## sql زبانی سطح بالا برای ایجاد، تغییر، و بازیابی داده‌ها و نیز عملیات بر روی آن‌ها در دیتابیس های رابطه ای میباشد

================================================================================

**SELECT**

از دستور SELECT برای انتخاب اطلاعات از یک Table استفاده میشود

پیشنهاد میشود دستورات sql را با حروف بزرگ و اسم جدول ها و ستون ها را با حروف کوچک بنویسید

در آخر همه ی کدهای sql لازم است که حتما ; قرار دهید

SELECT \* FROM employees ====> انتخاب تمام ستون ها از جدول employees

SELECT firstname, lastname FROM employees ====> انتخاب ستون اسم و فامیل از جدول employees ( توجه شود که ابتدا ستون firstname و سپس ستون lastname را نمایش میدهد)

SELECT DISTINCT country FROM customer ====> موارد تکراری را که در ستون country از جدول customer میباشد را حذف میکند و مقادیر متمایز را نمایش میدهد

SELECT COUNT(\*) FROM customer ====> تعداد رکورد های (سطرها) موجود در جدول customer را نمایش میدهد

SELECT COUNT(DISTINCT country) FROM customer ====> تعداد موارد متمایز از ستون country در جدول customer را نمایش میدهد

SELECT customerid, shippeddate – orderdate FROM orders ====> دو ستون را نشان میدهد که یکی از ستون ها customerid میباشد و ستون دیگر حاصل تفریق دو ستون shippeddate و orderdate میباشد ( میخواهیم ببینم بعد از چند روز سفارش به دست مشتری رسیده است )

================================================================================

**WHERE**

از دستور WHERE زمانی استفاده میشود که بخواهیم با ایجاد کردن شرط اطلاعات انتخاب شده را فیلتر کنیم

SELECT \* FROM customer WHERE country=’Germany =====> سطرهایی را برمیگرداند که مقدار country برابر Germany باشد ( به حروف بزرگ و کوچک حساس میباشد)

SELECT \* FROM order\_details WHERE quantity >=10 ====> سطرهایی را برمیگرداند که مقدار quantity بزرگتر یا مساوی 10 باشد

SELECT \* FROM order WHERE orderdate > ‘1996-7-8 ===> سطرهایی را برمیگرداند که مقدار orderdate بزرگتر از تاریخ 8-7-1996 باشد ( توجه شود که باید داخل ‘ ’ قرار گیرد )

دو کلمه کلید AND و OR در دستور WHERE وجود دارد که میتوانیم شرط هایمان را بزرگتر کنیم مثال:  
SELECT \* FROM customer WHERE country=’Germany ‘ AND city=’berlin ====> سطرهایی را برمگرداند که کشورشان Germany و شهرشان berlin باشد

SELECT \* FROM customer WHERE country=’Germany ‘ OR city=’paris ===> سطرهایی را برمگرداند که کشورشان Germany یا شهرشان paris باشد

SELECT \* FROM customer WHERE NOT country=’Germany ====> دستور NOT شرطی که قرار داده اید را معکوس میکند در این مثال رکوردهایی را برمیگرداند که کشورشان Germany نباشد

ما میتوانیم از BETWEEN در مواقعی که میخواهیم بین دو عدد را انتخاب شود استفاده کنیم :  
SELECT \* FROM order\_details WHERE unitprice >= 10 AND unitprice <= 43   
 =  
SELECT \* FROM order\_details WHERE unitprice BETWEEN 10 AND 43 ====> در این مثال تمامی رکورد هایی که در جدول order\_detail وجود دارد و قیمت هر واحدشان بین 10 و 43 میباشد را نمایش میدهد

گاهی اوقات میخواهیم چندین شرط OR را بنویسیم به جای اینکار میتوانیم از دستور IN استفاده کنیم مثال:  
SELECT \* FROM customer WHERE country=’Germany ‘ OR country=’Mexico’ OR country=’Spain  
 =  
SELECT \* FROM customer WHERE country IN(‘Germany’, ‘Mexico’, ‘Spain’) ===> در این مثال در سطرهایی را برمیگرداند که یا کشورشان Germany یا Spain یا Mexico باشد

================================================================================

**ORDER BY**

دستور ORDER BY برای مرتب کردن اطلاعات انتخاب شده استفاده میشود این مورد به ما در به دست آوردن نتایج بهتر کمک میکند

