5.3 – Working with Data in Databases

## **5.3.1 – Learn SQL with Mode Analytics**

|  |  |
| --- | --- |
| Query clause order | 1. SELECT 2. FROM 3. WHERE 4. GROUP BY 5. HAVING 6. ORDER BY |
| Basic SQL |  |
| **SELECT** |  |
| **LIMIT** | **SELECT** \* **FROM** tutorial.us\_housing\_units **LIMIT** 100 |
| **WHERE** | **SELECT** \* **FROM** tutorial.us\_housing\_units **WHERE** **month** = 1 |
| **LIKE** | **SELECT** \* **FROM** file **WHERE** "group" **LIKE** 'Snoop%'  ## “%” 🡪 wildcard  ## **ILIKE** 🡪 to ignore case |
| **IN** | **SELECT** \* **FROM** file **WHERE** artist **IN** ('Taylor', 'Usher', 'Ludacris') |
| **BETWEEN** | **SELECT** \* **FROM** file **WHERE** year\_rank **BETWEEN** 5 **AND** 10 |
| **IS NULL** | **SELECT** \* **FROM** file **WHERE** artist **IS** **NULL** |
| **AND** | **SELECT** \* **FROM** file **WHERE** **year** = 2012 **AND** year\_rank <= 10 |
| **OR** | **SELECT** \* **FROM** file **WHERE** year\_rank = 5 **OR** artist = 'Gotye' |
| **NOT** | **SELECT** \* **FROM** file **WHERE** **year** = 2013 **AND** year\_rank **NOT** **BETWEEN** 2 **AND** 3 |
| **ORDER BY** | **SELECT** \* **FROM** file **ORDER BY** artist |
| Intermediate SQL |  |
| **COUNT** | SELECT **COUNT**(high) FROM tutorial.aapl\_historical\_stock\_price  ## simply counts the total number of non-null rows, not the distinct values. |
| **SUM** | Numerik sutunlarin satirlarini toplar, NULL’lari sifir olarak alir |
| **MIN/MAX** | Non-numerik verilerde de kullanilabilir, “WHERE” ile birlikte kullanilamazlar |
| **AVG** | Numerik veriler icindir, NULL verileri dikkate almaz (sifir olarak algilamasini istiyor isek, belirtmeliyiz) |
| **GROUP BY** | SELECT year, month, COUNT(\*) AS count FROM tutorial.aapl\_historical\_stock\_price **GROUP BY** year, month |
| **HAVING** | SELECT year, month, MAX(high) AS month\_high FROM tutorial.aapl\_historical\_stock\_price GROUP BY year, month **HAVING** MAX(high) > 400 ORDER BY year, month  ## “clean” way to filter a query that has been aggregated, but this is also commonly done using a subquery  ## “WHERE “aggregated” veriler uzerinde kullanilamadigindan, “HAVING” kullanilir |
| **DISTINCT** | SELECT year, COUNT(**DISTINCT** month) AS months\_count FROM file.xxx GROUP BY year ORDER BY year  ## viewing for unique values |
| **CASE** | ## always goes in the SELECT clause // must include the following components: WHEN, THEN, and END. /// ELSE is an // You can make any conditional statement using any conditional operator  # SELECT **CASE** WHEN year = 'FR' THEN 'FR'  WHEN year = 'SO' THEN 'SO'  ELSE 'No Year Data' END AS year\_group,  COUNT(1) AS count  FROM benn.college\_football\_players  GROUP BY year\_group  # SELECT COUNT(**CASE** WHEN year = 'FR' THEN 1 ELSE NULL END) AS fr\_count,  COUNT(**CASE** WHEN year = 'SO' THEN 1 ELSE NULL END) AS so\_count,  FROM benn.college\_football\_players |
| (INNER) **JOIN**  **LEFT** (OUTER) **JOIN**  **RIGHT** (OUTER) **JOIN**  **FULL** (OUTER) **JOIN** | SELECT companies.state\_code,  COUNT(DISTINCT companies.permalink) AS unique\_companies,  COUNT(DISTINCT acquisitions.company\_permalink) AS unique\_companies\_acquired  FROM tutorial.crunchbase\_companies companies  LEFT JOIN tutorial.crunchbase\_acquisitions acquisitions  ON companies.permalink = acquisitions.company\_permalink  WHERE companies.state\_code IS NOT NULL  GROUP BY 1  ORDER BY 3 DESC |
| WHERE-AND  # Buradaki “AND”, joining’den once ve sadece ilgili tabloda gerceklesir  **FROM** tutorial.crunchbase\_companies companies  **LEFT** **JOIN** tutorial.crunchbase\_acquisitions acquisitions  **ON** companies.permalink = acquisitions.company\_permalink  **AND** acquisitions.company\_permalink '/company/1000memories'  ## ayni seyi “WHERE” ile yaparsak, islem joining’den sonra ve birlesik tablo uzerinde gerceklesir.  **WHERE** acquisitions.company\_permalink != '/company/1000memories' |
|  | Comparison Operators  ## “Comparison Operators”, WHERE” ve “AND” icersinde kullanilabilir |
|  | Multiple Keys  **LEFT** **JOIN** tutorial.crunchbase\_investments\_part1 investments  **ON** companies.permalink = investments.company\_permalink  **AND** companies.name = investments.company\_name  ## birlesme kosullarini artirmak, dogruluk ve hizi artirir |
