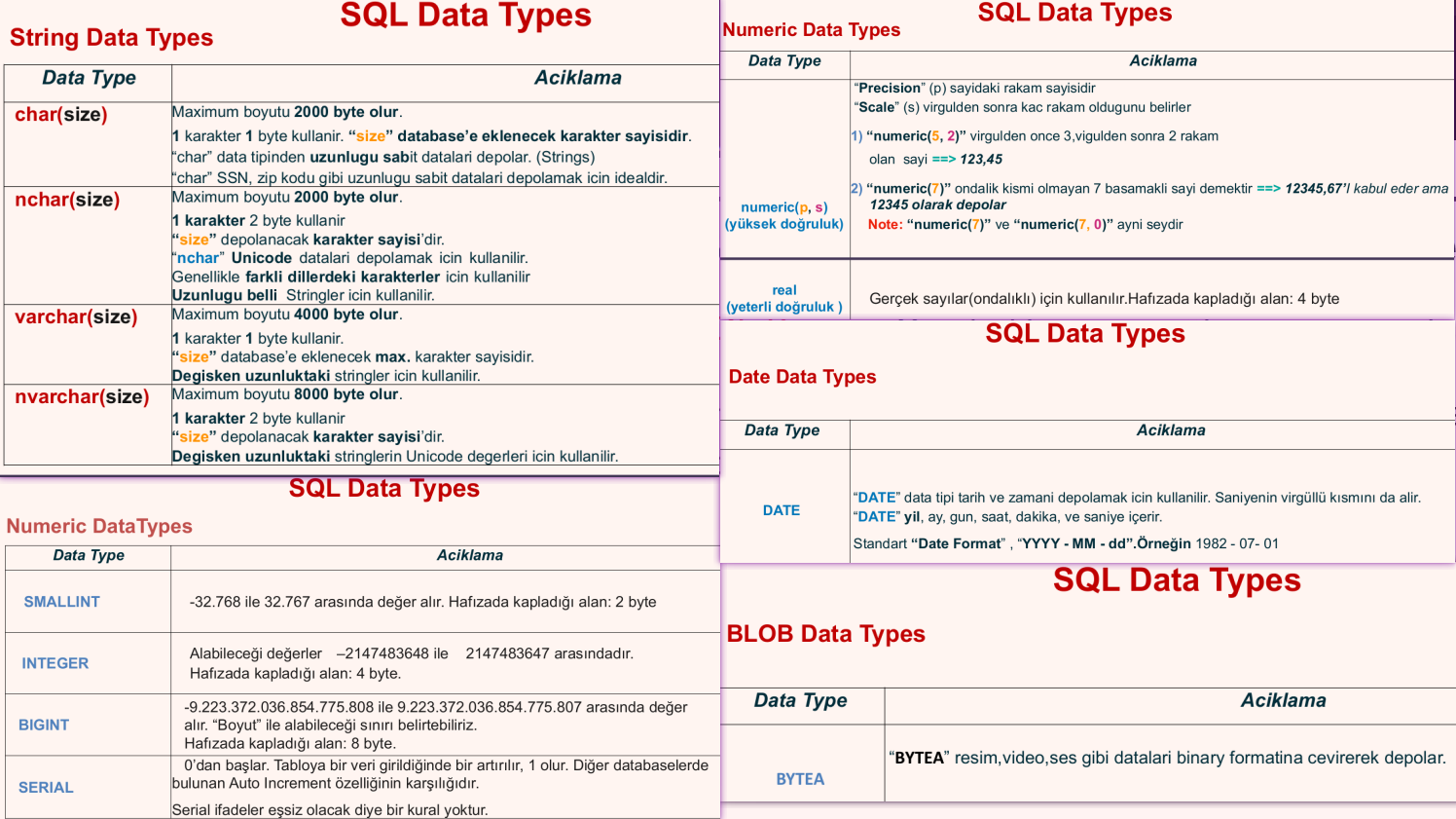
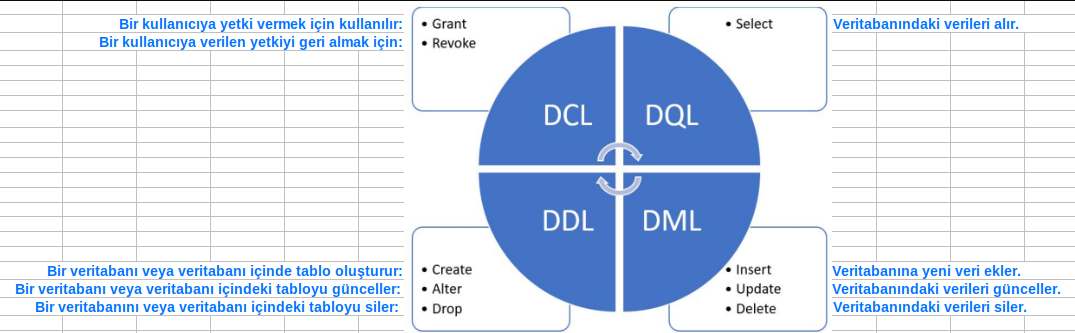
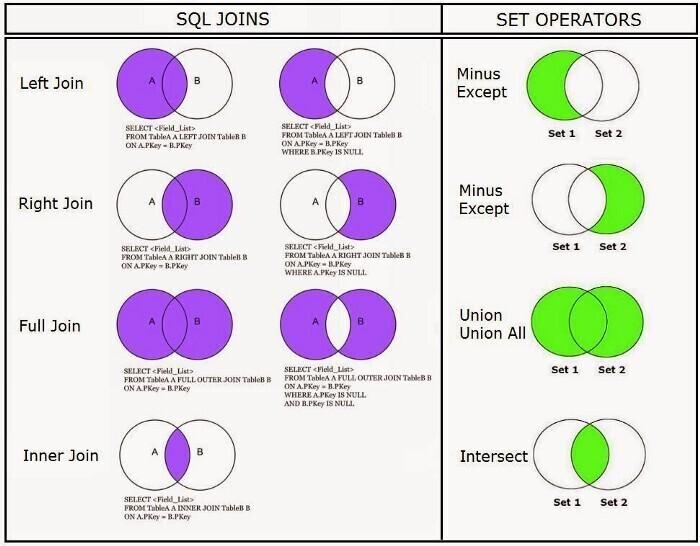
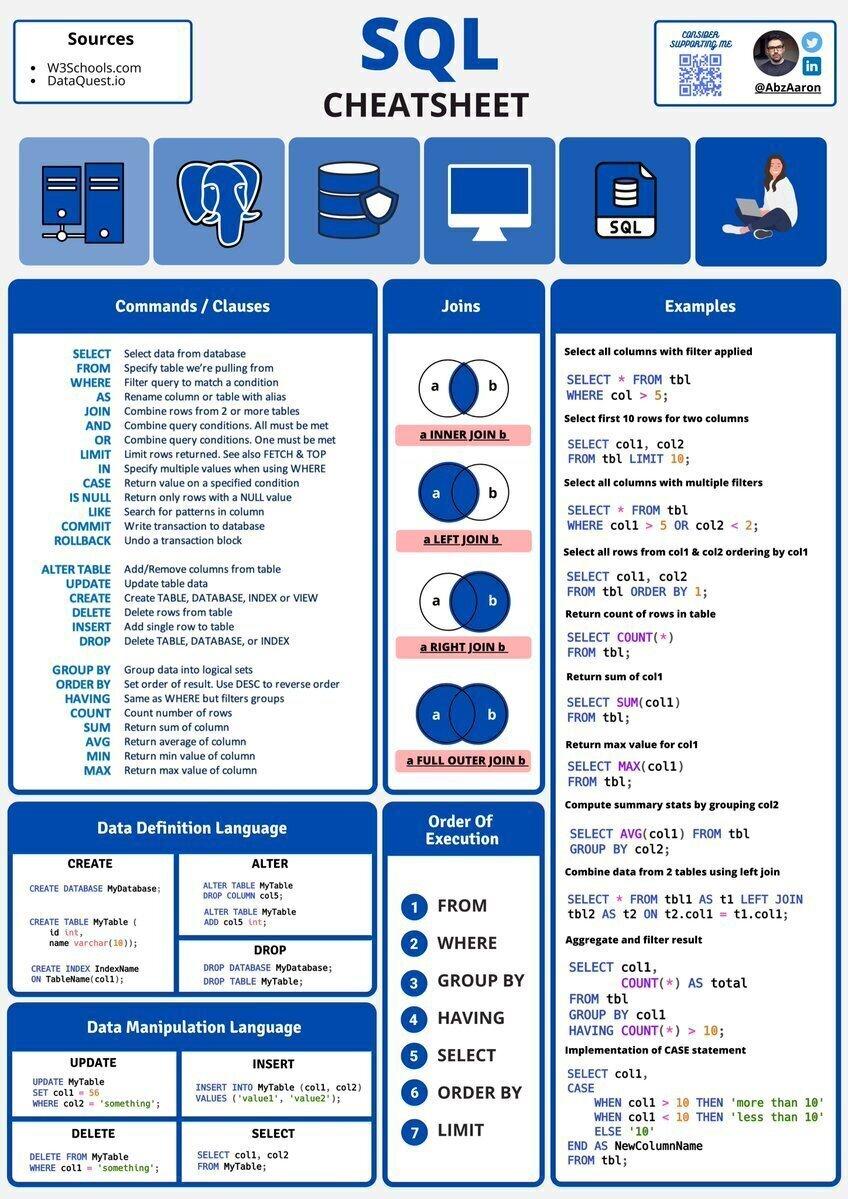
# SQL\_OZET TABLOLAR









ENGLISH

# **SQL\_REMINDER\_NOTES-ENGLISH**

* **SQL:** Structured Query Language isn’t a programing language. It is a query language
* If you send data to database, you can validate date;
* by using search functionality
* by using SQL codes
* by using API codes
* **DBMS:** Data Base Management System. It provides;
* To access database,
* To **C**reate, **R**ead, **U**pdate, **D**elete, **(CRUD)**
* To get reports form database,
* To control access to the database, (Security)
* To interact with other applications
* **Relational Database (SQL Database):** To store date in tabels.
* **Schema:** A relationship between tables and field
* **Non-relational database:** It does not use the tabular schema. Data can be stored in a single file

**Primary Key**

* Primary key can be anything, a number, String, character etc.
* Primary keys must contain UNIQUE values,
* It cannot contain NULL values
* A table can have only one primary key
* Natural Primary Key: uses real values for primary key (SSN, e-mail address etc.)
* Surrogate Primary Key: uses just numbers

**Foreign Key**

* It is used to create link between two tables.
* It can has NULL values
* It can has repeated values
* A table can have multiple Foreign Keys
* It can create a relation between the table and the table itself.

**Composite Key** 

* It’s mean is the use of multiple columns together to create uniqueness.
* when it taken individually it does not guarantee uniqueness.

**Unique Key**

* A table can have more than one unique key
* It allows Null values
* It does not allow duplication except null

**SQL is the combination of 4 different languages**

* **Data Query Language ( DQL):** It is used to query the database for information. **R**ead
* **SELECT:** Fetch the data from the database(READ)
* **Data Definition Language ( DDL):** It deals with descriptions of the database schema (tables, records, fields) and is used to create and modify the structure of database objects
* **DROP:** Deletes the table
* **ALTER:** Updates the table
* **CREATE:** Creates the table
* **Data Manipulation Language ( DML):** It deals with the manipulation of data present in the database. **C**reate, **U**pdate and **D**elete
* **INSERT:** Adds data to database(CREATE)
* **DELETE:** Deletes the data from the database(Delete)
* **UPDATE:** Updates the data in the database(Update)
* **Data Control Language ( DCL):** It is used to control privileges in Database. DCL manages users and permissions
* **GRANT:** Used to authorize the user
* **REVOKE:** Used to revoke the priviledge users
* While we create a table, we can put some directive rules for fields (UNIQUE, NOT NULL, PRIMARY KEY etc) with 2 ways;
* student\_name VARCHAR(50) **NOT NULL,**
* **CONSTRAINT** student\_id**\_pk** **PRIMARY KEY** (student\_id)

**DATA TYPES**

* **VARCHAR(***size***)** defines the field value as a string. Size is the maximum number of characters to store
* **DATE** defines the field value as a date and time. The format is ‘dd - MMM – yy’. We can use abbreviation of mounts, like ‘09 - Aug – 21’
* **SMALLINT** defines the field value as a integer. The range is between -37, 767 to 32, 767
* **NUMERIC** defines the field value as a integer. But range is more than smallint.
* **CREATE TABLE** *students* **(…);** provides to create a table with the given parameter inside the parentheses.
* Also we can create a new table from using existing table with using **AS SELECT…**,… and **FROM…** key Word

**CREATE TABLE** studentsIdName

**AS**

**SELECT** id**,** name

**FROM** students**;**

* **…\_pk PRIMARY KEY (…) :** makes the field primary key in the table
* **…\_fk FOREIGN KEY (….) REFERENCES (…) :** makes the field foreign key in child table

**CONSTRAINT** student\_id**\_fk FOREIGN KEY** (student\_id) **REFERENCES** parents(student\_id)

* **…\_check CHECK(…),** allows to add constrait to a field

**CONSTRAINT** student\_age**\_check CHECK(**student\_age **BETWEEN** 15 **AND** 20**),**

**CONSTRAINT** student\_name**\_upper\_case CHECK (**student\_name **= upper(**student\_name**))**



**INSERT**

* **INSERT…** allows to add data to the table with 2 ways. Whole record or specific fields

**INSERT INTO** students **VALUES(**103, 'AYSE HAN', 15, '29-Feb-2008'**);** // whole record

**INSERT INTO** students**(student\_name, student\_age, student\_id)** **VALUES(**'MARK TWAIN', 16, 105**);** // with choosing specific fields

* **NOTE:** We have to follow orders of the fields in the table to insert a data
* For varchar we have to use single quotes ==> '….'