SELECT DISTINCT country FROM customers ORDER BY country ASC  
یا   
SELECT DISTINCT country FROM customers ORDER BY country DESC ===> بر اساس حروف الفبا از بالا به پایین یا از پایین به بالا مرتب میکند ( توجه شود که میتوانید بر روی دو ستون با حالت های مختلف ORDER BY انجام داد ) ( توجه شود که ORDER BY بر روی اعداد نیز اعمال میشود) ( حروف الفبا فقط انگلیسی میباشد و بر روی فارسی کاربرد ندارد )  
SELECT DISTINCT country , city FROM customers ORDER BY country ASC , city DESC

دستورات MAX و MIN و AVG و SUM وجود دارد که به ترتیب به معنی بزرگترین و کوچکترین و میانگین و حاصل جمع میباشد که میتوانیم بر روی یک ستون اعمال شود مثال

SELECT MAX(unitprice) FROM order\_details ===> بیشترین مقدار در ستون unitprice از جدول order\_details نمایش میدهد

================================================================================

**LIKE**

دستور LIKE در کنار دستور WHERE به کار رفته و برای جستجوی اطلاعات براساس الگویی خاص به کار میرود به زبان دیگر ما میتوانیم در sql پترنی را مشخص کنیم و بگوییم نتایج را براساس آن پترن برای ما نمایش بده

SELECT \* FROM customers WHERE companyname LIKE ‘A%p ===> در این مثال الگوی ما به این صورت میباشد که با حرف A شروع شود و صفر یا بیشتر کاراکتر وجود داشته باشد و در آخر با حرف p تمام شود این الگو را بر روی ستون companyname اعمال کن ( Ap و Adjfdjp وAbp و ….)

% ====> به معنی صفر یا به هر تعداد کاراکتر

ـ ( زیر خط ) ====> به معنی یک کاراکتر میباشد

مثال :

SELECT \* FROM customers WHERE companyname LIKE ‘A\_p% ===> Aipldfkdslfk, Aup, Arpafdddd

================================================================================

**AS LIMIT**

در زبان sql از دستور AS برای تغییر نام ستون اطلاعات و دستور LIMIT برای محدودسازی تعداد نتایج بازگردانده شده استفاده میشود

SELECT unitprice \* quantity FROM order\_details ===>هنگامیکه این دستور را اجرا میکنیم از جدول order\_details دو ستون unitprice و quantity را در هم ضرب میکند و در یک ستون نمایش میدهد اما این ستون اسم مشخصی ندارد و بی مفهوم میباشد برای رفع این مورد میتوانیم از دستور زیر استفاده کنیم که اسم ستون را برابر total قرار دهیم   
SELECT unitprice \* quantity AS total FROM order\_details

حتی میتوانیم از ORDER BY نیز استفاده کنیم :

SELECT unitprice \* quantity AS total  
FROM order\_details ORDER BY total DESC

توجه شود که بهتر است بر روی total عملیات ORDER BY انجام شود

SELECT unitprice \* quantity FROM order\_details LIMIT 30 ===> در این مثال با استفاده از دستور LIMIT به تعداد ۳۰ اول نمایش داده میشود

SELECT companyname , region FROM customers WHERE region IS null ===> در این مثال شرط null بودن را بر روی ستون region اعمال کردیم ( بر عکس این مورد IS NOT میباشد )

================================================================================

**INNER JOIN**

از دستور INNER JOIN برای انتخاب اطلاعاتی که در بین دو Table مشترک هستند استفاده میشود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SELECT companyname, orderdate, shipcountry

FROM customers JOIN orders

ON orders.cusomderid = customers.customerid

ستون companyname در جدول customers و دو ستون orderdate و shipcountry در جدول orderds قرار دارند   
مفهوم این query به این صورت است که orderdate و shipcountry که مربوط به سفارش میباشد را در کنار companyname همان مشتری قرار میدهد ( به دلیل این که customerid هم در جدول customers و هم در جدول orders وجود دارد میتوانیم بر اساس آن فیلتر کنیم "ON orders.cusomderid = customers.customerid” )

مفهوم ON به این صورت است که بر اساس چه فیلدی به هم متصل شود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SELECT companyname, orderdate, shipcountry, firstname

FROM customers

JOIN orders ON orders.cusomderid = customers.customerid

JOIN employees orders.eployeeid = employees.employeeid

در این مثال ۳ جدول customers و orders و employees را بر اساس فیلد مشترکی که در جدول orders وجود داشت به

هم متصل کردیم

دقیقا JOIN که در این دو مثال استفاده شد برابر با INNER JOIN میباشد و یک مفهوم را میرساند