| **UNION** | ## Both tables must have the same number of columns  ## The columns must have the same data types in the same order as the first table  **SELECT** \*  **FROM** tutorial.crunchbase\_investments\_part1  **UNION** ALL  **SELECT** \*  **FROM** tutorial.crunchbase\_investments\_part2 |
| ADVANCED SQL |  |
| Data Types |  |
| **CAST** / **::** | **CAST**(column\_name AS integer) and column\_name::integer |
| DATE FORMAT | Arithmetics  **SELECT** acquisitions.acquired\_at\_cleaned -  companies.founded\_at\_clean::**timestamp** **AS** time\_to\_acquisition  INTERVAL: introduce intervals using the function  **SELECT** companies.founded\_at\_clean::**timestamp** +  INTERVAL '1 week' **AS** plus\_one\_week  NOW()  **SELECT** companies.permalink,  companies.founded\_at\_clean,  **NOW**() - companies.founded\_at\_clean::**timestamp** **AS** founded\_time\_ago |
| String Functions to Clean Data | |
| **LEFT/RIGHT** | to pull a certain number of characters from the specified side  **LEFT**(string, number of characters) |
| **SUBSTR** | If you want to start in the middle of a string, you can use SUBSTR  **SUBSTR**(date, 4, 2) **AS** **day** |
| **LENGHT** | **SELECT** incidnt\_num,  date,  **LEFT**(date, 10) **AS** cleaned\_date,  **RIGHT**(date, **LENGTH**(date) - 11) **AS** cleaned\_time  **FROM** tutorial.sf\_crime\_incidents\_2014\_01 |
| **TRIM** | The TRIM function takes 3 arguments. First, you have to specify whether you want to remove characters from the beginning (‘leading’), the end (‘trailing’), or both (‘both’, as used above). Next you must specify all characters to be trimmed.  **SELECT** location,  **TRIM**(**both** '()' **FROM** location)  **FROM** tutorial.sf\_crime\_incidents\_2014\_01 |
| **POSITION**  **STRPOS** | ## allows you to specify a substring, then returns a numerical value equal to the character number (from left)  **SELECT** incidnt\_num,  descript,  **POSITION**('A' **IN** descript) **AS** a\_position  **FROM** tutorial.sf\_crime\_incidents\_2014\_01  ## to achive the same result;  **STRPOS**(descript, 'A') **AS** a\_position |
| **CONCAT** | **CONCAT**(day\_of\_week, ', ', **LEFT**(date, 10)) **AS** day\_and\_date  using the **||** syntax instead of CONCAT  day\_of\_week || ', ' || **LEFT**(date, 10) **AS** day\_and\_date |
| **UPPER/LOWER** | **UPPER**(address) **AS** address\_upper,  **LOWER**(address) **AS** address\_lower |
| Strings to dates | CAST/::  (**SUBSTR**(date, 7, 4) || '-' || **LEFT**(date, 2) || '-' || **SUBSTR**(date, 4, 2))**::**date **AS** cleaned\_date |
| **EXTRACT** | **SELECT** cleaned\_date,  **EXTRACT**('year' **FROM** cleaned\_date) **AS** **year**,  **FROM** tutorial.sf\_crime\_incidents\_cleandate  ## 'month'/'day'/'hour'/'minute'/'second'/'decade'/'dow' |
| **DATE\_TRUNC** | ## rounds a date to whatever precision you specify  **SELECT** cleaned\_date,  **DATE\_TRUNC**('day' , cleaned\_date) **AS** **day**,  **FROM** tutorial.sf\_crime\_incidents\_cleandate |
| Coordinated Universal Time (UTC) | **SELECT** **CURRENT\_DATE** **AS** date,  **CURRENT\_TIME** **AS** **time**,  **CURRENT\_TIMESTAMP** **AS** **timestamp**,  **LOCALTIME** **AS** **localtime**,  **LOCALTIMESTAMP** **AS** **localtimestamp**,  **NOW**() **AS** **now** |
| **AT TIME ZONE** | **SELECT** **CURRENT\_TIME** **AS** **time**,  **CURRENT\_TIME** **AT TIME ZONE** 'EDT' **AS** time\_pst  ## EDT 🡪 Eastern Daylight Time |
| **COALESCE** | to replace the null values  **SELECT** incidnt\_num,  descript,  **COALESCE**(descript, 'No Description')  **FROM** tutorial.sf\_crime\_incidents\_cleandate  **ORDER** **BY** descript **DESC** |
| Writing Subqueries in SQL | |
| FROM statement | **SELECT** **LEFT**(sub.date, 2) **AS** cleaned\_month,  sub.day\_of\_week,  **AVG**(sub.incidents) **AS** average\_incidents  **FROM** (  **SELECT** day\_of\_week,  date,  **COUNT**(incidnt\_num) **AS** incidents  **FROM** tutorial.sf\_crime\_incidents\_2014\_01  **GROUP** **BY** 1,2  ) **sub**  **GROUP** **BY** 1,2  **ORDER** **BY** 1,2  ## subquery’i |
