درج رکورد جدید و بازگرداندن رکوردهای درج شده:

INSERT INTO cn\_nodes (node\_id, name)

VALUES (‘something’, ‘something’)

RETURNING \*;

به روز رسانی رکورد و بازگرداندن رکوردهای به روز شده:

UPDATE cn\_nodes

SET name = ‘something’

RETURNING \*;

نحوه اجرای UPSERT:

INSERT INTO cn\_nodes (node\_id, name)

VALUES (‘something’, ‘something’)

ON CONFLICT (node\_id)

DO NOTHING; //اگر آی دی مد نظر هم اکنون در جدول وجود دارد، آن را نادیده بگیر

INSERT INTO cn\_nodes (node\_id, name)

VALUES (‘something’, ‘something’)

ON CONFLICT (node\_id)

DO

UPDATE

SET name = EXCLUDED.name;

نحوه برخورد با مقادیر NULL در SORT BY:

SELECT \*

FROM cn\_nodes

ORDER BY name DESC NULLS FIRST //or NULLS LAST

گرفتن تعداد مشخصی رکورد و همزمان، نادیده گرفتن 5 رکورد اول:

SELECT \*

FROM cn\_nodes

ORDER BY name

LIMIT 10 OFFSET 5 //the OFFSET section is optional

LIMIT 10 OFFSET 5

Is exactly equal to (the second is SQL standard):

OFFSET 5

FETCH NEXT 10 ROWS ONLY

در جستجوی متن، ILIKE دقیقا مانند LIKE است؛ با این تفاوت که LIKE یک عملگر case-sensitive است، اما ILIKE نیست. در جستجو، کاراکتر % به معنای هر تعداد کاراکتر دلخواه و کاراکتر \_ (underline) به معنای فقط یک کاراکتر است.

برای چسباندن مقادیر متنی به یکدیگر سه راه وجود دارد:

Inputs: “gesi”, “chaghochi”, “khers”

Output: “gesi, chaghochi, khers”

1: ‘gesi’ || ‘, ‘ || ‘chaghochi’ || ‘, ‘ || ‘khers’

2: CONCAT(‘gesi’, ‘, ‘, ‘chachochi’, ‘, ‘, ‘khers’)

3: CONCAT\_WS(‘, ‘, ‘gesi’, ‘chachochi’, ‘khers’)

در صورتی که یکی از مقادیر سمت چپ یا راست عملگر || برابر با NULL باشد، خروجی هم NULL خواهد بود. عملگر || مقادیر را تبدیل نمی کند؛ یعنی اگر یکی از دو طرف آن عدد باشد، آن را به رشته تبدیل نمی کند.

توابع CONCAT و CONCAT\_WS مقادیر را به رشته تبدیل می کنند؛ یعنی می توانید به آنها ورودی عددی هم بدهید. به علاوه، این توابع مقادیر NULL را نادیده می گیرند و به جای آن رشته خالی یعنی ‘’ را قرار می دهند.

همه DataType ها می توانند Array هم باشند.

برای INSERT کردن یک مقدار به صورت آرایه باید به روش زیر عمل کنید:

INSERT INTO x (arr) VALUES (ARRAY[‘first value’, ‘second value’])

برای SELECT کردن ایندکس خاصی از یک آرایه باید به روش زیر عمل کنید (مقدار از 1 شروع می شود):

SELECT arr[1] FROM x //gets the first element of array

برای تبدیل DataType می توانید از یکی از دو روش زیر استفاده کنید که دقیقا معادل هستند:

1. SELECT CAST(‘1212:45454’ AS FLOAT)

2. SELECT ‘1212:45454’::FLOAT

برای محاسبه فاکترویل به صورت زیر عمل کنید:

SELECT 20!

برای مقایسه مقادیر رشته ای با استفاده از Regular Expression به صورت زیر عمل کنید:

SELECT \*

FROM x

WHERE title ~ E’[Expression]’

e.g.