================================================================================

# **LEFT, RIGHT, FULL JOIN**

SELECT companyname, orderid

FROM customers LEFT JOIN orders ON orders.cusomderid = customers.customerid

مفهوم LEFT JOIN به این صورت است که در جدول سمت چپ (‌در این مثال customers ) اگر مقداری برایش در جدول orders وجود نداشت یعنی مشتری وجود داشت که سفارشی نداشته باشد بازهم همچین مشتری را در خروجی برگردان

RIGHT JOIN هم دقیقا ماننده LEFT JOIN میباشد با تفاوت این که جدول سمت راست را در نظر میگیرد

FULL JOIN از ترکیب LEFT JOIN و RIGHT JOIN به دست می آید

================================================================================

**SELF JOIN**

SELECT e1.birthdate, e2.birthdate

FROM employees e1, employees e2

WHERE e1.birthdate > e2.birthdate;

مفهوم query بالا به این صورت است که دوبار جدول employees را باهم ترکیب کرده است و خروجیش به این صورت است که از جدول اول ‌( e1 ) هر رکورد را به ترتیب بر میدارد و با تمام رکورد های جدول دوم ( e2 ) مقایسه میکند و آن رکورد هایی که در e2 کوچکتر از رکورد انتحابی در e1 بود را برمیگرداند

مثالا 8-12-1948 تنها از یک تاریخ بزرگتر میباشد و 19-2-1952 از ۲ تاریخ بزرگتر میباشد

================================================================================

**GROUP BY**

از دستور GROUP BY زمانی استفاده میشود که بخواهید نتایج شما بر اساس چند ستون بازگردانده شوند به عبارت دیگر مثلا در جدول مشتریان میخواهیم ببینیم از هر کشور چندتا مشتری داریم برای این کار باید از دستور GROUP BY استفاده شود (توجه شود که هنگامیکه از این دستور استفاده میشود حتما باید از یکی از AVG , SUM و یا COUNT استفاده شود)

SELECT country, COUNT(\*)

FROM customers

GROUP BY country;

الان در این مثال خروجی به این صورت است که از هر کشور به هر تعداد تکراری که در جدول مشتریان داشته است را نمایش میدهد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SELECT companyname, COUNT(freight)

FROM customers

JOIN orders ON customers.customerid = orders.customerid

GROUP BY companyname

در این مثال تعداد مالیات هایی که هر شرکت پرداخت کرده است را نمایش میدهد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

میتوانیم به جای COUNT از AVG یا SUM نیز استفاده کینم که اگر از این دو مورد استفاده کنیم اعداد نمایش داده شده اعشاری میشود برای رفع این مشکل میتوانیم از دستور ROUND استفاده کنیم

SELECT companyname , ROUND(AVG(freight))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SELECT productname, SUM(quantity \* order\_details.unitprice) AS total

FROM order\_details

JOIN products ON products.productid = order\_details.productid

JOIN orders ON orders.orderid = order\_details.orderid

WHERE orderdate BETWEEN ‘1997-01-01’ AND ‘1997-12-31’

GROUP BY productname

در این مثال میزان فروش هر محصول در سال 1997 را نشان میدهد به این صورت که میزان فروش هر محصول برابر است با مجموع قیمت هر محصول ضرب در تعداد در تمام سفارشات میباشد

================================================================================

**HAVING**

دستور HAVING برای انتخاب کردن اطلاعات با استفاده از متدهای aggregate استفاده میشود به عبارت دیگر هماننده WHERE باعث ایجاد شرط میشود با این تفاوت که همیشه و همیشه WHERE باید قبل از GROUP BY بیاید و HAVING باید بعد از GROUP BY بیاید ( توجه شود که نمیتوان جلوی HAVING از اسم مستعار که به وسیله AS ساخته شده است استفاه کرد )

SELECT productname, SUM(quantity \* order\_details.unitprice) AS total

FROM order\_details

JOIN products ON products.productid = order\_details.productid

JOIN orders ON orders.orderid = order\_details.orderid

WHERE orderdate BETWEEN ‘1997-01-01’ AND ‘1997-12-31’

GROUP BY productname

HAVING SUM(quantity \* order\_details.unitprice) > 2000

اسم هر محصول همراه با مجموع فروش در سال 1997 به شرطی که مجموع فروش بیشتر از 2000 باشد

================================================================================

**UNION**

دستور UNION برای ترکیب کردن نتایج دو یا چند دستور SELECT استفاده میشود

SELECT companyname FROM customers

UNION

SELECT companyname FROM suppliers

در این مثال موارد غیر تکراری که در دو جدول suppliers و customers بر اساس companyname وجود دارد را برمیگرداند ( اگر بخواهیم خروجی تمام موارد همراه با موارد تکراری باشد میتوانیم از دستور UNION ALL استفاده کنیم )