| in conditional logic | ## You can use subqueries in conditional logic (in conjunction with WHERE, JOIN/ON, or CASE)  ## IN is the only type of conditional logic that will work when the inner query contains multiple results  ## you should not include an alias, because it is treated as an individual value (or set of values in the IN case  **SELECT** \*  **FROM** tutorial.sf\_crime\_incidents\_2014\_01  **WHERE** Date **IN** (**SELECT** date  **FROM** tutorial.sf\_crime\_incidents\_2014\_01  **ORDER** **BY** date  **LIMIT** 5) |
| Joining | **SELECT** incidents.\*,  sub.incidents **AS** incidents\_that\_day  **FROM** tutorial.sf\_crime\_incidents\_2014\_01 incidents  **JOIN** ( **SELECT** date,  **COUNT**(incidnt\_num) **AS** incidents  **FROM** tutorial.sf\_crime\_incidents\_2014\_01  **GROUP** **BY** 1  ) sub  **ON** incidents.date = sub.date  **ORDER** **BY** sub.incidents **DESC**, **time** |
| Union | **SELECT** **COUNT**(\*) **AS** total\_rows  **FROM** (  **SELECT** \*  **FROM** tutorial.crunchbase\_investments\_part1  **UNION** ALL  **SELECT** \*  **FROM** tutorial.crunchbase\_investments\_part2) sub |
| SQL Window Functions | |
| **OVER**  **(PARTITION BY ..)** | **SELECT** start\_terminal,  duration\_seconds,  **SUM**(duration\_seconds) **OVER**  (**PARTITION** **BY** start\_terminal **ORDER** **BY** start\_time)  **AS** running\_total  **FROM** tutorial.dc\_bikeshare\_q1\_2012  **WHERE** start\_time < '2012-01-08' |
| **ROW NUMBER( )** | **ROW**\_**NUMBER()** **OVER** (**ORDER** **BY** start\_time)  **AS** row\_number |
| **RANK ( )**  **DENSE\_RANK ( )** | **RANK**() **OVER** (**PARTITION** **BY** start\_terminal  **ORDER** **BY** start\_time)  ## RANK ( ). ; ayni degerlere ayni sira numarasini verir, sira atlayip devam eder (1,2,2,4, ..gibi)  ## DENSE\_RANK( ); rank’den farkli olarak sira atlamaz. (1,2,2,3 gibi) |
| **NTILE( )** | **SELECT** …  **NTILE**(4) **OVER**  (**PARTITION** **BY** start\_terminal **ORDER** **BY** duration\_seconds)  **AS** quartile,  ## basket sayisini belirler, “partition by” penceresini basket sayisina bolup, basket numarasini verir. |
| **LAG/LEAD** | **SELECT** …  LAG(duration\_seconds, 1) **OVER**  (**PARTITION** **BY** start\_terminal **ORDER** **BY** duration\_seconds) **AS** lag,  **LEAD**(duration\_seconds, 1) **OVER**  (**PARTITION** **BY** start\_terminal **ORDER** **BY** duration\_seconds) **AS** **lead**  ## istenilen sutundan, onceki (LAG) veya sonraki (LEAD) satirlardan bilgi ceker ve ayri bir sutunda gosterir. |
| Defining a window alias | **SELECT** start\_terminal,  duration\_seconds,  NTILE(4) **OVER** ntile\_window **AS** quartile,  NTILE(5) **OVER** ntile\_window **AS** quintile,  NTILE(100) **OVER** ntile\_window **AS** percentile  **FROM** tutorial.dc\_bikeshare\_q1\_2012  **WHERE** start\_time < '2012-01-08'  WINDOW ntile\_window **AS**  (**PARTITION** **BY** start\_terminal **ORDER** **BY** duration\_seconds)  **ORDER** **BY** start\_terminal, duration\_seconds |
| Performance Tuning SQL Queries | |
| EXPLAIN | it shows the order in which your query will be executed |
| Pivoting Data in SQL | |
| Pivoting rows to columns | **SUM**(**CASE** **WHEN** **year** = 'FR' **THEN** players **ELSE** NULL **END**) **AS** fr,  **SUM**(**CASE** **WHEN** **year** = 'SO' **THEN** players **ELSE** NULL **END**) **AS** so,  **SUM**(**CASE** **WHEN** **year** = 'JR' **THEN** players **ELSE** NULL **END**) **AS** jr,  **SUM**(**CASE** **WHEN** **year** = 'SR' **THEN** players **ELSE** NULL **END**) **AS** sr |
|  |  |
|  |  |
|  |  |
|  |  |

mysql-for-data-analytics-and-business-intelligence

Udemy Course

|  |  |
| --- | --- |
| Statement | DDL (Data Defination Language) |
| CREATE | # used for creating entire databases and database objects as tables  **CREATE** **object\_type** **object\_name;**  **CREATE TABLE** **object\_name** (column\_name data\_type);  Exp: CREATE TABLE sales (purchase\_number INT); |
| ALTER  - ADD  - REMOVE  - RENAME  - CHANGE  - MODIFY | # used when altering existing objects  ALTER TABLE sales  ADD COLUMN date\_of\_purchase DATE; |
| DROP | # used for deleting a database object  **DROP** **object\_type** **object\_name**;  DROP TABLE customers; |
| RENAME | # allows you to rename an object  **RENAME** **object\_type** **object\_name** **TO** **new\_object\_name**;  RENAME TABLE customers TO customer\_data; |
| TRUNCATE | # instead of deleting an entire table through DROP, we can also remove its data and continue to have the table as an object in the database  **TRUNCATE** **object\_type** **object\_name**;  TRUNCATE TABLE customers; |