**DROP**

* **DROP TABLE** *students***;** deletes the specific table.

**NOTE:** You cannot drop “Parent Table” without dropping the “Child Table”

**SELECT**

* **SELECT \* FROM** *students***;** allows to see the given table
* **SELECT** *name* **FROM** *workers***;**  gives all *name* fields in *workers* table.
* **SELECT…** choose and shows desired value with **MAX-MIN-AVG** (etc.) and **FROM** key word

**SELECT MAX(**student\_age**) FROM** students**;**

**UPDATE**

* **UPDATE…** is used for changing data in the table with **SET** and **WHERE** key words. We can update multiple cell, multiple records. If WHERE key word isn’t used, all records are updated.

**UPDATE** students

**SET** student\_dob**= '**11-Dec-1997**',**

student\_address**=** ‘Ankara’

**WHERE** student\_name**='**ALI CAN**';** // new value for Ali Can is 11-Dec-1997

**UPDATE** students

**SET** student\_dob**='**01-Aug-2021**',**

student\_address**=** ‘Ankara’

**WHERE** student\_id **<**104**;** // new value for 101,102,103 is 01-Aug-2021'

**DELETE**

* **DELETE…** erases record/records from the table with **FROM** and **WHERE** key word. If **WHERE** key word isn’t used, all records are deleted.

**DELETE** **FROM** people

**WHERE** ssn **>**123456789 **AND** ssn**<**345678901**;** // All records are deleted which are between the given range

* **TRUNCATE…** deletes all records irreversibly but table is not deleted. WHERE keyword can not be used.

**TRUNCATE TABLE** people**;**



**WHERE**

* With using **WHERE**, we can get, delete one or more record’s fields.

| **SELECT** *name* **FROM** *workers*  **WHERE** *salary***=** '12000'**;** | **SELECT** \* **FROM** *workers*  **WHERE** *id***>**10002**;** |
| --- | --- |

* We can get the fields according to a condition

| **SELECT** *name* **FROM** *students*  **WHERE** *exam\_grade* **> 85 AND** *address***=** *'Ankara'***;** | **SELECT \* FROM** *students*  **WHERE** *exam\_grade* **IN(***70,95***);** |
| --- | --- |

* **IN(…)** ==> the fields which provides one of conditions in the parentheses. Boundries inclusive. Can be used in in a SELECT, INSERT, UPDATE, or DELETE

**SELECT** *name* **FROM** *students*

**WHERE** exam\_grade **IN(**70,95**); <==> WHERE** exam\_grade**=**70 **OR** exam\_grade**=**85**;**

// DRY code // // WET code //

* **BETWEEN** ==>the fields between given conditions in the parentheses. Boundries inclusive

**DELETE FROM** *students*

**WHERE** exam\_grade **BETWEEN** 75 **AND** 95**;**

* We also have **NOT IN , NOT BETWEEN** key words. Usage’s are esame.
* **DRY code:** **D**on’t **R**epeat **Y**ourself
* **WET code:** **W**rite **E**verything **T**wice (Not Recommended)
* **AS** keyword is used to create a temporary field from the table

**SELECT AVG(**salary**) AS** average\_salary

**FROM** workers**;** //gives average of salary fields under the average\_salary heading

* **CONCAT** is used for concatination with more than one string or non-string. Also || operator can be used

**SELECT** employee\_id **AS** id**,** first\_name **|| ' ' ||** last\_name **AS** full\_name

**FROM** employees**;**

**SELECT** employee\_id **AS** id**, CONCAT(**first\_name,' ', last\_name**) AS** full\_name

**FROM** employees**;**

* **COUNT** returns the number of the requested column. Usage is same with SUM
* **SUM** returns the total value of the requested column.

**SELECT SUM (**salary**)**

**FROM** workers**;**

* **Sub-query:** means using nested query. That is embedded in WHERE clause
* can be used with SELECT, UPDATE, INSERT, DELETE
* Subquery must be enclosed in parentheses.
* If mainquery has only one column subquery can has just one column. If mainquery has two column, subquery can has two column.
* ORDER BY cannot be used in subquery

**SELECT MAX(***salary***) AS** *third\_highest\_salary*

**FROM** *workers*

**WHERE** *salary***< (SELECT MAX(***salary***)**

**FROM** *workers*

**WHERE** *salary***< (SELECT MAX(***salary***) FROM** *workers***));**

* **ORDER BY** sorts the table according to given statement. **ASC** is default. If descending order is desired, **DESC** keyword must be used.

**SELECT \* FROM** workers

**ORDER BY** salary **DESC** // NOTE: We can use field name or field number

* **OFFSET** skips as many lines as the given number and returns the others. It has to be used with **ORDER BY.** It can be used with **FETCH** for limiting number of returned lines
* **FETCH** returns as many lines as the given number with using **FIRST** or **NEXT** keyword. It has to be used with **ORDER BY**

**SELECT \* FROM** workers

**ORDER BY** salary **DESC**

**OFFSET** 2 **ROWS**

**FETCH NEXT** 1 **ROW ONLY;**

* **EXISTS** checks if records exist according to the given expression. It is used with sub-query. It can be used with SELECT, UPDATE, INSERT, DELETE commands. We also have the **NOT EXISTS** keyword

**UPDATE** *customers\_likes*

**SET** *customer\_name***=** *'No Name'*

**WHERE EXISTS (SELECT** *liked\_product* **FROM** *customers\_likes* **WHERE** *customer\_name***=** *'Lary'***);**

// if Lary exist in the table, updating operation will be done for all customer\_name

* **LIKE** checks a specified pattern in a field with using **WHERE** key Word. There is also **NOT LIKE.** It can be used with OR, AND keywords. It is used in SELECT, INSERT, UPDATE, or DELETE

**SELECT** *name* **FROM** *employees*

**WHERE** *name* **LIKE** *'%u%r%'* **OR** *name* **LIKE** *'%r%u%'***;**



**WILDCARD**

* WildCards are special characters to handle search activities. WHERE and LIKE are used together
* **%**  It represents zero or more characters
* ‘*E*%’ ==> **starts** with E
* ‘%*e’* ==> **ends** with e
* '%*e*%' ==> has e **in any order**
* '%*e*%*n*%' **OR** *…..* **LIKE** '%*n*%*e*%' ==> has e and n in any order
* **\_** It represents a single character for wildcard
* '\_*e*\_*n*%' ==> second character is e fourth is n
* '%*e*\_' ==> last second character is e
* '\_*e*\_\_\_\_%' ==> has at least 6 character and second one is e
* '\_\_%*e*%' ==> has e in any order after 2 character

**REGEXP\_LIKE**

* We can use regular expressions in SQL

| **SELECT** *word* **FROM** *words*  **WHERE REGEXP\_LIKE(***word***,** *'h[oai](.\*)t'***);** | **SELECT w***ord* **FROM** *words*  **WHERE** *word* **~** *'h[oai](.\*)t'***;** |
| --- | --- |

* . is one single character for regexp
* \* means to repeat whatever came before it any number of times
* ^ means first character.
* [^] means different from that character
* $ means last character
* '*h*[*oai*](.\*)*t*' ==> first h, second o,a or i, then more character, last one is t
* '*h*[*a-e*](.\*)*t*' ==> first h, second between a-e, then more character, last one is t
* '^[*say*](.\*)' ==> first s,a or y, then more character
* '(.\*)[*maf*]$' ==> after any number of characte last one should be m,a or f
* '^[*d-t*].*l*' ==> first between d-t, second can be any character, third is l
* '^[^E]' ==> first character should be different from E
* '[^nx]..$' ==> third letter from the end should be different from n and x
* ‘..[ox]’ ==> third character o or x

**ALIASES**

* It is used to assign a temporary name a field or a table
* AS keyword is optional. We can use for more readablity.
* If we want to use for temporary field name, we should write the name in “ ”

**SELECT** e.employee\_first\_name **AS** first\_name**,** a.state **AS** "Employee state"

**FROM** employees e, addresses a

**WHERE** e.employee\_id=a.employee\_id**;**

-- We assigned e=employees / a=addresses for this query



**GROUP BY**

* It devides the returning rows to the groups
* We can apply aggregate function to calculate
* GROUP BY clause should be used after the FROM and WHERE clauses and before the HAVING SELECT, DISTINCT, ORDER BY and LIMIT clauses.

**SELECT** company**, MIN(**salary**) AS** min\_salary**, MAX(**salary**)** max\_salary

**FROM** workers

**GROUP BY** company

**ORDER BY** min\_salary**;**



**HAVING**

* It is used for filtering. Can only be used with aggregate function
* WHERE keyword can not be used with aggregate function
* We cannot use WHERE after GROUP BY because of that we should use HAVING keyword

**SELECT** state**, MAX(**salary**) FROM** workers

**GROUP BY** state

**HAVING MAX(**salary**)<** 3000**;**



**UNION**

| * It is used to join the result of more than 1 queries * Operator returns unique records. It removes the repeated records. * By using UNION operator we are able to put different fields into a single field. * When we put diffent fields into a sinlge field, data types must be same. | **SELECT** state **AS** "state/name"**,** salary  **FROM** workers  **WHERE** salary >3000  **UNION**  **SELECT** name,salary  **FROM** workers  **WHERE** salary <2000**;** |
| --- | --- |

* **UNION ALL** is similar with **UNION**. But it returns repated values as well

| **INTERSECT** returns unique an common (intersection, joint) records  **SELECT** name  **FROM** workers  **WHERE** salary> 1000  **INTERSECT**  **SELECT** name  **FROM** workers  **WHERE** salary< 2000**;**  Returns 2 statement’s intersection | **EXCEPT** is used to extract one query result from another query result. It returns unique records  **SELECT** name  **FROM** workers  **WHERE** salary<3000    **EXCEPT**    **SELECT** name  **FROM** workers  **WHERE** company='GOOGLE'**;**  Gets names which are their salary less than 3000 and extracts the GOOGLE’’s employees. |
| --- | --- |

**JOINS**

| **INNER JOIN** returns the common records of two tables | **LEFT JOIN** returns all rows from the LEFT-Hand table |
| --- | --- |
| **RIGHT JOIN** returns all rows from the RIGHT-Hand table | **FULL JOIN** returns all rows from both table with  nulls in place where the join condition is not met. |

* Usages are same. Just keyword should be different

**SELECT** companies.company\_name**,** orders.order\_id

**FROM** companies **INNER JOIN** orders

**ON** companies.company\_id **=** orders.company\_id**;**

* **SELF JOIN** allows to use a single table as two table keyword is same with **INNER JOIN**

**SELECT** employee.name **AS** manager\_name**,** manager.name **AS** employee

**FROM** workers employee **INNER JOIN** workers manager

**ON** employee.id**=**manager.manager\_id**;**



**ALTER**

* **ALTER TABLE** is used to add, modify, delete columns, add-drop constraints and rename the table
* Keywords are **ADD, DROP, COLUMN, RENAME, TO, SET**

| Add field  **ALTER TABLE** workers  **ADD** number\_of\_workers **SMALLINT DEFAULT 0,**  **ADD name\_of\_ceo VARCHAR(50);** | Delete Field  **ALTER TABLE** workers  **DROP COLUMN** name\_of\_ceo**;** |
| --- | --- |
| Rename field  **ALTER TABLE** workers  **RENAME COLUMN** title **TO** position**;** | Rename table  **ALTER TABLE** workers  **RENAME TO** employees**;** |
| Add constraint  **ALTER TABLE** employees  **ADD CONSTRAINT** id **UNIQUE(**id**);** | Modify field  **ALTER TABLE** employees  **ALTER COLUMN** position **TYPE CHAR(**7**);** |

* If NOT-NULL constraint will added, sytnax is different from others

**ALTER TABLE** employees

**ADD COLUMN** number\_of\_workers **SET NOT NULL;**



**FUNCTION CREATING**

* A Function is created to do some tasks faster.
* It can be created for CRUD operations.
* In SQL, every function returns a data as return type.
* Operations that does not return data as return type are called 'Procedure'
* For creating a funtion;
* Use CREATE FUNCTION keyword
* OR REPLACE deletes the old method if it is exist with same name and parameters
* method\_nameame (parameter\_name DATA TYPE)
* RETURNS Data\_Type
* LANGUAGE: can be sql, c, internal, or the name of a user-defined procedural language, e.g., plpgsql.
* Type method body in dolar signs ==> $$ … $$

**CREATE OR REPLACE FUNCTION** volume\_of\_cone**(**r NUMERIC, h NUMERIC**)**

**RETURNS** NUMERIC

**LANGUAGE** plpgsql

**AS**

**$$**

**BEGIN**

**RETURN** r\*r\*h\*3.14/3**;**

**END**

**$$**

**SELECT \* FROM** volume\_of\_cone(5,6); ==> returns Numeric 157.0000000000000000

**SELECT to\_char(**volume\_of\_cone(5,6)**,** '000.0'**);** -- arrenges return type as 000.0 ==> 157.0



**FOR INTERVIEW**

* What are the Aggregate Functions in SQL?
* SUM, COUNT, MAX, MIN, AVG
* Interview Question: Find the second highest salary?

