELECT A.emp AS '부하직원', B.emp AS '직속상관', B.department AS '직속상관부서' FROM enpTbl A INNER JOIN empTbl B ON A.manager = B.emp WHERE A.emp = '우대리';	
< 결과 합하기/ 제외하기 >		
UNION (ALL)	**SELECT** stdName, addr **FROM** stdTbl **UNION SELECT** clubName, roomNo **FROM** clubTbl;	
	-- 두 결과 중복 포함 하래로 합침, UNION은 중복 제거	
EXCEPT	**SELECT** name, mobile1+mobile2 AS '전화번호' **FROM** userTbl **EXCEPT SELECT** name, mobile1+mobile2 **FROM** userTbl **WHERE** mobile1 **IS NULL**;	첫 번째 쿼리 결과 중 두 번째 결가 제거
INTERSECT	**SELECT** name, mobile1+mobile2 AS '전화번호' **FROM** userTbl **INTERSECT SELECT** name, mobile1+mobile2 **FROM** userTbl **WHERE** mobile1 **IS NULL**;	첫 번째 쿼리 결과 중 두 번재 쿼리 해당되는 것만.

프로그래밍

CASE WHEN - THEN

```
DECLARE @credit CHAR(1), @point INT = 77
SET @credit =
  CASE
    WHEN (@point >= 90) THEN 'A'
    WHEN (@point >=80) THEN 'B'
    ELSE 'C'
  END
PRINT N'학점 : ' + @credit
```

ELSE 는 생략 가능. END는 생략 불가.

프로그래밍 (cont)

--간편 CASE

```
DECLARE @credit CHAR(1), @point INT = 77
SET @credit =
  CASE @point
    WHEN 90 THEN 'A'
    WHEN 80 THEN 'B'
    ELSE 'C'
  END PRINT N'학점 : ' + @credit
```

행열 변환

```
SELECT SUM(CASE WHEN classfy = '의류' THEN price ELSE 0) AS sum_price_close,
    SUM(CASE WHEN classfy = '주방용품' THEN price ELSE 0) AS sum_price_kitchen
    SUM(CASE WHEN classfy = '사무용품' THEN price ELSE 0) AS sum_price_office
  FROM Goods;
```

IF - ELSE

```
IF @var1 = 100 PRINT '100'
ELSE PRINT 'Not 100'
```

실행문 여러개면 begin - end

WHILE

```
DECLARE @i INT = 1, @hap BIGINT = 0
WHILE (@i < 100)
BEGIN
  IF (@i%7 = 0)
  BEGIN
    SET @i += 1
    CONTINUE
  END
  SET @hap += @i
  IF (@hap>1000) BREAK
  SET @i += 1
END
PRINT N'합계 : ' + CAST(@hap AS NCHAR(10))
```

-- CONTINUE / BREAK

GOTO

```
IF (@hap > 1000) GOTO endpoint
endpoint:
PRINT N'합계= ' + CAST(@hap AS NCHAR(10))
```

문자 + 숫자는 형변환 후 사용

WAIT FOR DELAY

```
WAIT FOR DELAY '00:00:05'
```

5초 딜레이

WAIT FOR TIME

```
WAIT FOR TIME '23:59'
```

23:55까지 정지

프로그래밍 (cont)

TRY / CATCH	**BEGIN TRY** 원래 SQL문 **END TRY** **BEGIN CATCH** 오류 시 SQL문 **END CATCH**
-- ERROR_LINE	**BEGIN TRY** INSERT INTO userTbl VLAUES('LSG', '이상구') **END TRY** **BEGIN CATCH** PRINT **ERROR_LINE**( ) **END CATCH**
	-- ERROR_MESSAGE( ) / ERROR_NUM( ) / ERROR_PROCEDURE( )
RAISERROR	**RAISERROR**(N'에러 발생!!', 5, 1); 수준 5, 상태 1로 Error message 표시
THROW	**THORW** 50000, N'에러 발생!!', 1 Error 번호 50000, 상태 1로 Error 표시. 50000이상을 적어줘야 함.
EXEC( )	**DECLARE** @curDATE DATE, @curMonth VCHAR(2), @curDay VCHAR(2), @sql VCHAR(100) **SET** @curDATE = GERDTE( ) **SET** @curMonth = MONTH(@curDATE) **SET** @curDay = DAY(@curDATE) **SET** @sql = 'CREATE TABLE myTbl' + @curMonth + '_' + @curDay **SET** @sql **+=** '(id INT, name NCHAR(10))' **EXEC**(@sql)

Procedure

Syntex	처음 실행 후 메모리에 저장되어 이후 반복 사용할 경우 실행 속도가 빨라진다.
	Python 에서 불러 사용하기 편하고 수정, 삭제 등 관리가 편해진다.
	저장 프로시저는 SQL 서버에 저장되므로 클라이언트에서 실행하는 것보다 네트워크 전송량이 감소한다.
	본문 안에 BEGIN - END 없이 사용한다.
CREATE PROCEDURE	CREATE PROCEDURE usp_user1 @userBirth INT, @userHeight INT = 178 AS SELECT * FROM userTbl WHERE birthYear > @userBirth AND height > @userHeight; EXEC usp_user2 1970, 178; EXEC usp_user2 @userHeight=178, @userBirth=1970; EXEC usp_user1 1960, 170; EXEC usp_user1 1680;
	CREATE #usp_temp AS SELECT * FROM userTbl; -- 임시 프로시저