|  | Keywords: ADD, CREATE, ALTER etc.  # objects or databases cannot have names that coincide with SQL keywords |
|  | DML (Data Manipulation Language)  # its statements allow us to manipulate the data in the tables of a database |
| SELECT | # used to retrieve data from database objects, like tables  **SELECT** **FROM** sales; |
| INSERT | # used to insert data into tables  **INSERT** **INTO object\_name** (column\_1, column\_2, …, column\_n)  **VALUES** (value\_1, value\_2, ..., value\_n);  INSERT INTO sales (purchase\_number, date\_of\_purchase) VALUES (2, ‘2017-10-27’);  INSERT INTO sales VALUES (1, ‘2017-10-11’);  #tum columnlari yasmak zorunda degiliz, ama yazdigimiz degerlerin, belirttigimiz column sirasina gore girilecegini unutmamaliyiz. Ayni sayida ve ayni sirada giriyoruz degerleri.  # rakam disindaki tum “value”lar tirnak icersinde girilmelidir.  Inserting data into a new table:  INSERT INTO table\_2 (column\_1, column\_2, …, column\_n)  SELECT column\_1, column\_2, …, column\_n FROM table\_1  WHERE condition;  # burada “values” bolumunun yerine SELECT .. FROM.. aliyor, yani hazir olan data’yi value olarak assign ediyoruz. |
| UPDATE | # allows you to renew existing data of your tables  **UPDATE** table\_name **SET** column\_1 = value\_1, column\_2 = value\_2 …  WHERE conditions;  **UPDATE** sales **SET** date\_of\_purchase = ‘2017-12-12’ **WHERE** purchase\_number = 1;  # WHERE kullanilmaz ise butun satirlari update eder |
| DELETE | # TRUNCATE allows us to remove all the records contained in a table  # with DELETE, you can specify precisely what you would like to be removed  **DELETE** **FROM** table\_name **WHERE** conditions;  **DELETE** **FROM** sales **WHERE** purchase\_number = 1;  DROP: Sutun veya tabloyu tamamen ortadan kadirir.  TRUNCATE: icini bosaltir, value’larin tamamini siler, “auto-increment” value’lar silinir. Yani 1-10 arasi silindi ise 11, 1’e donusur.  DELETE: value’lari satir satir siler. “auto-increment” value’lar silinmez. |
|  | DCL (Data Control Language) |
| GRANT | # gives (or grants) certain permissions to users. one can grant a specific type of permission, like complete or partial access  **GRANT** **type\_of\_permission** **ON** **database\_name.table\_name** **TO** ‘username’@’localhost’ |
| REVOKE | # used to revoke permissions and privileges of database users  **REVOKE** **type\_of\_permission** **ON database\_name.table\_name** **FROM** ‘username’@’localhost’ |
|  | TCL (Transaction Control Language) |
| COMMIT | - related to INSERT, DELETE, UPDATE  - will save the changes you’ve made  - will let other users have access to the modified version of the database  UPDATE customers SET last\_name = ‘Johnson’ WHERE customer\_id = 4 **COMMIT**; |
| ROLLBACK | # the clause that will let you make a step back  UPDATE customers SET last\_name = ‘Johnson’ WHERE customer\_id = 4 COMMIT;  **ROLLBACK**; |
|  | Basic Database Terminology |
|  | Relational Schemas: |
| Primary Key | a column (or a set of columns) whose value exists and is unique for every record in a table is called a primary key  each table can have one and only one primary key  may be composed of a set of columns. e.g. “purchase\_number” + “date\_of\_purchase” (ayni tarihin birden fazla gorunmesi mumkun)  primary keys are the unique identifiers of a table  cannot contain null values! |
| Foreign Key | identifies the relationships between tables, not the tables themselves  always look for the foreign keys, as they show us where the relations are  no repeating and missing values (unique values only) |
| Unique Key & Null Values | used whenever you would like to specify that you don’t want to see duplicate data in a given field  # telefon numarasi gibi, birden fazla tekrar etmesi istenmeyen degerlerdir, bir tabloda birden fazla “unique key” olabilir, ornegin hem telefon hem de adres bilgisi. |
| Relationship | # relationships tell you how much of the data from a foreign key field can be seen in the primary key column of the table the data is related to and vice versa  # types of relationships:  - one-to-many (many-to-one)  - one-to-one  - many-to-many |