**SELECT MAX(***salary***) AS** *second\_highest\_salary*

**FROM** *workers*

**WHERE** *salary***< (SELECT MAX(***salary***) FROM** *workers***);**

/\***STRING MANIPULATION FONKSIYONLARI**

**TRIM(**Sütun parametresi) string ifadenin başındaki ve sonundaki boşluk karakterlerini siler

**LTRIM** sadece soldan boşluk siler

**RTRIM** sadece sağdan boşluk siler

**REPLACE**(sütun,'değişecek ifade','yeni ifade') belirtilen ifadeyi o sütunda bulursa, belirlediğiniz yeni ifade ile değiştirir

bir değişiklik yaptığı için UPDATE ile kullanılır

**CONCAT**(sütunadı, 'string') iki veya daha fazla dizeyi birleştirme işlemi için kullanılır

**SUBSTRING**(sütun adı,başlangıç index,bitiş index) o sütunda belirtilen index aralığındaki string değerini verir

istediğimiz bir sütunda stringlerin istediğimiz kısımlarını değiştirebiliriz. Tek index ilede çalışır belirtilen kısımdan

en sona kadar alır

**UPPER** (sütun adı) o sütundaki strng değerleri büyük harfle getirir

**LOWER** (sütun adı) o sütundaki strn değerleri küçük harfle getirir

**INITCAP**(sütun adı) o sütundaki her bir satırdaki stringlerin baş harfini büyük harf yapar

**LENGTH**(sütun adı) belirtilen sütundaki her satır içinde bulunan stringlerin uzunluklarını alır

**REVERSE**(sütun adı) sütunu ters çevirir

**LEFT()** / **RIGHT()** (Başlangıç ve Son Karakterler): Bir dizinin başlangıç veya sonundaki belirli karakterleri alır.

**LEFT**(sütun, 3) ilk 3 karakteri alır RIGHT olsa son 3 karakteri alırdı

--**FETCH NEXT n ROW ONLY**: sadece sıradaki ilk n satırı getirir

-- **LIMIT n** : sadece sıradaki ilk n satırı getirir

-- **OFFSET n** : n satırı atlayıp sonrakileri getirir

\*/

# **SQL-BATCH189-DERS NOTLARI**

İÇİNDEKİLER TABLOSU

[**DAY 1 3**](#_heading=h.gjdgxs)

[PRIMARY KEY 4](#_heading=h.30j0zll)

[FOREIGN KEY 4](#_heading=h.1fob9te)

[**DAY 2 4**](#_heading=h.3znysh7)

[VARİABLE’LAR : STRİNG DATA TYPES 4](#_heading=h.2et92p0)

[NUMERİC DATA TYPES 5](#_heading=h.3dy6vkm)

[DATE DATA TYPES 5](#_heading=h.1t3h5sf)

[BLOB DATA TYPES 5](#_heading=h.4d34og8)

[TABLO OLUŞTURMA 5](#_heading=h.2s8eyo1)

[Varolan bir tablodan yeni tablo üretmek 6](#_heading=h.17dp8vu)

[**DAY 3 6**](#_heading=h.3rdcrjn)

[Primary Key oluşturmanın birinci yolu 6](#_heading=h.26in1rg)

[Primary Key oluşturmanın ikinci yolu 6](#_heading=h.lnxbz9)

[Birden fazla sutun kullanarak Primary Key oluşturma 7](#_heading=h.35nkun2)

[Foreign Key oluşturmak 7](#_heading=h.1ksv4uv)

[TABLOYA VERİ GİRİŞİ YAPMA 8](#_heading=h.44sinio)

[INSERT INTO: içine yerleştir 8](#_heading=h.2jxsxqh)

[Bir tablonun belli field'larına data eklemek 8](#_heading=h.z337ya)

[TÜM TABLOYU KONSOLA YAZDIRMA 8](#_heading=h.3j2qqm3)

[INSERT INTO kullanılırken dikkat edilmesi gerekenler 8](#_heading=h.1y810tw)

[bir veriyi değiştirme UPDATE etme 9](#_heading=h.4i7ojhp)

[**DAY 4 9**](#_heading=h.2xcytpi)

[Table'dan istenen bir satırı silmek 9](#_heading=h.1ci93xb)

[TABLODAKİ TÜM SATIRLARI SİLME 9](#_heading=h.3whwml4)

[BİRDEN FAZLA SATIR SİLME 9](#_heading=h.2bn6wsx)

[AND KULLANARAK SİLME 9](#_heading=h.qsh70q)

[OR KULLANARAK SİLME 9](#_heading=h.3as4poj)

[TRUNCATE: Tablodaki tüm satırları siler 9](#_heading=h.1pxezwc)

[DROP İLE TABLE SİLME 9](#_heading=h.49x2ik5)

[PURGE KULLANARAK TABLE SİLME 10](#_heading=h.2p2csry)

[SELECT KULLANIMI 10](#_heading=h.147n2zr)

[IN 10](#_heading=h.3o7alnk)

[SUM, COUNT, AVG, MIN, MAX 12](#_heading=h.ihv636)

[**DAY 5 12**](#_heading=h.32hioqz)

[EXISTS 12](#_heading=h.1hmsyys)

[BETWEEN 13](#_heading=h.41mghml)

[NOT BETWEEN, BETWEEN'in tersi olarak çalışır. 13](#_heading=h.2grqrue)

[IS NULL🡺 BOŞ ALANLARI SEÇ 14](#_heading=h.3fwokq0)

[ORDER BY 🡺 SIRALA 14](#_heading=h.1v1yuxt)

[DESC 🡺 BÜYÜKTEN KÜÇÜĞE SIRALAMA 14](#_heading=h.4f1mdlm)

[ASC 🡺 KÜÇÜKTEN BÜYÜĞE SIRALAMA 15](#_heading=h.2u6wntf)

[AS 🡺 RAPOR İÇİN SÜTUN İSİMLERİNİ DEĞİŞTİRME 15](#_heading=h.19c6y18)

[GROUP BY 15](#_heading=h.3tbugp1)

[HAVING 16](#_heading=h.28h4qwu)

[UNION 17](#_heading=h.nmf14n)

[**DAY 6 18**](#_heading=h.37m2jsg)

[UNION ALL 19](#_heading=h.1mrcu09)

[INTERSECT 19](#_heading=h.46r0co2)

[MINUS 21](#_heading=h.2lwamvv)

[JOIN 22](#_heading=h.111kx3o)

[LEFT JOIN 23](#_heading=h.3l18frh)

[RIGHT JOIN 24](#_heading=h.4k668n3)

[FULL JOIN 24](#_heading=h.2zbgiuw)

[**DAY 7 24**](#_heading=h.1egqt2p)

[WİLDCARDS 24](#_heading=h.3ygebqi)

[% : Bir veya birden fazla karakter gösterir. 25](#_heading=h.2dlolyb)

[LIKE gibi demek J% : J ile başlasın devamı ne olursa olsun demektir. 25](#_heading=h.sqyw64)

[\_ : Sadece bir karakteri temsil eder, ne olduğu önemli değil. 25](#_heading=h.3cqmetx)

[[ ] : sembol içine koyulan harflerden herhangi birisi olabilir manasında 26](#_heading=h.1rvwp1q)

[^ : ile başlayan manasında 27](#_heading=h.4bvk7pj)

[(\*) : herşey manasında 27](#_heading=h.2r0uhxc)

[**DAY 8 27**](#_heading=h.1664s55)

[PIVOT: Satırları sütuna dönüştürmek için kullanılır. Özet Tablo(Excel) 28](#_heading=h.3q5sasy)

[DISTINCT: sütun içerisinde geçen dataları tekrarsız olarak gösteren kod. 29](#_heading=h.25b2l0r)

[ALTER TABLE: Tablolar üzerinde değişiklik yapma 30](#_heading=h.34g0dwd)

[ADD: Tabloya yeni sütun ekleme 30](#_heading=h.1jlao46)

[DROP COLUMN: Tablodan sütun silme 32](#_heading=h.43ky6rz)

[RENAME COLUMN: Sütun ismini değiştirme 32](#_heading=h.2iq8gzs)

[RENAME TO: Tablo ismini değiştirme 33](#_heading=h.3hv69ve)

[MODIFY: Tablo sütunlarının yapısını değiştirme 33](#_heading=h.1x0gk37)

[SQL Interview Questions 34](#_heading=h.2w5ecyt)

[MOD METODU: çift veya tek sayıları filtreleme. 36](#_heading=h.1baon6m)

[ROWNUM: Tabloda istenilen sayıda satır sayısını gösterme 37](#_heading=h.3vac5uf)

[**DAY 9 39**](#_heading=h.2afmg28)

[SELF JOIN 41](#_heading=h.pkwqa1)

[UPPER: Bir sütundaki dataların büyük harfle yazılmasını sağlar. 42](#_heading=h.39kk8xu)

[LOWER: Bir sütundaki dataların küçük harfle yazılmasını sağlar. 42](#_heading=h.1opuj5n)

[INITCAP: Bir sütundaki dataların ilk harfini büyük yapar diğerlerini küçük. 42](#_heading=h.48pi1tg)

SQL NOTLARI ( 5 AĞUSTOS 2020 )

# DAY 1

* API farklı applicationlar arasında haberleşmeyi sağlar.
* API’ın haber yollayıp cevap alma kabiliyeti var.
* DBMS data base management system
* Girişi düzenleyen, rapor almayı sağlayan, yeni table lar oluşturmaya, silmeye, diğer applicationlar ile irtibat kurmamızı sağlayan bir systemdir.
* Create: oluşturma
* Read: okuma
* Update: değiştirme, güncelleme
* Delete: silme
* Hepsi CRUD
* Database’den rapor alma: Her bölümden en çok satılan ürün hangisi gibi…
* SQL’de her şey bir tabloda saklanır. SQL içerisinde tablosuz data yoktur.
* Row: satır Column: sütun normalde
* Record: satır Field: Sütun SQL’de bu terimler kullanılıyor
* Header: başlık
* Relational Databases: birbiriyle ilişkili tablolar veya SQL Database aynı manaya gelir. Structured Query Language: yapılandırılmış sorgulama dili.
* Database hızlı çalışsın diye farklı tablolar oluşturulup bunlar birbiriyle ilişkilendirilir.
* Birbiriyle ilişkili tablolar yapısına Schema (şema) denir.
* Microsoft SQL Server : user interface güzel, büyük datalarda çok başarılı ama pahalı.
* MySQL : ücretsiz, kullanan çok, karşılaşılan problemlerle ilgili internette çok çözüm yolları mevcut, çok aşırı iş yüklemesinde başarısız.
* PostgreSQL: piyasaya yeni çıktı. Diğerlerinde olmayan farklı özellikler mevcut. Yüklemesini yapma, çalışır hale getirme vakit alıyor. Yeni başlayanları zorlayabilir. Bunun haricinde başarılı bir program.
* PL / SQL : Güvenlik ön planda, oracle ait bir program. Object – Oriented Programming ile uyumlu.
* Aralarında kullanım olarak çok küçük fark var. Biri öğrenildiğinde diğerleride öğrenilmiş olur.
* SQL olmayan Databaseler: Non Relational Databases : birbiriyle ilişkili olmayan tablolar. Tablo kullanılmadan yapılan veri oluşturma NoSQL denir.

## PRIMARY KEY

* **Primary Key’i** kullanarak uniq(ikincisi olmayan) datalara ulaşırsınız. Tablolarda buna ihtiyaç duyulur. Tablo oluştururken bir data girerek bir şeylere ulaşmak istiyorsak Primary Key oluşturmalıyız. Primary Key tek sütunda olabilir, iki sütun veya üç sütunlada olabilir ama bunlar beraber tek bir Primary Key olur. Ad, soyad, baba adı gibi… Primary key Null olamaz. Bir tane Primary key birden fazla sütun içerebilir. Primary Key yapılacak sütun her tip datayı içeribilir. E-Mail adresi iyi bir Primary Key’dir. Gerçek bir Datayı email gibi Primary Key olarak kullanırsanız “Natural Key” denir. Gerçek olmayan Primary Key’lere “Surrogate Key” denir. İki sütun birleştirilerek Primary Key olarak tanımlanmışsa buna “Composite Key” denir.

## FOREIGN KEY

* **Foreign Key** => Primary Key Başka bir Tablonun Primary Key’i ile irtibat kurulan sütuna Foreign Key deniyor. Primary Key’e hangi sütun bağlanıyorsa ona Foreign Key deniyor. Foreign Key de tekrarlı değerler olabilir. Foreign Key Null değer alabilir. Bir tabloda bir tane Primary Key olabilir ama birden fazla Foreign Key olabilir. Foreign Key Tablolar arasında bağlantıyı sağlar. Foreign Key’in bulunduğu Tabloya Parent derler, bağlanan Tabloya Child Table denir.
* Database validation: veri doğrulama demektir. Tester bu işi yapar.