Procedure (cont)

Output

```
CREATE PROCEDURE usp_user2 @textValue NCHAR(10), @outValue INT OUTPUT AS
    INSERT INTO testTbl VALUES (@textValue);
    SELECT @outValue = IDENT_CURRENT('testTbl');
CREATE TABLE testTbl (id INT IDENTITY, txt NCHAR(10));
DECLARE @myValue INT;
EXEC usp_user2 '테스트 값1', @myValue OUTPUT;
PRINT '현재 입력된 id 값 ==> ' + CAST(@myValue AS CHAR(5));
```

RETURN

```
CREATE PROCEDURE usp_return @userName NVARCHAR(10) AS
    DECLARE @userID CHAR(8);
    SELECT @userID = userID FROM userTbl WHERE name = @userName;
    IF @userID <> RETURN 0;
    ELSE RETURN -1;
DECLARE @retVal INT
EXEC @retVal = usp_return '나몰라';
SELECT @retVal;
```

Table Type

```
CREATE TYPE userTblType AS TABLE (userID CHAR(8), name NVARCHAR(10), birthYear INT, addr NCHAR(2));
CREATE PROCEDURE usp_tableTypeParam @tblParam userTblType READONLY AS    -- 테이블 형식 매개 변수는 READONLU 필수
    SELECT * FROM @tblParam WHERE birthYear < 1970;
DECLARE @tblVar userTblType;
INSERT INTO @tblVar SELECT userID, name, birthYear, addr FROM userTbl;
EXEC usp_tableTypeParam @tblVar;
```

쿼리 내용 확인

```
EXEC sp_helptext usp_user1;
```

```
SELECT o.name, m.definition FROM sys.sql_modules m JOIN sys.objects o ON m.object_id = o.object_id AND o.TYPE = 'P';
```

Function

Syntex

함수 내부에 TRY - CATCH / CREATE / ALTER / DROP 사용 못함

오류가 발생하면 즉시 함수 실행을 몸추고 값을 반환하지 않는다.

BEGIN - END 를 기본 사용

CREATE FUNCTION

```
CREATE FUNCTION ufn_getAge(@bYear INT)
    RETURNS INT AS    -- RETURNS에서 'S' 주의
BEGIN
    DECLARE @age INT;
    SET @age = YEAR(GETDATE()) - @bYear;
    RETURN @age;
END
```

Function (cont)	
	SELECT dbo.ufn_getAge(1979);--주의 : Schema 이름 적어야 함.
	SELECT userID, name, dbo.ufn_getAge(birthYear) AS [만 나이] FROM userTbl;
ALTER FUNCTION	ALTER FUNCTION ufn_getAge(@bYear INT) RETURNS INT AS BEGIN DECLARE @age INT; SET @age = YEAR(GETDATE()) - @bYear + 1; RETURN @age; END
DROP FUNCTION	DROP FUNCTION ufn_getAge;
WITH SCHEMABINDING	-- 참조하고 있는 테이블이나 뷰 등을 수정하지 못하게 함.
Inline Table 반환	CREATE FUNCTION ufn_getUser(@ht INT) RETURNS TABLE WITH SCHEMABINDING -- 여기에서 참조되는 Table의 컬럼 변경이 안된다. AS RETURN (SELECT userID, name, height FROM dbo.userTbl WHERE height > @ht) -- SCHEMABINDIND을 사용하려면 Table명에 Schema 이름도 붙여줘야 함. SELECT * FROM dbo.ufn_getUser(177);

Function (cont)

다중문 테이블 반환

```
CREATE FUNCTION ufn_userGrade(@bYear INT)
   RETURNS @retTable TABLE (userID CHAR(8), name NCHAR(10), grade NCHAR(5)) AS
BEGIN
   DECLARE @rowCnt INT;
   SELECT @rowCnt = COUNT(*) FROM userTbl WHERE birthYear >= @bYear;

   -- 행이 하나도 없으면 '없음'을 입력하고 테이블 리턴하고 끝남.
   IF @rowCnt <= 0
   BEGIN
      INSERT INTO @retTable VALUES ('없음', '없음', '없음');
      RETURN;
   END;

   -- 행이 1개 이상이면 아래 수행
   INSERT INTO @retTable
      SELECT U.userID, U.name,
         CASE
            WHEN (SUM(price*amount) >= 1500) THEN '최우수고객'
            WHEN (SUM(price*amount) >= 1000) THEN '우수고객'
            WHEN (SUM(price*amount) > 1) THEN '일반고객'
            ELSE '유령고객'
         END
      FROM buyTbl B RIGHT OUTER JOIN userTbl U ON B.userID = U.userID
      WHERE birthYear >= @bYear
      GROUP BY U.userID, U.name;
    RETURN;
END;
SELECT * FROM dbo.ufn_userGrade(1970);
```

Cussor

커서 선언(DECLARE) - 커서 열기(OPEN) - 커서에서 데이터 가져오기(FETCH) - 데이터 처리 - 커서 닫기(CLOSE) - 커서 해제(DEALLOCATE)