| Relational schemas | = blueprints, or a plan for a database  - represent the concept database administrators must implement  - depict how a database is organized  - will help you immensely while writing your queries! |
|  | First Steps in SQL |
|  | Creating a Database |
|  | CREATE DATABASE [IF NOT EXISTS] database\_name;  CREATE DATABASE: creates a database as an abstract unit  [IF NOT EXISTS]: verifies if a database with the same name exists already  database\_name: - the SQL code is not case sensitive  ; (the semicolon character): it functions as a statement terminator  - when your code contains more than a single statement, ; is indispensable  - will help you avoid errors sometimes  - will improve the readability of your code |
|  | Database sembolu |
|  | “i” feature: gives information about the database |
|  | USE Sales; |
|  | Data Types |
| introduction | length: a measure used to indicate how many symbols a certain string has  size: indicates the memory space used by a data type. -measured in bytes  1 byte ~ 1 symbol  storage: the physical space in the computer drive’s memory, where the data is being saved or stored |
| string | = a variable of the alphanumeric data type   |  |  | | --- | --- | |  | Char, varchar’dan 50% daha hizli calisir. | | “5” max sembol sayisini belirtir. Fixed storage (her halukarda 5 byte’lik yer kaplar. “Size” en fazla 255 “symbol” yapilabilir. | | “5” max sembol sayisini belirtir. variable storage (girilen symbol kadar yer kaplar. “Size” en fazla 65,535 “symbol” yapilabilir. | | Belirtilen degerler disinda baska bir deger kabul etmez, hata verir | |
| integers | |  |  | | --- | --- | | Integers: whole numbers with no decimal point e.g. 5; -200;  signed: if the encompassed range includes both positive and negative values (default one)  unsigned: if integers are allowed to be only positive  DECIMAL = NUMERIC |  | |
| Fixed and Floating Data Types | |  |  | | --- | --- | |  |  |   precision: refers to the number of digits in a number  scale: refers to the number of digits to the right of the decimal point in a number  fixed-point data: eksik decimaller “0” ile tamamlanir, fazla olursa yuvarlar (10.5🡪 10.500 // 10.52367 🡪 10.524), yuvarlama yaptiginda ikaz verir.  Floating-point data: yuvarlamada ikaz vermez |
| Date | DATE : YYYY-MM-DD 🡪 date without time 🡪 ‘2018-07-25’  DATETIME : YYYY-MM-DD HH:MM:SS [.fraction] (0 – 23:59:59.999999)  🡪 ‘2018-07-25 9:30:00’  TIMESTAMP : the number of seconds since 00:00:00 UTC on January 1, 1970  ‘2018-07-25 10:30:00’ UTC. (25th of July 2018 🡪 1,535,155,200)  Saat farki otomatik olarak hesaplanir. |
| BLOB | Binary Large Object  - refers to a file of binary data – data with 1s and 0s  - involves saving files in a record |
|  |  |
|  | Creating a Table |
|  | CREATE TABLE sales;  DROP TABLE sales; |
|  | |  |  | | --- | --- | | CREATE TABLE table\_name  (  column\_1 data\_type constraints  column\_2 data\_type constraints,  …  column\_n data\_type constraints  ); | create table sales  (  purchase\_number INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,  date\_of\_purchase DATE NOT NULL,  custmer\_id INT,  item\_code varchar(10) NOT NULL  ); |   # alternative: PRIMARY KEY (purchase\_number) [son satir olarak]  # auto\_increment: frees you from having to insert all purchase numbers manually through the INSERT command at a later stage. - assigns 1 to the first record of the table and automatically increments by 1 for every subsequent row |
|  | Using Databases and Tables |
|  | query:  a command you write in SQL with the idea of either retrieving information from the database on which you are working, or, alternatively, to insert, update, or delete data from it. it is a representation of a complete logical thought  Whenever you would like to refer to an SQL object in your queries, you must specify the database to which it is applied  SQL objects:  - SQL table  - views  - stored procedures  - functions  set a default database: USE sales; 🡪 SELECT \* FROM customers;  call a table from a certain database: database\_object . sql\_object  . – “dot operator”: signals the existence of a connection between the two object types |
|  | MySQL Constraints |
|  | constraints:   * specific rules, or limits, that we define in our tables. * the role of constraints is to outline the existing relationships between different tables in our database |