توجه شود که هنگامیکه از دستور UNION استفاده میشود حتما باید تعداد ستون های انتخابی در هر دو دستور SELECT و نوع آنها یکسان باشد

================================================================================

**INSERT INTO**

دستور INSERT INTO برای وارد کردن اطلاعات جدید به Tableها استفاده میشود

INSERT INTO table\_name

(column1, column2, column3, …)

VALUES (value1, value2, value3, …)

================================================================================

**UPDATE**

دستور UPDATE برای بروزرسانی اطلاعات یک فیلد به کار میرود

UPDATE table\_name

SET column1 = value1 , column2 = value2, …

WHERE conditions

================================================================================

**DELETE**

دستور DELETE برای حذف یک یا چند فیلد از داخل یک Table استفاده میشود

DELETE FROM table\_name

WHERE condition;

================================================================================

**SELECT INTO**

از دستور SELECT INTO برای کپی کردن اطلاعات از یک Table به یک Table جدید به کار میرود

SELECT column1, column2, …

INTO new\_table

FROM old\_table

WHERE condition;

================================================================================

INSERT INTO SELECT

دستور INSERT INTO SELECT برای کپی کردن اطلاعات و وارد کردن آنها به یک Table جدید به کار میروند. نوع داده های ستون ها در مبدا و مقصد باید به یک شکل باشند ( تفاوتی که با SELECT INTO دارد این است که این دستور table جدیدی ایجاد نمیکند )

INSERT INTO table2 (column1, column2, …)

SELECT column1, column2, …

FROM table1

WHERE condition;

or if all fields match

INSERT INTO table2

SELECT \* FROM table1

WHERE condition;

در این دو الگوی بالا table2 جدول هدف میباشد و table1 جدول مبدا میباشد به این معنی که اطلاعات از table1 وارد table2 میشود

================================================================================

**DROP DATABASE $ CREATE DATABASE**

از دستورات CREATE DATABASE و DROP DATABASE برای ایجاد کردن و حذف کردن دیتابیس ها استفاده میشود

CREATE DATABASE name\_database

DROP DATABASE name\_database

================================================================================

**CREATE TABLE**

از دستور CREATE TABLE برای ایجاد کردن یک Table استفاده میشود

CREATE TABLE table\_name (  
column1 datatype,

column2 datatype

….

)

column1 و column2 در واقع اسم ستون ها میباشد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

در pgadmin برای این که بفهمیم هر table با چه دستوری ساخته شده است میتوانیم بر روی table راست کلیک کرده بر روی گزینه ی script رفته و بر روی CREATE script کلیک کرده

================================================================================

**ALTER TABLE**

از دستور ALTER TABLE برای ایجاد تغییر در Tableها به کار میرود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

تغییر اسم یک ستون

ALTER TABLE table\_name

RENAME column\_oldname TO column\_newname

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

تغییر اسم جدول

ALTER TABLE old\_table\_name

RENAME TO new\_table\_name

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

اضافه کردن یک ستون جدید

ALTER TABLE table\_name

ADD column\_name datatype

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

حذف یک ستون

ALTER TABLE table\_name

DROP COLUMN column\_name

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

تغییر data type یک ستون

ALTER TABLE table\_name

ALTER COLUMN column\_name SET DATA TYPE datatype

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# INDEX

دستور CREATE INDEX برای ایجاد کردن یک ایندکس استفاده میشود و دستور DROP INDEX باعث حذف ایندکس میشود

ایندکس ها به شما کمک میکنند که زمانیکه داخل data base سرچ انجام میدهید دیگه نیاید تک تک فیلد ها رو بگردید تا نتیجه رو برگردونه در این حالت از ایندکس ها کمک میگیرید تا نتیجه رو سریع تر برگرداند

index ها را هماننده فهرست اول کتاب در نظر بگیرید که نشان میدهد هر موضوع در چه صفحه ای میباشد

indexی که از ستون مورد نظر ساخته میشود مرتب شده میباشد به همین دلیل سرعت سرچ بسیار بالا میرود

هنگامیکه index ساخته میشود ستونی اضافه میشود به نام pointer که آدرس حافظه ای که بقیه اطلاعات در آن قرار دارد را نمایش میدهد

CREATE INDEX index\_name

ON table\_name ( column1, column2, …)

or

CREATE UNIQUE INDEX index\_name

ON table\_name (column1, column2, …)

index\_name بهتر است از ترکیب اسم ستون مورد نظر و اسم table ساخته شود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DROP INDEX index\_name