# DAY 2

* SQL kullanarak Database oluşturup yönetebiliriz, design’ini değiştirebilir, yeni tablolar oluşturabiliriz, silebiliriz, security ayarlarını yapabilir ve kullanıcı yetkilerini ayarlama.
* SQL içerisinde 4 farklı dil var. 1- Data kontrol language 2-Data Definition Language data oluşturma dizayn etme 3- Data Definition Language data değiştirme dili 4- Data Query Language veriyi sorgulama
* dili Oluşturmak ve yönetmek, oluşturmak ve dizayn etmek, data create yapma, security de kullanılıyor.

## VARİABLE’LAR : STRİNG DATA TYPES

* Char(size): çoklu karakteri kabul ediyor. Uzunluğu belli olan verilerde kullanılıyor. TC Kimlik 10 11 hanedir. Uzunluğu bellidir, sabittir, bunun için char kullanılır. Uzunluğu sabit olan verilerde kullanılır.
* Varchar2(size): çoklu karakteri kabul ediyor. Uzunluğunu bilmediğimiz, değişen stringlerde kullanılır. Bir max size belirlenir riske girilmeden bir rakam atanır. Yoksa geri kalan kısmı almaz.
* Nchar(size): n: number demek, ascii kodlarını depolamış oluruz. Uzunluğu belli olan kodların ascii kodlarını depolar.
* Nvarchar2(size): uzunluğu belli olmayan kodların ascii kodlarını depolar. Var: varie değişen demek.
* Veri uzunluğu sabit değişmiyorsa char(size), uzunluk sabit değil bilmiyorsak varchar2(size) kullanıyoruz.

## NUMERİC DATA TYPES

* number(p, s) p: sayıda kullanılan rakamların miktarını gösterir. S: ondalik kısımdaki sayıların miktarını gösterir.
* number(5,2) 123,45 demektir. number(4) 5678 olarak kabul eder ama 12,53 yazarsak ondalık kısmı belirtilmediği için onu 0 kabul eder bu sayıyı hafızaya 12 olarak alır. number(4) ile number(4,0) aynı anlama gelir.
* Number(5,1) 6789,12 böyle bir yazım mümkün değildir, bunu kabul etmez error verir.
* P sayısı 38 den büyük olamaz. S yerine negatif değerler kullanılabiliyor.
* Number(5, -2) = 63781,38 yuvarlayıp 63800 olur. 2 rakam geri git yuvarlama yap demek.
* Number(4,-1) = 6537,2 yuvarlayıp 6540

## DATE DATA TYPES

* 13-Apr-20 standart format budur istenirse format değiştirilebilir.

## BLOB DATA TYPES

* Resim Video gibi data typleri için BLOB denilen bir data type’i vardır.
* Binary Large objects BLOB
* SQL açıp SQL Worksheet sayfasına gittik
* CREATE TABLE students yazdık. Tablo oluştur ismin students olsun
* Birden fazla kelime varsa update\_date şeklinde yazılıyor.
* Bir tablo aşağıdaki şekilde oluşturuluyor

## TABLO OLUŞTURMA

CREATE TABLE students

(

id char(11),

name varchar2(50),

grade number(3),

address varchar2(80),

update\_date

);

* Bunu seçip sağ üstte run yaptık, tablo oluşturdu.
* Schema’ya giderek oluşturduğumuz tablomuzun özelliklerine bakabiliriz.
* Tempory: geçici
* Nested: başka bir tablo içindeki bir tablo mu
* VALİD: geçerli
* Başlık isimlerini küçük yazıyoruz, kendisi otomatik olarak bunları büyük harf olarak yazar.
* Nullable: boş bırakılabilir mi. Hiçbir sütunu primary key olarak tanımlamadığımız için boş bırakılabilir mi sütun izahına yes yazmış.

## Varolan bir tablodan yeni tablo üretmek

* -- mesela sadece isim ve notlar olsun
* CREATE TABLE student\_grade
* AS -- gibi demek
* SELECT id, grade
* FROM students;
* id char(11) NOT NULL, -- bu sütun boş kalmasın istiyorsak NOT NULL yazıyoruz.
* Constraints: sınırlama demektir.
* name varchar2(50) UNIQUE, bir sütunun tekrarlı olmamasını istiyorsak UNIQUE yazıyoruz.

# DAY 3

* Bir sutünu primary key yapabilmek için sonun PRIMARY KEY yazıyoruz
* Constraint: sınırlandırma demektir. NOT NULL, UNIQUE, PRIMARY KEY(boş olmaz, tekrarlı olamaz) bunların hepsi Constraint’tir. Bir sutünu PRIMARY KEY yaptığımız zaman Constraint’te ona SYS\_C00055335 bir rakam atar.
* Status:ENABLED aktif demek
* Nullable: boş olabilir
* Uniqueness: UNIQUE benzeri yok

## Primary Key oluşturmanın birinci yolu

CREATE TABLE students3

(

id char(11) PRIMARY KEY, -- bir sutünu primary key yapmak istiyorsak bu şekilde yazıyoruz.

name varchar2(50),

grade number(3),

address varchar2(80),

update\_date date

);

## Primary Key oluşturmanın ikinci yolu

CREATE TABLE students4

(

id char(11),

name varchar2(50),

grade number(3),

address varchar2(80),

update\_date date,

--CONSTRAİNTS İSİM PRIMARY KEY(SUTUN İSMİ) bazen

CONSTRAINTS id\_pk PRIMARY KEY(id)

);

Bazen primary key ile başka tabloyu birleştirmemiz gerekecek bu durumda id\_pk ismi gerekli olacak.

## Birden fazla sutun kullanarak Primary Key oluşturma

CREATE TABLE students5

(

id char(11),

name varchar2(50),

grade number(3),

address varchar2(80),

update\_date date,

--CONSTRAİNTS İSİM PRIMARY KEY(SUTUN İSMİ) bazen

CONSTRAINTS id\_name\_pk PRIMARY KEY(id, name)

);

* Composite: birleştirmek demek

## Foreign Key oluşturmak

CREATE TABLE students6

(

id char(11),

name varchar2(50),

grade number(3),

address varchar2(80),

update\_date date,

CONSTRAINTS id\_fk FOREIGN KEY(id) REFERENCES students4(id) -- students6 tablosundan students4 tablosundaki primary keye ulaşmak istiyorum.

-- REFERENCES dan sonra tablo ismi ve primary key olan sütun ismini yazıyoruz.

-- bu sutün primary key olmazsa incompatible(uygun değil) diye hata alınır.

-- FOREIGN KEY de index oluşturmaz.

);

## TABLOYA VERİ GİRİŞİ YAPMA

CREATE TABLE students

(

students\_id char(7),

students\_name varchar2(50),

students\_grades number(3),

students\_cinsiyet char(5)

);

## INSERT INTO: içine yerleştir

INSERT INTO students VALUES('2020301', 'Ali Can', 87, 'Erkek');

## Bir tablonun belli field'larına data eklemek

INSERT INTO students(students\_id,students\_name,students\_cinsiyet) VALUES('2020404', 'Emine saz', 'Kız');

INSERT INTO students(students\_name,students\_cinsiyet) VALUES('Asım Genç', 'Erkek');

## TÜM TABLOYU KONSOLA YAZDIRMA

-- altta konsolda yazdırmak için aşağıdaki kodları yazıyoruz.

SELECT \* -- hepsini seç students tablosundan

FROM students;

CREATE TABLE students\_family1

(

students\_id char(7) UNIQUE,

mother\_name varchar2(50) NOT NULL,

father\_name varchar2(50),

address varchar2(80)

);

## INSERT INTO kullanılırken dikkat edilmesi gerekenler

-- 1) Constraints'lerle çelişen data girişi yapmayın

-- 2) Data Type'larına uygun veriler girin

-- 3) Sutun sıraları ile data sıraları uyumlu olmalıdır.

-- aşağıda ilk ikisini çalıştırır , üçüncüsünde error verir. mother\_name boş bırakılamaz, boş bırakılarak giriş yapmak isteniyor hata verir.

INSERT INTO students\_family1 VALUES('2020301', 'Halime', 'Recep' , 'Istanbul Bakirkoy');

INSERT INTO students\_family1 VALUES('2020302' , 'Melahat' , 'Kerem' , 'Istanbul Basaksehir');

INSERT INTO students\_family1(students\_id,father\_name,address) VALUES('2020304', 'Ramazan', 'Istanbul Bagcilar');

## bir veriyi değiştirme UPDATE etme

UPDATE students\_family1

SET address = 'Ankara Cankaya'

WHERE students\_id ='2020301';

# DAY 4

## Table'dan istenen bir satırı silmek

DELETE FROM products -- DELETE bir satır siler

WHERE supplier\_id = 103;

## TABLODAKİ TÜM SATIRLARI SİLME

--DELETE FROM products; kodu table'daki tüm dataları siler ama table'in yapısı durur.

DELETE FROM products; -- bir satır belirtilmezse tüm satırları siler.

## BİRDEN FAZLA SATIR SİLME

DELETE FROM products

WHERE supplier\_id < 102;

## AND KULLANARAK SİLME

--AND kullanırsanız SQL iki şartda sağlanıyorsa bu satırı bulur ve siler

--iki şart birden sağlanmazsa SQL silme işlemini yapamaz, error da vermez.

DELETE FROM products

WHERE product\_name = 'Phone' AND costumer\_name = 'Ramazan';

## OR KULLANARAK SİLME

--altta iki satırıda silecektir

DELETE FROM products

WHERE product\_name = 'Phone' OR costumer\_name = 'Ramazan';

## TRUNCATE: Tablodaki tüm satırları siler

TRUNCATE TABLE products;

--TRUNCATE ile DELETE FROM arasındaki fark DELETE kullanırsanız data tekrar kurtarabilirsiniz, TRUNCATE kullanırsanız tekrar kullanamazsınız.

--çöp kutusuna atma veya del+enter gibi bir şey

--özel bilgilerin geri getirilmemesi isteniyorsa TRUNCATE kullanılır.

## DROP İLE TABLE SİLME

--DROP: Bir Table'ı yapısı ve içindeki dataları ile birlikte silmek için

--kullanılır. Artık tablo yok demektir

--DROP: ingilizcede düşür demektir.

--DROP yazarak imha ettiğimiz TABLE bazı kodlar kullanılarak geri çağırılabilir.

DROP TABLE products;

## PURGE KULLANARAK TABLE SİLME

--DROP kullanarak TABLE imha ettiğinizde geri çağırılmasını engellemek istiyorsanız

DROP TABLE products PURGE; --PURGE: Tenkil manasında

## SELECT KULLANIMI

-- 1) Table'daki tüm dataları(\*) görmek için kod yazınız.

SELECT \*

FROM products

-- 2) Table'daki belli bir satırı görmek için kod yazınız.

SELECT \*

FROM products

WHERE product\_id = 1001;

-- 3) TABLE'daki bazı satırları görmek için kod yazınız.

SELECT \*

FROM products

WHERE supplier\_id = 101 OR supplier\_id = 103;

## IN

-- birden fazla satır görmek için OR dan daha kısa bir çözüm IN kullanmaktır.

SELECT \*

FROM products

WHERE supplier\_id IN (101,103) -- bu küme içinde olanları göster.

SELECT \*

FROM products

WHERE supplier\_id < 103; -- 103 ten küçük olanları göster.

-- 4) Bir TABLE'daki belli bir sutunu görmek için kod yazınız.

SELECT costumer\_name

FROM products;

-- 5) Bir TABLE'da birden fazla sutunu görmek için kod yazınız.

SELECT costumer\_name, product\_name

FROM products;

-- 6) Bir TABLE'dan birden fazla sütun ve bir satır görmek için kod yazınız.

SELECT costumer\_name, product\_name

FROM products

WHERE costumer\_name = 'Suleyman';

-- Tek sütun iki satır

SELECT costumer\_name

FROM products

WHERE costumer\_name IN ('Suleyman', 'Ramazan');

-- SELECT'ten sonra SELECT kullanırsanız içerdeki koda SUBQUERY denir.

-- işçi sayısı 15000 den fazla olan company'lerin company isimlerini ve işçi

-- isimlerini gösteren kodu yazınız.

-- SELECT'ten sonra SELECT kullanırsanız içerdeki SUBQUERY denir.

SELECT company, name

FROM employees

WHERE company IN ( SELECT company

FROM companies

WHERE number\_of\_employees >15000);

-- company ıd si 102 den küçük olan companylerin salary'lerini ve state lerini

-- gösteren kodu yazınız

SELECT salary, state

FROM employees

WHERE company IN ( SELECT company

FROM companies

WHERE company\_id < 102);

-- Florida'daki company'lerin company id lerini ve number of employees i gösteren kodu yazınız

SELECT company\_id, number\_of\_employees

FROM companies

WHERE company IN ( SELECT company

FROM employees

WHERE state = 'Florida');

-- NOT: SUBQUERY'ler WHERE den sonra kullanılabildiği gibi SELECT'den sonra da kullanılabilirler.

-- Her şirketteki işçi sayısını ve ortalama işçi ücretlerini gösteren kodu yazınız.