| PRIMARY KEY |  |
| FOREIGH KEY | points to a column of another table and, thus, links the two tables. the foreign key maintains the referential integrity within the database.  child table = referencing table / parent table = referenced table  # “foreigh key”in diger tablodaki isminin farkli olmasi onemli degil, onemli olan data type’in ve icindeki bilginin ayni olmasi.  # “foreigh key’in,”primary key” oldugu table’daki bilgi silindiginde, ana tablodaki bilgide silinir.  1.yol:  CREATE TABLE sales  (  purchase\_number INT AUTO\_INCREMENT,  date\_of\_purchase DATE,  customer\_id INT,  item\_code VARCHAR(10),  PRIMARY KEY (purchase\_number),  **FOREIGN KEY** (customer\_id) **REFERENCES** customers(customer\_id) **ON DELETE CASCADE**  );  2.yol:  **ALTER TABLE** sales  **ADD** **FOREIGN KEY** (customer\_id) **REFERENCES** customers(customer\_id) **ON DELETE CASCADE**  Foreign key kaldirma:  DDL sekmesinde “foreign key” bilgisi onunde gorunen ‘sales\_ibfk\_1’ benzeri gorunen bilgi kopyalanir  **ALTER TABLE** sales  **DROP** FOREIGN KEY sales\_ibfk\_1;  # ‘ON DELETE CASCADE’ ilgili tabloda silinen bilginin ayni sekilde ana tabloda da silinmesini saglamak icin.  # Alter Table’a ilgili table’a sag klik yapildiginda cikan menuden de ulasilabilir.  Mesela Alttaki pencerede “Foreign Keys” sekmesine basinca acilan kutucuga “Foreign Key Name” girilebiliyor. Buraya verilen isim “column”a verilmis olmuyor, sadece “foreign key” isimlendirilmis oluyor. Ayni yerde “Referenced Table”, “Column” ve “Referenced Column” bilgilerinin de girilmesi gerekiyor. Hatta “on delete cascade” bile eklenebilir buradan. “Apply” ve “Finish” diyerek islemleri tamamliyoruz.  **Checking:**  1.yol: Bunu “i-ikonu”na bastigimizda gelen pencereden “DDL-tab”e basarak da check edebiliriz. Burada belirttigimiz column’in Foreign Key olarak atandigini goruruz.  2.yol: ayni pencerede “Foreign Keys” tab’i de var. buradan da tum bilgileri kontrol etmek mumkun. |
| UNIQUE KEY | used whenever you would like to specify that you don’t want to see duplicate data in a given field. ensures that all values in a column (or a set of columns) are different. Index gibi fonksiyon gorurler  # bu constraint de digerleri gibi blok sonuna  **UNIQUE KEY** (column\_name) seklinde eklenebilir ya da “ALTER TABLE” kullanilir.  ALTER TABLE table\_name  DROP INDEX unique\_key\_field; |
| DEFAULT | helps us assign a particular default value to every row of a column  …. number\_of\_complaints INT DEFAULT 0, ….. |
| NOT NULL | when you insert values in the table, you cannot leave the respective field empty  Don’t confuse a NULL value with the value of 0 or with a “NONE” response!  Think of a null value as a missing value.  0, NONE: assigned by the user  NULL: assigned by the computer, missing value. |
|  | SQL Best Practices |
|  | Coding Techniques and Best Practices |
|  | complying with coding style is crucial  clean code: code that is focused and understandable, which means it must be readable, logical, and changeable  good code is not the one computers understand; it is the one humans can understand  good practice implies you will choose the version that will be easiest to read and understandassumption:at your workplace, you will always type code cleanly –  as simple as possible, perfectly organized, maintaining a steady logical flow  isimlendirme: her zaman kisa ve anlamli, telaffuzu kolay isimler verilmeli  en onemlisi, kod butunlugu  /\* … \*/. 🡪 uzun yorumlar icin kullanilir.  # or -- 🡪 bir satirlik komutlar icin kullanilir. |
|  | Data Manipulation: The SQL SELECT Statement |
|  | SELECT… FROM… |
| the SELECT statement | **SELECT** column\_1, column\_2,… column\_n **FROM** **table\_name**;  SELECT DISTINCT (reserved word): selects all distinct, different data values  SELECT DISTINCT column\_1, column\_2,… column\_n FROM table\_name; |
| the WHERE clause | it will allow us to set a condition upon which we will specify what part of the data we want to retrieve from the database  SELECT column\_1, column\_2,… column\_n FROM table\_name **WHERE** **condition**; |