این دستون باعث حذف ایندکس میشود

================================================================================

# **DROP TABLE**

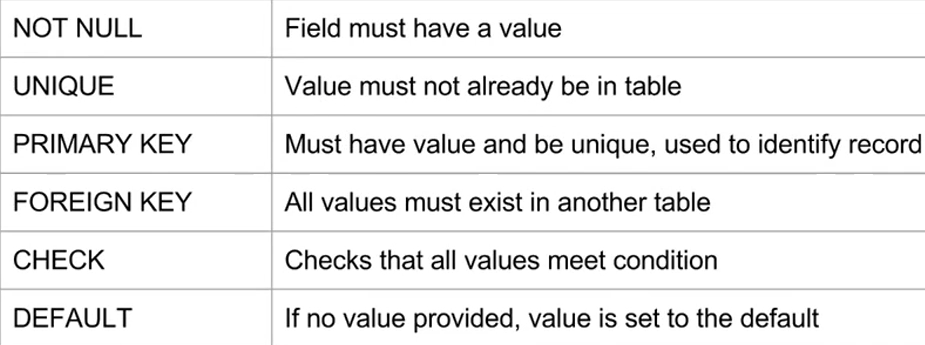
دستور DROP TABLE برای حذف کردن یک Table به کار میرود

DROP TABLE table\_name

================================================================================

**Constraint**

constraint در لغت به معنی محدودیت میباشد به این صورت که با استفاده از یک سری دستورات میتوانیم محدودیت هایی رو بر روی ستون های table اعمال کنیم   
ساده ترین نوع محدودیت data type میباشد که ستون را مجبور میکنیم فقط از آن نوع data type قبول کند



================================================================================

**NOT NULL**

با استفاده از دستور NOT NULL میتونیم یک ستون رو مجبور کنیم که مقدار null رو قبول نکنه

CREATE TABLE table\_name(

column\_name datatype NOT NULL

)

================================================================================

**UNIQUE**

با استفاده از دستور UNIQUE میتونید مطمئن بشید که ستون هاتون مقدار تکراری قبول نمیکنن

CREATE TABLE table\_name(

column\_name datatype UNIQUE

)

================================================================================

**PRIMARY KEY**

دستور PRIMARY KEY برای ایجاد کردن یک کلید اصلی به کار میروند. کلیدهای اصل برای شناسایی رکوردها از یکدیگر به کار میروند. کلید های اصلی هیچگاه نباید null و تکراری باشند

CREATE TABLE table\_name(

column\_name datatype PRIMARY KEY

)

Equivalent to : UNIQUE NOT NULL

================================================================================

**FOREIGN KEY**

دستور FOREIGN KEY برای ایجا کلیدهای خارجی استفاده میشود. کلیدهای خارجی برای برقراری ارتباط بین دو Table به کار میروند

CREATE TABLE table\_name(

column\_name1 datatype,

column\_name2 datatype,

…..

FOREIGN KEY (column\_name) REFERENCES table2 (column\_name)

)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE groups(

groupid inteeger PRIMARY KEY,

customerid varchar(10),

name text UNIQUE,

FOREIGN KEY (customerid) REFERENCES customers (customerid)

)

در این مثال table جدیدی ساخته شد که در آن فیلدی به نام customerid وجود دارد و این فیلد را به عنوان کلید خارجی به جدول customers به فیلد customerid متصل کردیم

================================================================================

**DEFAULT**

دستور DEFAULT در sql برای ایجاد مقدار پیشفرض استفاده میشود. مقدار پیشفرض به جای مقدار null قرار میگیرد

CREATE TABLE table\_name(

column\_name1 datatype,

column\_name2 datatype DEFAULT value/function

)

================================================================================

**CHECK**

دستور CHECK برای ایجاد کردن یک شرط در زمان ایجاد Table به کار میرود

CREATE TABLE table\_name(

column\_name1 datatype

column\_name2 datatype CONSTRAINT name CHECK (condition)

)

CREATE TABLE test(

name text

age inteeger CONSTRAINT ch\_age CHECK (age>10)

)

OR

CREATE TABLE table\_name(

column\_name1 datatype

column\_name2 datatype,

CONSTRAINT name CHECK (conditions)

)

CREATE TABLE test(

name text,

age inteeger,

CONSTRAINT ch\_age CHECK (age>10 AND LENGTH(name)>4)

)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

برای حذف این شرط ها میتوانیم از دستور زیر استفاده کنیم

ALTER TABLE table\_name

DROP CONSTRAIN name

ALTER TABLE test

DROP CONSTRAIN ch\_age