SELECT company,number\_of\_employees, (SELECT AVG(salary) -- AVG ortalamasını alan bir method

FROM employees

WHERE companies.company=employees.company) average\_salary -- . içindeki manasında

FROM companies;

-- her şirketteki toplam state sayısını gösteren kodu yazınız( COUNT(state))

SELECT company, (SELECT COUNT(state)

FROM employees

WHERE companies.company=employees.company) count\_state

FROM companies;

-- Her şirketin company id'sini ve işçilerine yaptığı toplam ödemeyi gösteren kodu yazınız

SELECT company,company\_id, (SELECT SUM(salary)

FROM employees

WHERE companies.company=employees.company) sum\_salary

FROM companies;

## SUM, COUNT, AVG, MIN, MAX

gibi fonksiyonlar AGGREGATE fonksiyonlar olarak adlandırılır.

* SUM: sütun toplamını alır. SELECT SUM(salary)
* COUNT: sütunda geçen adet sayısını yazar. SELECT COUNT(state)
* AVG: sütunun ortalamasını alır. SELECT AVG(salary)
* MIN: sütun içerisindeki min değeri alır. SELECT MIN(salary)
* MAX: sütun içerisindeki max değeri alır. SELECT MAX(salary)

# DAY 5

## EXISTS

-- EXISTS komutu SUBQUERY'lerle beraber kullanılır.

-- EXISTS: sütunda var mı yokmu diye bakar

-- IN komutu OR komutunun kısa yazılmış halidir. IN komutu tek başına SUBQUERY'lerle

-- Beraber kullanılmıyor. SUBQUERY kullanacaksanız EXISTS kullanmamız gerekir.

-- Product\_id leri aynı olan müşterilerin isimlerini gösteriniz.

SELECT customer\_name

FROM customers\_products

WHERE EXISTS (SELECT product\_id

FROM customers\_likes

WHERE customers\_products.product\_id = customers\_likes.product\_id

);

## BETWEEN

-- BETWEEN(arasında ==> 10 ile 20 arasında : BETWEEN 10 AND 20) komutunun daha anlaşılır halidir.

-- BETWEEN komutunda sınırlar dahildir. (10 ve 20 dahil)

-- BETWEEN kullanırken ilk datanın ikinciden küçük olması gerekiyor.

-- Product\_id'si 20 ile 40 arasında olan productların isimlerini ve product\_id lerini gösteren kodu yazınız.

SELECT product\_name, product\_id

FROM customers\_products

WHERE product\_id BETWEEN 20 AND 40;

-- Product\_name ile Liked\_product'ı aynı olan müşterilerin isimlerini gösteren

-- kodu EXISTS kullanarak yazınız.

SELECT customer\_name

FROM customers\_products

WHERE EXISTS ( SELECT liked\_product

FROM customers\_likes

WHERE customers\_products.product\_name = customers\_likes.liked\_product

);

-- ismi J'den T'ye kadar harflerle başlayan müşterilerin tüm bilgilerini gösteren kodu yazınız

SELECT \*

FROM customers\_products

WHERE customer\_name BETWEEN 'J' AND 'T';

## NOT BETWEEN, BETWEEN'in tersi olarak çalışır.

-- product\_id'si 20 ile 40 arasında olmayan product'ların tüm bilgilerini gösteren kodu yazınız.

SELECT \*

FROM customers\_products

WHERE product\_id NOT BETWEEN 20 AND 40;

## IS NULL🡺 BOŞ ALANLARI SEÇ

-- IS NULL data girilmemiş olan satırları seçmek için kullanılır.

-- customers\_products Tablosundan müşteri ismi girilmemiş dataların tamamını gösteren kodu yazınız.

SELECT \*

FROM customers\_products

WHERE customer\_name IS NULL;

-- IS NULL, UPDATE komutu ile de kullanılır

UPDATE customers\_products

SET customer\_name = 'İsim girilmemiş'

WHERE customer\_name IS NULL;

## ORDER BY 🡺 SIRALA

-- ORDER BY dataların belli bir field'a göre natural orda şeklinde sıralanmasına yarar.

-- ORDER BY'dan sonra field(sütun) ismi kullanıldığı gibi field numarasıda kullanılabilir.

-- Yani; ORDER BY product\_id; ile ORDER BY 1; aynı şeydir

-- customers\_products'daki tüm dataları product\_name'e göre sıralayan kodu yazınız.

SELECT \*

FROM customers\_products

ORDER BY product\_name; -- product\_name' e göre sırala dedik.

-- customers\_products'daki customer\_name'i 'Mark' olan dataları product\_id'ye göre sıralayan kodu yazınız.

SELECT \*

FROM customers\_products

WHERE customer\_name = 'Mark'

ORDER BY product\_id;

-- aşağıdaki kod ile yukarıdaki aynı sonucu verir.

SELECT \*

FROM customers\_products

WHERE customer\_name = 'Mark'

ORDER BY 1;

## DESC 🡺 BÜYÜKTEN KÜÇÜĞE SIRALAMA

-- customers\_products table'ındaki tüm dataları product\_id'lerine göre

-- büyükten küçüğe(reverse order / descending order) sıralayınız.

SELECT \*

FROM customers\_products

ORDER BY product\_id DESC;

## ASC 🡺 KÜÇÜKTEN BÜYÜĞE SIRALAMA

-- customers\_products table'ındaki tüm dataları product\_name'lerine göre

-- büyükten küçüğe(reverse order / descending order) sıralayınız.

-- customer\_name'lerine göre de natural order'da sıralayınız.

SELECT \*

FROM customers\_products

ORDER BY product\_name DESC, customer\_name ASC;

## AS 🡺 RAPOR İÇİN SÜTUN İSİMLERİNİ DEĞİŞTİRME

-- ALIASESE Tablodaki field(sütun) isimlerini çıktıda farklı görmek için kullanılır (Data Base değişmiyor)

-- AS kullandığınızda database deki tablonun sütun isimleri değişmez siz

-- sadece yeni sütun isimlerine sahip rapor alırsınız.

-- Field isimlerini Türkçeleştirin.

SELECT product\_id AS urun\_kodu, customer\_name AS musteri\_adı, product\_name AS urun\_ismi

FROM customers\_products;

-- Field isimlerini Türkçeleştirin ve product\_id ile product\_name aynı sütunda olsun.

SELECT customer\_name AS musteri\_ismi, product\_id || product\_name AS urun\_kodu\_ismi

FROM customers\_products;

-- product\_id || product\_name AS urun\_kodu\_ismi

-- ÜSTTEKİ KOD İLE DATABASE DEĞİŞMEDEN İKİ SATIRI BİRLEŞİRİP O SÜTUNA YENİ İSİM ATAMA YAPTIK

## GROUP BY

-- GROUP BY dataları gruplandırarak görmemizi sağlar

-- Herbir product'ı alan müşteri sayısını gösteren kodu yazınız.

SELECT product\_name, COUNT(product\_name) AS number\_of\_customers

FROM customers\_products

GROUP BY product\_name

-- Herbir product\_id'nin kaç kere kullanıldığını gösteren kodu yazınız.

SELECT product\_id, COUNT(product\_id) AS number\_of\_productID

FROM customers\_products

GROUP BY product\_id

-- employees tablosundan her içşçinin aldığı toplam ücreti bulunuz.

SELECT name, SUM (salary) AS total\_salary

FROM employees

GROUP BY name;

--Herbir state'de kaç işçi olduğunu gösteren kodu yazınız

SELECT state, COUNT(name) AS total\_employees

FROM employees

GROUP BY state;

--Herbir şirkette maaşı 2000'inin üzerinde olan kaç işçi olduğunu gösteren kodu yazınız.

SELECT company, COUNT(name) AS total\_employees

FROM employees

WHERE salary>2000

GROUP BY company;

--Herbir şirkette verilen max ve min ücretleri gösteren kodu yazınız

SELECT company, MAX(salary) AS max\_salary, MIN(salary) AS min\_salary

FROM employees

GROUP BY company

## HAVING

--HAVING, GROUP BY'dan sonra AGGREGATE functionlarla filtrelemek için kullanılır.

--Herbir şirketin min salary'lerini 2000 üzerinde ise göster ve max salary'lerini göster.

SELECT company, MIN(salary) AS min\_salary, MAX(salary) AS max\_salary

FROM employees

GROUP BY company

HAVING MIN(salary)>2000;

--Toplam geliri 2500 den fazla olan herbir işçiyi gösteren kodu yazınız

SELECT name, SUM(salary) AS sum\_salary

FROM employees

GROUP BY name

HAVING SUM(salary)>2500;

--Herbir State'de çalışan işçi sayısını ve state ismini gösteren kodu yazınız

-- ama işçi sayısı birden fazla olmalı

SELECT state, COUNT(name) AS number\_of\_employees

FROM employees

GROUP BY state

HAVING COUNT(name)>1;

--Herbir State'te verilen max ücreti 3000'den az ise gösteren kodu yazınız

SELECT state, MAX(salary) AS max\_salary

FROM employees

GROUP BY state

HAVING MAX(salary)<3000; --HAVING=...olan

## UNION

--UNION Operation: iki farklı sorgulamanın sonuclarını birleştirme işlemidir.

--Aynı sütünda birleştirilecek sütunların data typleri aynı olması gerekli yoksa hata verir.

--UNION kullanırken her iki sorgudaki sütun data typleri örtüşmelidir.

--Salary'si 3000'den fazla olan state ve işçi isimlerini gösteren kodu yazınız.

SELECT state AS name\_and\_state,salary

FROM employees

WHERE salary>3000

UNION

SELECT name AS name\_and\_state,salary

FROM employees

WHERE salary>3000;

--Eddie Murphy'inin aldığı toplam ücreti ve Florida'da ödenen ücretleri

--bir tabloda gösteren kodu yazınız

SELECT name AS name\_state, salary

FROM employees

WHERE name = 'Eddie Murphy'

UNION

SELECT state AS name\_state, salary

FROM employees

WHERE state= 'Florida';

--state'lerde ödenen ücreti 3000'den fazla olan ve işçilere ödenen ücreti 2000

-- den az olanları bir tabloda gösteren kodu yazınız

SELECT state AS name\_and\_state, salary

FROM employees

WHERE salary>3000

UNION

SELECT name AS name\_and\_state, salary

FROM employees

WHERE salary<2000;

# DAY 6

CREATE TABLE students

(

students\_id char(9),

students\_name varchar2(50),

students\_address varchar2(80),

students\_grade number(3),

last\_modification\_date date,

CONSTRAINTS id\_pk PRIMARY KEY(students\_id)

);

​

INSERT INTO students VALUES('123456789', 'Ali Can', 'Istanbul', 93, '12-Aug-20');

INSERT INTO students VALUES('234567890', 'Veli Han', 'Istanbul', 95, '13-Aug-20');

INSERT INTO students VALUES('345678901', 'Ayse Tan', 'Berlin', 95, '13-Aug-20');

​

SELECT \*

FROM students;

​

​

CREATE TABLE students\_information

(

students\_id char(9),

students\_phone char(10) UNIQUE,

students\_avg\_score number(4,2) NOT NULL,

CONSTRAINTS id\_fk FOREIGN KEY(students\_id) REFERENCES students(students\_id)

);

​

INSERT INTO students\_information VALUES('123456789', '4071234567', 78);

INSERT INTO students\_information VALUES('234567890', '4071234598', 91);

INSERT INTO students\_information VALUES('345678901', '4071230000', 93);

SELECT \*

FROM students\_information;

--students tablosundan 'Ali Can' ın adresini ve grade'ini alınız.students\_information

--tablosundan da Ali Can'ın tlf numarası ve average score(ortalamasını) alınız

-- ve bunları bir tabloda birleştirerek gösteriniz

SELECT students\_name AS id\_name, students\_address AS address\_phone, students\_grade AS avg\_scrore\_grade

FROM students

WHERE students\_id = '123456789'

UNION

SELECT students\_id , students\_phone, students\_avg\_score

FROM students\_information

WHERE students\_id = '123456789'

--UNION kullanırken üstteki tablo ile alttaki tablo sütun sayıları aynı olması lazım.

--eşleştirilen sütunların data typleri aynı olması lazım yoksa hata verir.

--UNION işlemi aynı recod(satır)ları sadece bir kere yazar.

--UNION işlemi kullanırsanız Tabloda tekrarlı record göremezsiniz.

--aynı query'i union yaptığımızda bunu görürürüz.

SELECT students\_id, students\_address, students\_grade

FROM students

WHERE students\_id = '123456789'

UNION

SELECT students\_id, students\_address, students\_grade

FROM students

WHERE students\_id = '123456789';

## UNION ALL

--olsun ben ikisinide görmek istiyorum derseniz o zaman UNION ALL kullanabilirsiniz.

SELECT students\_id, students\_address, students\_grade

FROM students

WHERE students\_id = '123456789'

UNION ALL

SELECT students\_id, students\_address, students\_grade

FROM students

WHERE students\_id = '123456789';

## INTERSECT

--iki tane query'nin(sorgu) ortak sonuçlarını görmek için kullanılır

--iki sorgulamada kesişim olanları göster

-- INTERSECT işleminde kesişimde ortak eleman bulamazsa no data founda yazar error vermez.

SELECT students\_grade AS grade\_avg\_score

FROM students

INTERSECT

SELECT students\_avg\_score

FROM students\_information;

--Grade'i 94'ten küçük olanlarla, average score'u 80 den büyük olanların keşişimini gösteriniz.