| operators | in SQL, there are many other linking keywords and symbols, called operators, that you can use with the WHERE clause     |  |  | | --- | --- | | AND | … WHERE condition\_1 AND condition\_2;  # conditions set on different columns # AND > OR | | OR | # conditions set on the same column | | IN / NOT IN | # Kendisinden sonraki parantez icersinde belirtilen ifadeleri iceren verileri getirir. OR … OR .. demek yerine “IN” kullanabiliriz  ….WHERE first\_name **IN** (‘Cathie’, ‘Mark’, ‘Nathan’)  # “**NOT IN**” dedigimiz de bu isimleri icermeyen tum kayitlati getirirler. | | LIKE |  | | BETWEEN… AND… | # designate the interval to which a given value belongs  …WHERE hire\_date **BETWEEN** '1990-01-01' **AND** '2000-01-01';  # both values will be included | | NOT BETWEEN… AND… | # an interval composed of two parts  # both values will not be included in the interval. | | IS NOT NULL | # used to extract values that are not null  … WHERE column\_name **IS NOT NULL**; | | IS NULL | # used to extract values that are null | | Comparison Operators | = / > / >= / < / < =  < > / != 🡪 not equal, different from | |  |  | |
| Pattern matching with wildcards | |  |  | | --- | --- | |  | limitsiz “wildcard” | |  | bir karakterlik “wildcard” | |  |  | |
| aggregate functions = summarizing functions | COUNT( ) / SUM( ) / MIN( ) / MAX( ) / AVG( )  SELECT COUNT(column\_name) FROM table\_name; (‘count’dan sonra bosluk yok)  SELECT COUNT(DISTINCT column\_name) FROM table\_name;  # COUNT, numeric ve non-numeric value’larin her ikisi icinde uygulanabilir. Diger tum “aggregate function”lar sadece numeric data’lar icin kullanilir.  # DISTINCT, unique value sayisini bulmak icin kullanilir  # aggregate functions’lar “null” value’lari dikkate almazlar.  # COUNT(\*) seklinde kullanilir ise “NULL” value’lari da sayar. |
| ORDER BY | SELECT \* FROM employees ORDER BY first\_name, last\_name DESC;  ASC: ascending, A once gelir  DESC: descending, Z once gelir |
| GROUP BY | - WHERE’den hemen sonra, (varsa) ORDER BY’dan once gelir.  - Cogunlukla “aggregate function” kullanildiginda “GROUP BY” da kullanilmalidir  - GROUP BY ile kullandiginin degisken mutlaka “SELECT statement”da yer almalidir.  SELECT column\_name(s) FROM table\_name  WHERE conditions  GROUP BY column\_name(s) ORDER BY column\_name(s);  # dikkat edilmesi gereken hususlardan biri, GROUP BY distinct value verir, 10,000 adet usimde 1000 farkli isim vermesi gibi. ORDER BY, 10,000 isim verir ama sirasini “azalan” ya da “artan” sekilde duzenleyerek getirilmesini saglar. |
| HAVING | # output’un rafine edilmesi icin ve genellikle GROUP BY blogu ile birlikte kullanilir.  # WHERE conditionslari “aggregate functions” icermez ama HAVING icerebilir.  # WHERE’de belirtilen ‘condition’lar “output’larin gruplandirilmasindan once gerceklesir.  # “HAVING clause” icinde hem “aggregated” hem de “non-aggregated” contidition ayni anda bulunamaz. Ornegin “hire\_date > … and count(first\_name)>250” birarada olmaz.  Aggregate functions – GROUP BY and HAVING  General conditions – WHERE  SELECT column\_name(s) FROM table\_name  WHERE conditions  GROUP BY column\_name(s)  HAVING conditions  ORDER BY column\_name(s); |
| LIMIT | … LIMIT number; |
|  | The SQL Statements |
| ROUND( ) | ROUND(#,decimal\_places)  .. ROUND(AVG(salary)) 🡪 decimal birakmaz  .. ROUND(AVG(salary),2) 🡪 2 decimal’e yuvarlar  #- usually applied to the single values that aggregate functions return |
| ADD COLUMN | ALTER TABLE customers  ADD COLUMN gender ENUM('M','F') AFTER last\_name; |
| CHANGE | Default ekleme ornegi:  ALTER TABLE customers  CHANGE COLUMN number\_of\_complaints number\_of\_complaints INT DEFAULT 0;  Default’u kaldirma ornegi:  ALTER TABLE customers  ALTER COLUMN number\_of\_complaints DROP DEFAULT;  # bu gunluk hayatta yasanabilecek bir durum. Sonradan bir degisiklik yapmak istedik. |
| Adding constraint | Bir column’in “constraint” eklemek istedim sonradan:  ALTER TABLE companies  CHANGE COLUMN company\_name company\_name VARCHAR(255) NOT NULL;  # “COLUMN” ifadesinin ardindan hangi column oldugunu da yaziyoruz, bu ikili ayrilmaz olarak dusunebiliriz. Sonrasinda adini tekrar yaziyoruz gibi gorunse de, aslinda yeniden tanimliyoruz ve tanimla sirasinda ismini de degistirebiliriz. Iki defa arka arkaya yazilmasi gozu tirmalasa da, bir sebebe binaen oldugu hatirlanmali. |
| Remove constraint |  |