SELECT students\_grade

FROM students

WHERE students\_grade < 94

INTERSECT

SELECT students\_avg\_score

FROM students\_information

WHERE students\_avg\_score > 80;

CREATE TABLE employees

(

id number(9),

name varchar2(50),

state varchar2(50),

salary number(20),

company varchar2(20)

);

INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');

INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');

INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');

INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');

INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');

INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');

INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');

--salary'si 3000 den çok , 2000'den az olan işçilerin tüm bilgilerini tekrar olmadan

--bir tabloda gösteren kodu yazınız.

SELECT \*

FROM employees

WHERE salary > 3000

UNION

SELECT \*

FROM employees

WHERE salary < 2000

--salary'si 3000'den çok 2000'den az olan işçilerin ortak isimlerini gösteren kodu yazınız.

SELECT name

FROM employees

WHERE salary > 3000

INTERSECT

SELECT name

FROM employees

WHERE salary < 2000

--IBM, APPLE ve GOOGLE da çalışıp 3000 den fazla salary'si olan işçileri gösteren kodu yazınız

SELECT name

FROM employees

WHERE company IN ('IBM','APPLE','GOOGLE')

INTERSECT

SELECT name

FROM employees

WHERE salary > 3000;

## MINUS

--MINUS işlemi ilk query(sorgu)'nun sonucları arasından ikinci query'de olanları

--silmek için kullanılır

--google'da çalışmayıp 2000'den düşük maaş alanların isimlerini ve company isimlerini

--gösteren kodu yazınız

SELECT name, company

FROM employees

WHERE salary < 2000

MINUS

SELECT name, company

FROM employees

WHERE company = 'GOOGLE';

--ismi Eddy Murphy olup Texas'ta yaşamayanları ve statelerini gösterenleri gösteren kodu yazınız

SELECT name, state

FROM employees

WHERE name = 'Eddie Murphy'

MINUS

SELECT name, state

FROM employees

WHERE state = 'Texas';

## JOIN

--JOINS: iki tablodaki dataları birleştirmeye yarar.

--JOINS tabloları birleştirmeye, UNION sorgulamaları birleştirmeye yarar.

--5 tane JOINS var.

-- 1) INNER JOIN : iki tablodaki ortak dataları gösterir.

-- 2) LEFT JOIN : ilk tabloda var olan dataları gösterir.

-- 3) RIGHT JOIN :ikinci tabloda var olan dataları gösterir.

-- 4) FULL JOIN: iki tablodaki var olan tüm dataları gösterir.

-- 5) SELF JOIN: tabloyu kendi kendiyle birleştirme işlemi.

CREATE TABLE companies

(

company\_id number(9),

company\_name varchar2(20)

);

INSERT INTO companies VALUES(100, 'IBM');

INSERT INTO companies VALUES(101, 'GOOGLE');

INSERT INTO companies VALUES(102, 'MICROSOFT');

INSERT INTO companies VALUES(103, 'APPLE');

SELECT \*

FROM companies;

-- order: sipariş demek

CREATE TABLE orders

(

order\_id number(9),

company\_id number(9),

order\_date date

);

INSERT INTO orders VALUES(11, 101, '17-Apr-2020');

INSERT INTO orders VALUES(22, 102, '18-Apr-2020');

INSERT INTO orders VALUES(33, 103, '19-Apr-2020');

INSERT INTO orders VALUES(44, 104, '20-Apr-2020');

INSERT INTO orders VALUES(55, 105, '21-Apr-2020');

SELECT \*

FROM orders;

--Companies table'daki Company id'leri ile orders table'daki company id'leri

--aynı olan order'ların campanyi name'lerini, order id'lerini ve order date'lerini

-- gösteren bir talo oluşturan kodu yazınız.

SELECT companies.company\_name, orders.order\_id, orders.order\_date

FROM companies

INNER JOIN orders -- companies tablosu orders tablosuyla birleştir, bu birleşimden aşağıdaki kurala göre alacaklarını al.

ON companies.company\_id = orders.company\_id;

-- 1) SELECT'den sonra Tabloda görmek istediğiniz sütun isimlerini yazarken

-- tablo ismi + nokta + sütun ismi şeklinde yazınız.

-- 2) FROM'dan sonra iki tablo ile çalıştığımızdan FROM'dan sonra Tablo ismi

-- yazarken birinci tablo ismi + INNER JOIN + ikinci tablo ismi yazmalıyız.

-- 3) JOIN'i hangi kurala göre yapacağımızı belirtmelissiniz bunun için de ON + kuralınızı yazmalısınız.

-- 4) INNER JOIN yerine sadece JOIN'de yazılabilir.

## LEFT JOIN

--LEFT JOIN'de ilk tablodaki tüm datalar gösterilir.İlk tablodaki datalara ikinci tablodan

--gelen ek datalar var ise bu ek datalar ortak datalar için gösterilir ancak ortak olmayan

--datalar için o kısımlar boş bırakılır.

SELECT companies.company\_name, orders.order\_id, orders.order\_date

FROM companies

LEFT JOIN orders

ON companies.company\_id = orders.company\_id;

## RIGHT JOIN

--RIGHT JOIN'de ilk tablodaki tüm datalar gösterilir.İlk tablodaki datalara ikinci tablodan

--gelen ek datalar var ise bu ek datalar ortak datalar için gösterilir ancak ortak olmayan

--datalar için o kısımlar boş bırakılır.

SELECT companies.company\_name, orders.order\_id, orders.order\_date

FROM companies

RIGHT JOIN orders

ON companies.company\_id = orders.company\_id;

## FULL JOIN

--FULL JOIN'de iki tablodada var olan tüm datalar gösterilir. genellikle bazı datalar

--boş kalabilir.

SELECT companies.company\_name, orders.order\_id, orders.order\_date

FROM companies

FULL JOIN orders

ON companies.company\_id = orders.company\_id;

# DAY 7

## WİLDCARDS

--Wildcards ==> % \_ [ ]

CREATE TABLE customers

(

customer\_id number(10) UNIQUE,

customer\_name varchar2(50) NOT NULL,

income number(6)

);

INSERT INTO customers (customer\_id, customer\_name, income)

VALUES (1001, 'John', 62000);

​

INSERT INTO customers (customer\_id, customer\_name, income)

VALUES (1002, 'Jane', 57500);

​

INSERT INTO customers (customer\_id, customer\_name, income)

VALUES (1003, 'Brad', 71000);

​

INSERT INTO customers (customer\_id, customer\_name, income)

VALUES (1004, 'Manse', 42000);

INSERT INTO customers (customer\_id, customer\_name, income)

VALUES (1005, 'Can', 57500);

​

INSERT INTO customers (customer\_id, customer\_name, income)

VALUES (1006, 'Cin', 71000);

​

INSERT INTO customers (customer\_id, customer\_name, income)

VALUES (1007, 'Con', 42000);

SELECT \*

FROM customers

## % : Bir veya birden fazla karakter gösterir.

--Ismi J harfi ile başlayan müşterilerin tüm bilgilerini gösteren bir tablo

-- oluşturmak için kod yazınız.

SELECT \*

FROM customers

WHERE customer\_name LIKE 'J%';

## LIKE gibi demek J% : J ile başlasın devamı ne olursa olsun demektir.

--Ismi 'e' ile biten müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan

-- kodu yazınız

SELECT customer\_name,income

FROM customers

WHERE customer\_name LIKE '%e';

--İsminin içinde 'n' olan müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan

-- kodu yazınız

SELECT customer\_name,income

FROM customers

WHERE customer\_name LIKE '%n%';

## \_ : Sadece bir karakteri temsil eder, ne olduğu önemli değil.

--İsmi dört harfli olup son üç harfi 'ohn' olan müşterilerin müşteri adı ve gelirlerini

--gösteren tabloyu oluşturan kodu yazınız.

SELECT customer\_name,income

FROM customers

WHERE customer\_name LIKE '\_ohn';

--İsmi dört harfli olup son iki harfi 'ne' olan müşterilerin müşteri adı ve gelirlerini

--gösteren tabloyu oluşturan kodu yazınız.

SELECT customer\_name,income

FROM customers

WHERE customer\_name LIKE '\_\_ne';

--İsmi dört harfli olup son üçüncü harfi 'n' olan müşterilerin müşteri adı ve gelirlerini

--gösteren tabloyu oluşturan kodu yazınız.

SELECT customer\_name,income

FROM customers

WHERE customer\_name LIKE '\_\_n\_';

--ikinci harfi 'a' olan müşterilerin müşteri adı ve gelirlerini

--gösteren tabloyu oluşturan kodu yazınız.

SELECT customer\_name,income

FROM customers

WHERE customer\_name LIKE '\_a%';

--Üçüncü harfi 'n' olan en az 5 harfli isme sahip olan müşterilerin müşteri adı ve gelirlerini

--gösteren tabloyu oluşturan kodu yazınız.

SELECT customer\_name,income

FROM customers

WHERE customer\_name LIKE '\_\_n\_\_%';

--'B' ile başlayıp 3.harfi 'a' olan isimlere sahip olan müşterilerin müşteri adı ve gelirlerini

--gösteren tabloyu oluşturan kodu yazınız.

SELECT customer\_name,income

FROM customers

WHERE customer\_name LIKE 'B\_a%';

## [ ] : sembol içine koyulan harflerden herhangi birisi olabilir manasında

--İlk harfi 'C' olan son harfi 'n' olan ikinci harfi 'a' veya 'i' olan 3 harfli isimlere sahip olan müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan kodu yazınız.

SELECT customer\_name,income

FROM customers

WHERE REGEXP\_LIKE(customer\_name, 'C[ai]n');

--REGEXP: Regelarexpaction

--İlk harfi 'C' olan son harfi 'n' olan ikinci harfi 'a' dan 'k'ya tüm harfler olan 3 harfli isimlere sahip olan müşterilerin

--müşteri adı ve gelirlerini gösteren tabloyu oluşturan kodu yazınız

SELECT customer\_name,income

FROM customers

WHERE REGEXP\_LIKE(customer\_name, 'C[a-k]n');

​

--Icinde 'a' veya 'n' olan tum isimlere sahip olan musterilerin musteri adi ve gelirlerini gosteren tabloyu olusturan kodu yaziniz.

SELECT customer\_name, income

FROM customers

WHERE REGEXP\_LIKE(customer\_name, '[an](\*)');

​

--'J' veya 'M' ile baslayan tum isimlere sahip olan musterilerin musteri adi ve gelirlerini gosteren tabloyu olusturan kodu yaziniz.

SELECT customer\_name, income

FROM customers

WHERE REGEXP\_LIKE(customer\_name, '^[JM](\*)');

## ^ : ile başlayan manasında

## (\*) : herşey manasında

​

--Ilk harfi 'J' olmayan tum isimlere sahip olan musterilerin musteri adi ve gelirlerini gosteren tabloyu olusturan kodu yaziniz.

SELECT customer\_name, income

FROM customers

WHERE customer\_name NOT LIKE 'J%';

​

--'a' icermeyen tum isimlere sahip olan musterilerin musteri adi ve gelirlerini gosteren tabloyu olusturan kodu yaziniz.

SELECT customer\_name, income

FROM customers

WHERE customer\_name NOT LIKE '%a%';

​

--Ikinci harfi 'a' olmayan isimlere sahip olan musterilerin musteri adi ve gelirlerini gosteren tabloyu olusturan kodu yaziniz.

SELECT customer\_name, income

FROM customers

WHERE customer\_name NOT LIKE '\_a%';

# DAY 8

CREATE TABLE customers\_products

(

product\_id number(10),

customer\_name varchar2(50),

product\_name varchar2(50)

);

INSERT INTO customers\_products VALUES (10, 'Mark', 'Orange');

INSERT INTO customers\_products VALUES (10, 'Mark', 'Orange');

INSERT INTO customers\_products VALUES (20, 'John', 'Apple');

INSERT INTO customers\_products VALUES (30, 'Amy', 'Palm');

INSERT INTO customers\_products VALUES (20, 'Mark', 'Apple');

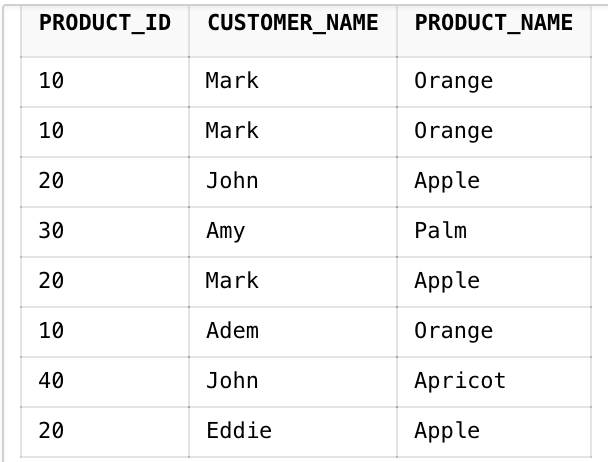
INSERT INTO customers\_products VALUES (10, 'Adem', 'Orange');

INSERT INTO customers\_products VALUES (40, 'John', 'Apricot');

INSERT INTO customers\_products VALUES (20, 'Eddie', 'Apple');

SELECT \*

FROM customers\_products;



## PIVOT: Satırları sütuna dönüştürmek için kullanılır. Özet Tablo(Excel)

SELECT \* FROM (SELECT customer\_name, product\_name FROM customers\_products)

PIVOT

(COUNT(product\_name) FOR product\_name IN ('Orange', 'Apple', 'Palm', 'Apricot'));

--product nameleri say product name için içinde orange, apple, palm, apricot

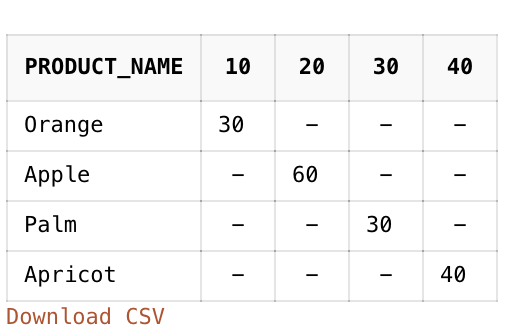
ışık, otobüs, cadde, saat içeren bir resim

Açıklama otomatik olarak oluşturuldu

SELECT \* FROM (SELECT product\_id, product\_name FROM customers\_products)

PIVOT

(SUM (product\_id) FOR product\_id IN (10,20,30,40));



## DISTINCT: sütun içerisinde geçen dataları tekrarsız olarak gösteren kod.

--product name leri tekrarsız olarak gösteren kodu yazınız.

SELECT DISTINCT product\_name

FROM customers\_products;



--kac farklı meyve var bunun sayısını gösteren kodu yazınız

SELECT COUNT(DISTINCT product\_name) AS meyve\_cesit\_sayısı

FROM customers\_products;

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

## ALTER TABLE: Tablolar üzerinde değişiklik yapma

CREATE TABLE employees

(

id number(9),

name varchar2(50),

state varchar2(50),

salary number(20),

company varchar2(20)

);

INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');

INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');

INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');

INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');

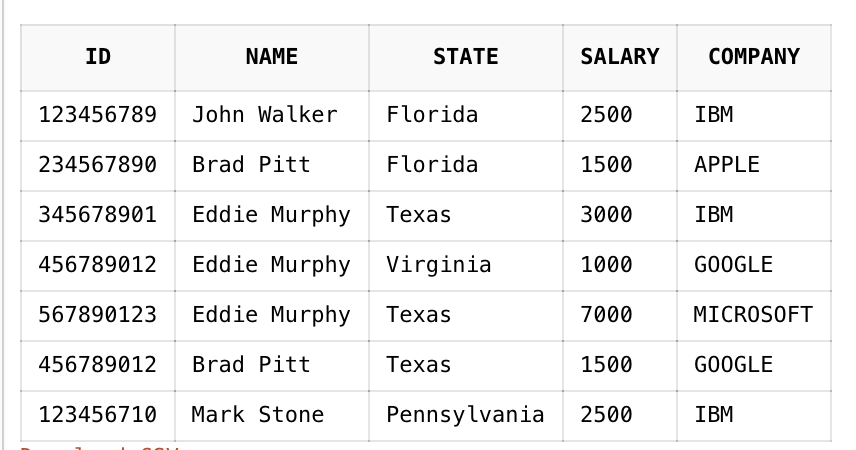
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');

INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');

INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');

SELECT \*

FROM employees;

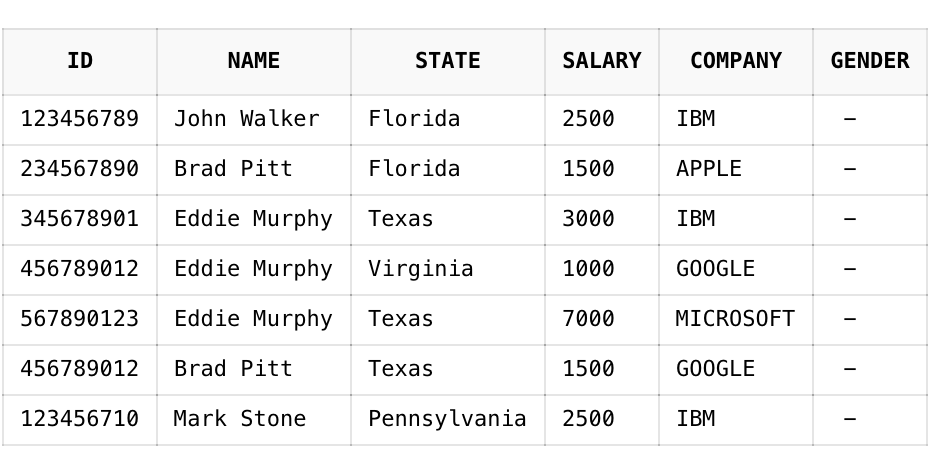


ALTER TABLE

## ADD: Tabloya yeni sütun ekleme

ALTER TABLE employees

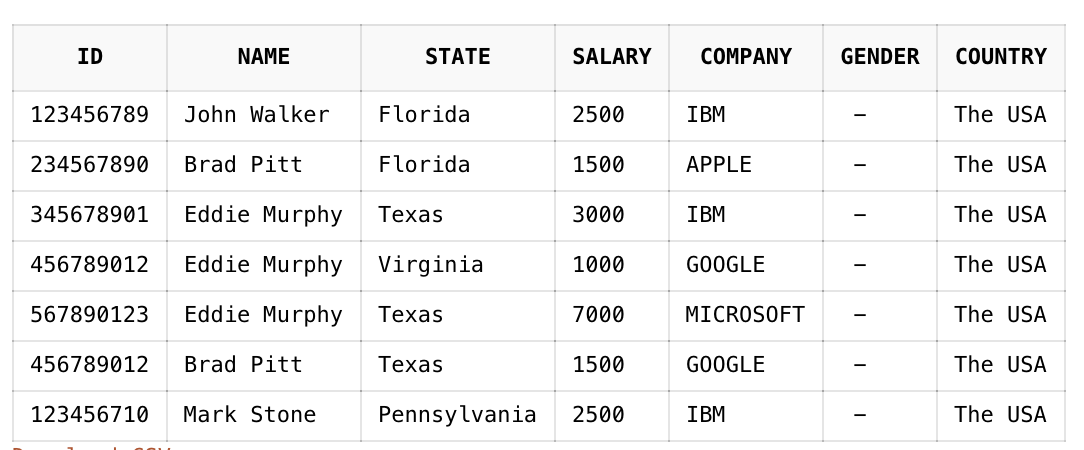
ADD gender varchar2(20)



--Tabloya country sütunu ekleyin her bir satır içinde 'The USA' olsun

ALTER TABLE employees

ADD country varchar2(50) DEFAULT 'The USA';



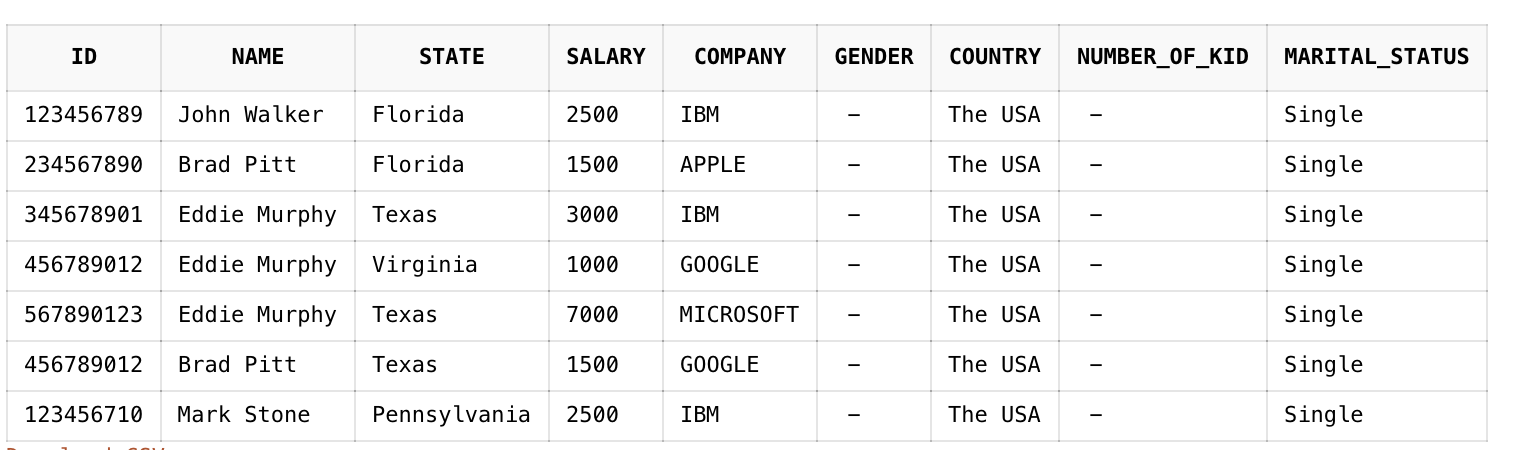
--Tabloya aynı anda birden fazla sütun ekleme

ALTER TABLE employees

ADD (number\_of\_kid number(2),

marital\_status varchar2(30) DEFAULT 'Single'

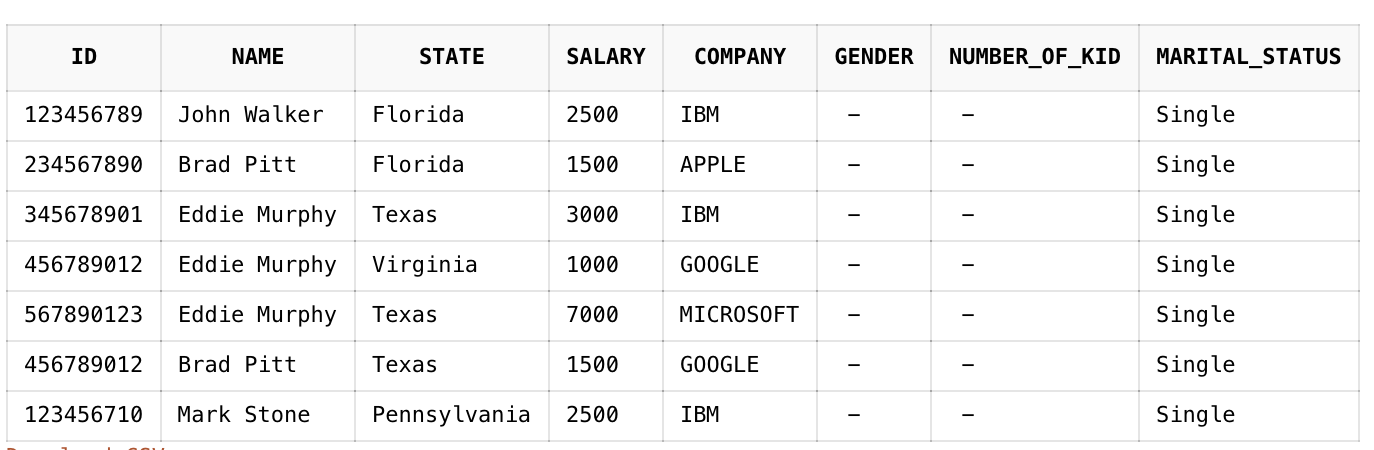
);



## DROP COLUMN: Tablodan sütun silme

ALTER TABLE employees

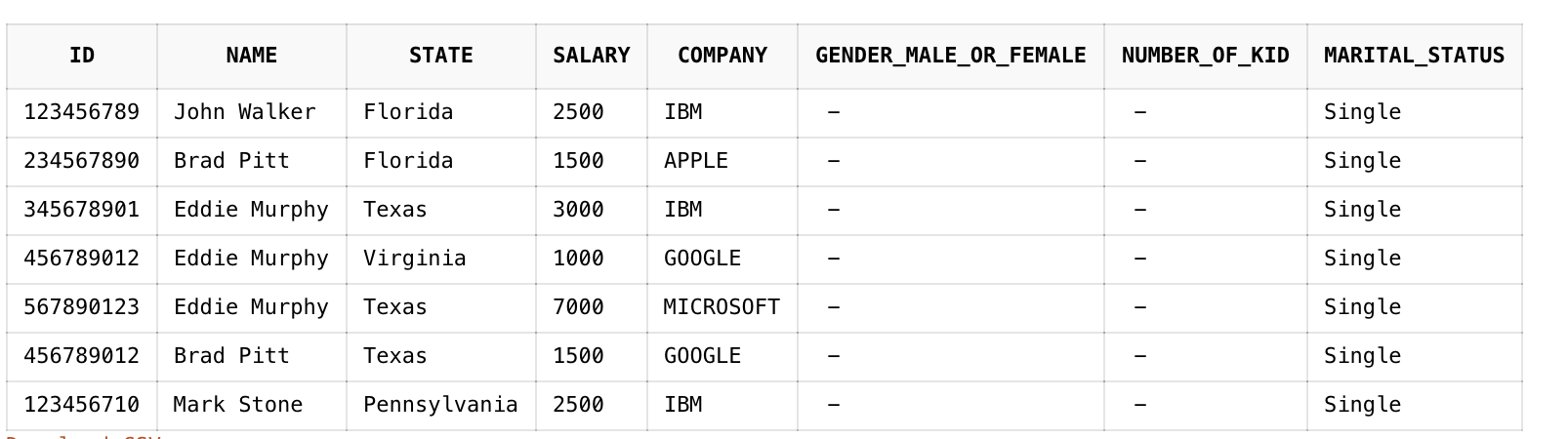
DROP COLUMN country;