| MODIFY | # bu statement ile hem “data type” hem de “constraint” eklenebilir veya degistirilebilir. Bu statement ile birlikte “column name”in bir kez kullanilmasi yeterlidir (COLUMN statement’i ekurisi ile yani “column name” ile birlikte gelir).  ALTER TABLE companies  MODIFY company\_name VARCHAR(255) NULL; |
| Coding technics | * = (beautify option) # yazilan kodun dikey olarak yeniden organize edilmesini sagliyor * indentation: “TAB” kullanarak “data type” ve “constraints”leri ayni hizaya getirebiliriz. * comments: “/\* … \*/” (long comments). “# or –” (short comments) * (lightining icon) =  : tum kodu calistirir * = : sadece curser’in bulundugu blogu calistirir. |
| Limit | Kod blogunun en sonuna: …. LIMIT 10;  Limit kaldirma:  Edit 🡪 Prefences 🡪 SQL Editor 🡪 SQL Eecusion 🡪 limitteki isareti kaldir 🡪 OK |
| IF NULL | # if null, “null” degerin yerine gorunmesini istedigimiz bilgiyi girmek icin kullanilir.  # iki parametrden fazlasini alamaz)  IFNULL(expresion1, expresion2)  SELECT dept\_no IFNULL(dept\_name, ‘not provided’) AS dept\_name FROM departments\_dup; |
| COALESCE | # Ifnull’in ikiden fazla parametre iceren versiyonu olarak dusunulebilir.  COALESCE(expresion1, expresion2, …, expresion-n)  #burada “expresion1’de “null” gordugunde “expresion2”deki bilgiyi, onda da yoksa ondan sonraki ifade de yer alan bilgiyi ceker) |
|  | SQL Joins |
|  | - we must find a related column from the two tables that contains the same type of data  - we will be free to add columns from these two tables to our output  - the columns you use to relate tables must represent the same object, such as id  - the tables you are considering need not be logically adjacent |
|  | **SELECT** t1.column\_name, t1.column\_name, …, t2.column\_name, …  **FROM** table\_1 **t1**  **JOIN** table\_2 **t2** **ON** t1.column\_name = t2.column\_name;  SELECT  m.dept\_no, m.emp\_no, d.dept\_name  FROM  dept\_manager\_dup m  INNER JOIN  departments\_dup d ON m.dept\_no = d.dept\_no  ORDER BY m.dept\_no;  # duplication’i engellemek icin “order by”dan once “GROUP BY m.emp\_no” |
| INNER JOIN = JOIN | Ortak sutun bilgisi ayni olan kayitlara ait istedigimiz bilgileri getirir. Kesisimi getirir. |
| LEFT JOIN | # table’larin sirasini degistirdigimizde, iki tablodaki ortak bilgilerin alinacagi table’in da degistirilmesi gerekir. Mesela her iki tablo da dept\_no var ise, dept\_no bilgisinin alinacagi “table” solda kalan “table” olmalidir.  # LEFT JOIN = LEFT OUTER JOIN |
| RIGHT JOIN | # “table”larin sirasini degistirdigimizde ayni islemi “left join” ile de yapabilecegimizden, “right join” genelde kullanilmaz. |
| CROSS JOIN | Bu iki tabloyunun tum verilerini biraraya getiriyor. Maksat nedir henuz anlamadim. Mesea dept\_1 de calisan birinin bilgisi 10 ayri dept bilgisi ile niye birlestirilir. Adamin calismadigi departmana ait bilgilerle gorme ihtiyaci niyedir.  # “ON” ile birlikte kullanilmayan, yani ortak bir column uzerinden birlestirilmediginde “(INNER) JOIN” statement da ayni sonucu veriyor. Ilk tablodaki her bir kayit icin ikinci tablo kayitlari tekrar ediliyor. Yani birinci tablodaki kayit sayisi, ikinci tablodaki kayit sayisi buyuklugunde bir katsayi ile buyutulmus oluyor.  # Kesin olarak anladigim, “ON” ile, yani bir condition ile birlestirme yapilmayacaksa “JOIN” degil “CROSS JOIN” kullanilmalidir.  (JOIN + ON) = (CROSS JOIN + WHERE)  SELECT  e.\*, d.\*  FROM  departments d  CROSS JOIN  dept\_manager dm  JOIN  employees e ON dm.emp\_no = e.emp\_no  WHERE  d.dept\_no <> dm.dept\_no  ORDER BY dm.emp\_no , d.dept\_no;  # Burada ne oldu:  Employees tablosu ile departmentsi biraraya getirmek istiyorum ama ortak noktalari yok. Dolaysi ile once eployees’I dept\_manager ile “conditional” birlestirme yapiyorum.  Birlesmis olan bu tablonun artik deparments ile ortak bir column’i var, “dept\_no”  Bu iki tabloyuda “cross join” ile birlestiriyorum ama tum bilgileri kullanmak zorunda degilim, hatta bu iki tabloyu biraraya getiren tablonun hicbir bilgisini almayabilirim ki burada oyle yapilmis.  “Where” ile manager’in kendi calistigi department haric tutulmus ama maksadi anlamadim tabi. |
|  | Pippin: The greater data we have, the farther we are from harm |
|  | as |
|  | |  |  | | --- | --- | | 66:10:b0:74:6c:87:af:12:bb:b2:a2:0b:29:6e:a7:f1:7c:8f:ff:fa |  | |
|  |  |