## RENAME COLUMN: Sütun ismini değiştirme

ALTER TABLE employees

RENAME COLUMN gender TO gender\_male\_or\_female;



## RENAME TO: Tablo ismini değiştirme

ALTER TABLE employees

RENAME TO workers;

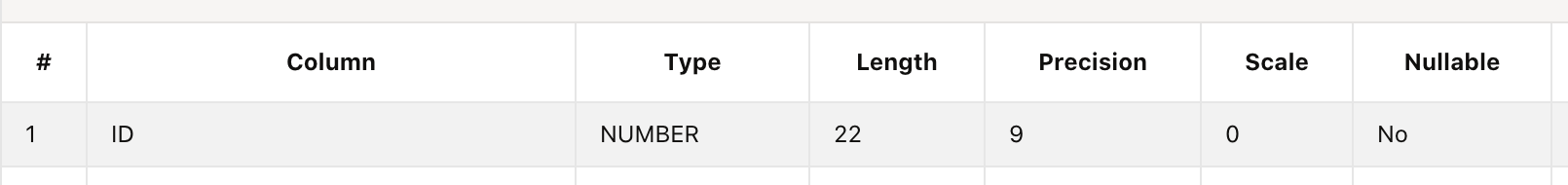
ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

## MODIFY: Tablo sütunlarının yapısını değiştirme

ALTER TABLE workers

MODIFY id number(9) NOT NULL;



--Birden fazla sütunun yapısını aynı anda değiştirmek

ALTER TABLE workers

MODIFY (state char(45) NOT NULL,

company char(30)

);

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

## SQL Interview Questions

CREATE TABLE students

(

id number(9),

name varchar2(50),

state varchar2(50),

salary number(20),

company varchar2(20)

);

INSERT INTO students VALUES(123456789, 'Johnny Walk', 'New Hampshire', 2500, 'IBM');

INSERT INTO students VALUES(234567891, 'Brian Pitt', 'Florida', 1500, 'LINUX');

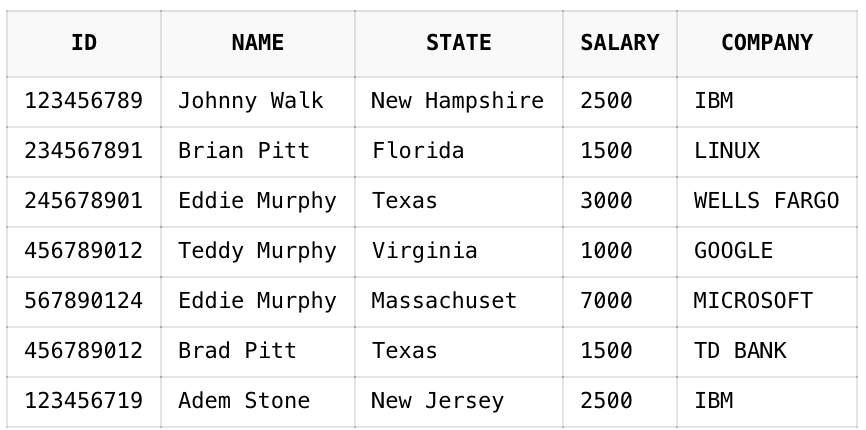
INSERT INTO students VALUES(245678901, 'Eddie Murphy', 'Texas', 3000, 'WELLS FARGO');

INSERT INTO students VALUES(456789012, 'Teddy Murphy', 'Virginia', 1000, 'GOOGLE');

INSERT INTO students VALUES(567890124, 'Eddie Murphy', 'Massachuset', 7000, 'MICROSOFT');

INSERT INTO students VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'TD BANK');

INSERT INTO students VALUES(123456719, 'Adem Stone', 'New Jersey', 2500, 'IBM');



CREATE TABLE workers01

(

id number(9),

name varchar2(50),

state varchar2(50),

salary number(20),

company varchar2(20)

);

INSERT INTO workers01 VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');

INSERT INTO workers01 VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');

INSERT INTO workers01 VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');

INSERT INTO workers01 VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');

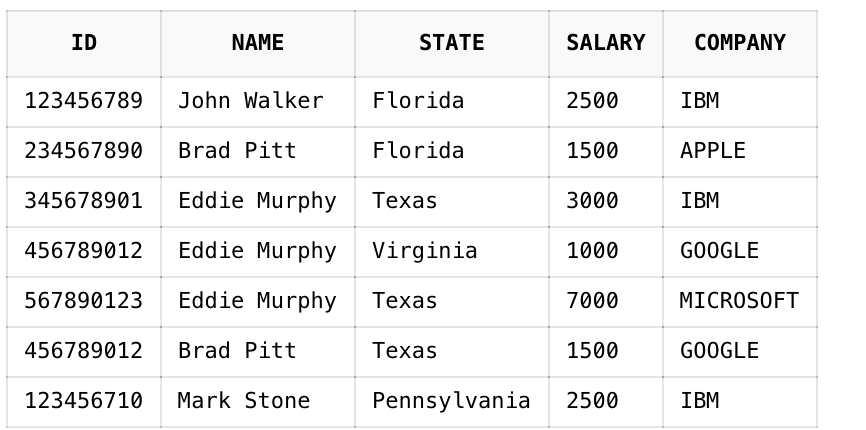
INSERT INTO workers01 VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');

INSERT INTO workers01 VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');

INSERT INTO workers01 VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');

SELECT \*

FROM workers01;



--her iki tablodaki ortak ıd ve isimleri gösteren kodu yazınız

SELECT id,name

FROM students

INTERSECT

SELECT id,name

FROM workers01;

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

--students tablosunda kac farklı state'den student var.

SELECT COUNT (DISTINCT state) AS num\_of\_states

FROM students;

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

## MOD METODU: çift veya tek sayıları filtreleme.

--id si çift sayı olan öğrencilerin tüm bilgilerini gösteren kodu yazınız

SELECT \*

FROM students

WHERE MOD(id,2)=0;

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

----id si tek sayı olan öğrencilerin tüm bilgilerini gösteren kodu yazınız

SELECT \*

FROM students

WHERE MOD(id,2)=1;

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

--Table'da kaç tane record olduğunu gösteren kodu yazınız.

SELECT COUNT(\*) AS num\_of\_record

FROM students;

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

## ROWNUM: Tabloda istenilen sayıda satır sayısını gösterme

--workers01 tablosunda en yüksek ücreti olan işçinin tüm bilgilerini gösteren kodu yazınız

SELECT \*

FROM workers01

WHERE ROWNUM = 1; -- sadece 1 satır göster demek

ORDER BY salary DESC

--workers01 tablosunda en yüksek ücreti olan işçinin tüm bilgilerini gösteren kodu yazınız

SELECT \*

FROM workers01

WHERE ROWNUM = 1; -- sadece 1 satır göster demek

ORDER BY salary DESC

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

--en yüksek ücreti alan işçinin tüm bilgilerini gösteren kodu yazınız.(SUBQUERY)

SELECT \*

FROM workers01

WHERE salary = (SELECT MAX(salary) AS Max\_salary

FROM Workers01

);

--en yüksek ücreti alan işçinin tüm bilgilerini gösteren kodu yazınız.(SUBQUERY)

SELECT \*

FROM workers01

WHERE salary = (SELECT MAX(salary) AS Max\_salary

FROM Workers01

);

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

--ikinci en yüksek maaşı alan işçinin tüm bilgilerin gösteren kodu yazınız

SELECT MAX(salary)

FROM workers01

WHERE salary < (SELECT MAX(salary)

FROM workers01

);

-- < select max(salary) demek en büyükten küçük olanlar demek bunların içinden

-- max olanı göster diyor.

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

--ikinci en düşük maaşı gösteren kodu yazınız

SELECT MIN(salary)

FROM workers

WHERE salary > (SELECT MIN(salary)

FROM workers01

);

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

# DAY 9

--Salary'si en yuksek olan isci disindaki iscilerin tum bilgilerini

--salary'e gore buyukten kucuge dizerek gosteren kodu yaziniz.

CREATE TABLE employees

(

id number(9),

name varchar2(50),

state varchar2(50),

salary number(20),

company varchar2(20)

);

INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');

INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');

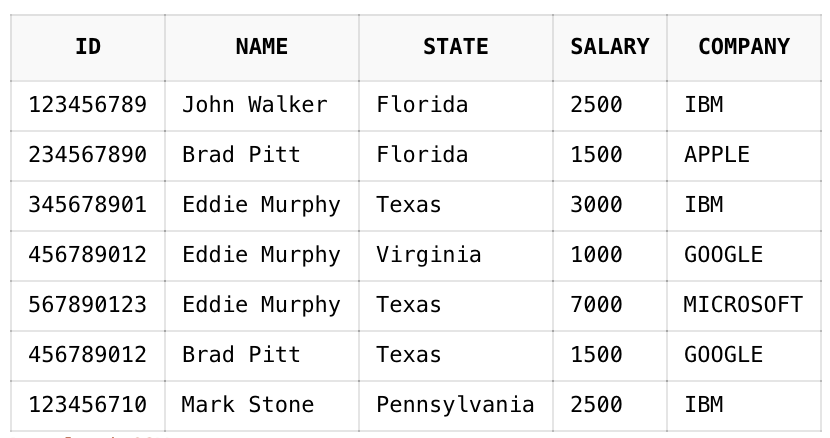
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');

INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');

INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');

INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');

INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');



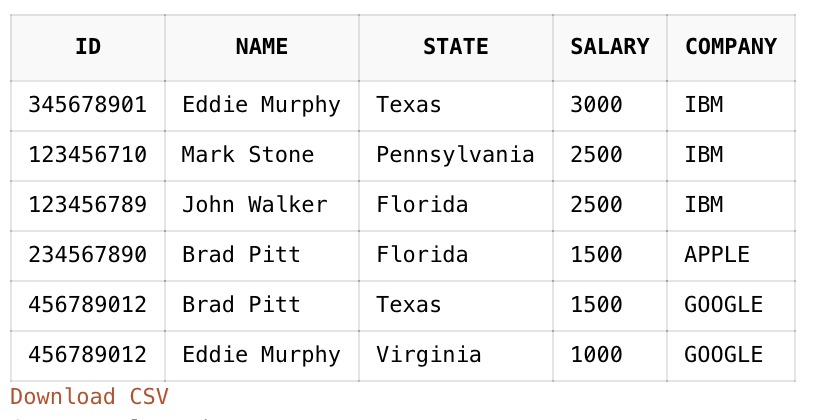
SELECT \*

FROM employees

WHERE salary != (SELECT MAX(salary)

FROM employees)

ORDER BY salary DESC;



--ikinci en yüksek maaş alan kişinin tüm bilgilerini gösteren kodu yazınız.

SELECT \*

FROM (SELECT \*

FROM employees

WHERE salary != (SELECT MAX(salary)

FROM employees)

ORDER BY salary DESC)

WHERE ROWNUM = 1;

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

--2.Yol

SELECT \*

FROM employees

ORDER BY salary DESC

OFFSET 1 ROW -- 1.satırı göremezden gel

FETCH NEXT 1 ROW ONLY; -- 1 satır sonrakini getir sadece

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

CREATE TABLE workers

(

id number(2),

name varchar2(20),

title varchar2(60),

boss\_id number(2)

);

INSERT INTO workers VALUES(1, 'Ali Can', 'SDET', 2);

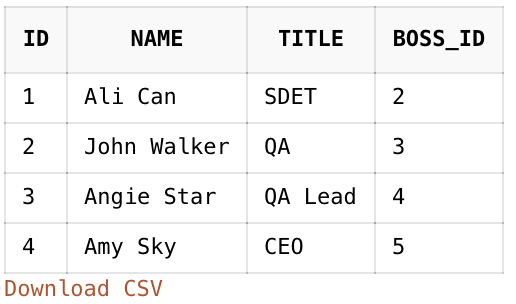
INSERT INTO workers VALUES(2, 'John Walker', 'QA', 3);

INSERT INTO workers VALUES(3, 'Angie Star', 'QA Lead', 4);

INSERT INTO workers VALUES(4, 'Amy Sky', 'CEO', 5);

SELECT \*

FROM workers;



## SELF JOIN

SELF JOIN iki tablo ile çalışır. Bir tabloyu iki tablo gibi düşündük birine w1 diğerine de w2 dedik.

--Her işçinin patronunu gösteren tabloyu oluşturunuz

SELECT w1.name AS worker\_name, w2.name AS boss\_name --w1 tablosundan name'i al worker\_name olarak adlandır

-- w2 tablosundan name'i al boss\_name olarak adlandır.

FROM workers w1 INNER JOIN workers w2 --burada w1 ve w2 adında aynı iki tablo oluşturmuş olduk.

ON w1.boss\_id = w2.id; -- w1 deki boss\_id w2 deki id'ye eşit olsun.

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

## UPPER: Bir sütundaki dataların büyük harfle yazılmasını sağlar.

## LOWER: Bir sütundaki dataların küçük harfle yazılmasını sağlar.

## INITCAP: Bir sütundaki dataların ilk harfini büyük yapar diğerlerini küçük.

SELECT INITCAP(name), UPPER(state), LOWER(company)

FROM employees;