SELECT \* FROM x WHERE title ~ E’^\\d\*$’

نحوه تعریف unique constraint در زمان تعریف جدول:

CREATE TABLE table\_name

(

id SERIAL PRIMARY KEY,

name varchar(100) NOT NULL,

code int NOT NULL,

UNIQUE (name, code)

);

نحوه اضافه کردن unique constraint به جدول موجود:

ALTER TABLE x

ADD CONSTRAINT constraint\_name UNIQUE (col1, col2, …)

نحوه تعریف check constraint در زمان تعریف جدول:

CREATE TABLE prices

(

id SERIAL PRIMARY KEY,

price FLOAT NOT NULL CHECK (price > 0)

);

نحوه تعریف check constraint برای جدول موجود:

ALTER TABLE prices

ADD CONSTRAINT price\_check

CHECK (price > 0 AND discount >= 0 AND price > discount);

کارهای محاسباتی با زمان را می توان به اشکال مختلفی انجام داد:

1. SELECT now() + INTERVAL ‘3 DAYS’ – INTERVAL ‘2 hours’

//MILLENNIUMS, CENTURIES, DECADES, YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, MILLISECONDS, MICROSECONDS

2. SELECT now() + ’10:02:22’::time //10 hours and 2 minutes and 22 seconds

3. SELECT now() + time ’10:02:22’ //same as above

توابع مفید مربوط به تاریخ:

SELECT

NOW(),

TRANSACTION\_TIMESTAMP(), //same as NOW()

STATEMENT\_TIMESTAMP(), //time of execution of this statement

CLOCK\_TIMESTAMP(), //shows current date and time (changes during statement execution)

TIMEOFDAY(),

CURRENT\_DATE,

CURRENT\_TIME,

CURRENT\_TIMESTAMP,

LOCALTIME,

LOCALTIMESTAMP

تبدیل تاریخ به ثانیه (از اول تاریخ تاکنون):

SELECT EXTRACT (EPOCH FROM NOW())

تابع DATE\_PART:

SELECT DATE\_PART(‘year’, NOW())

//millennium, century, decade, year, quarter, month, week, dow (day of week), doy (day of year), day, hour, minute, second

تابع DATE\_TRUNC:

این تابع برای تعیین دقت یک مقدار تاریخی است. مثلا اگر بخواهید فقط تا سطح ساعت اطلاعات تاریخی را ذخیره کنید، باید دستوری شبیه به کد زیر را اجرا کنید. با این کار، مقادیر کوچکتر از ساعت نظیر دقیقه، ثانیه و میلی ثانیه به صفر تغییر خواهند یافت:

SELECT DATE\_TRUNC(‘hour’, NOW())

فیلتر کردن نتایج بعد از GROUP BY (که البته توی SQL SERVER هم احتمالا همین طور باشه!):

SELECT movie\_lang,

SUM(movie\_length)

FROM movies

GROUP BY movie\_lang

HAVING SUM(movie\_length) > 200

ORDER BY SUM(movie\_length)

معادل ISNULL:

SELECT COALESCE(NULL, [new value])

e.g. SELECT COALESCE(NULL, TRUE)

کلمه کلیدی INTERSECT دقیقا شبیه به UNION و UNION ALL استفاده می شود و نتایجی را بر می گرداند که هم در دیتاست بالا و هم در دیتاست پایین مشترک هستند (اشتراک). UNION و UNION ALL هر دو اجتماع را بر می گردانند؛ تفاوت آنها در این است که UNION ALL موارد تکراری را هم بر می گرداند، اما UNION موارد تکراری را نادیده میگیرد.

کلمه کلیدی EXECPT هم دقیقا شبیه به INTERSECT استفاده می شود و نتایجی را بر می گرداند که در دیتاست بالایی وجود دارند، اما در دیتاست پایین وجود ندارند.

برای توابع بازگشتی مثل گرفتن لیست بچه ها، درست همانند SQL SERVER از WITH استفاده می شود؛ با این تفاوت که در صورتی که هدف از WITH یک عملیات بازگشتی باشد، باید بعد از کلمه WITH، کلمه RECURSIVE نیز نوشته شود. یعنی:

WITH RECURSIVE some\_name AS (some sql statement)

برای ایجاد یک جدول پارتیشن بندی شده به روش HASH که بر اساس کلید UUID پارتیشن بندی شده است و حاوی دو پارتیشن می باشد، به روش زیر عمل کنید:

CREATE TABLE IF NOT EXISTS tbl (

id UUID NOT NULL PRIMARY KEY,

value int

) PARTITION BY HASH (id);

CREATE TABLE IF NOT EXISTS tbl\_partition\_1 PARTITION OF tbl

FOR VALUES WITH (MODULUS 2, REMAINDER 0);

CREATE TABLE IF NOT EXISTS tbl\_partition\_2 PARTITION OF tbl

FOR VALUES WITH (MODULUS 2, REMAINDER 1);

برای جدا کردن یک پارتیشن از یک جدول:

ALTER TABLE tbl DETACH PARTITION tbl\_partition\_1

برای متصل کردن یک جدول به عنوان پارتیشن به جدول دیگر (در صورتی که از روش HASH استفاده شده باشد):

ALTER TABLE tbl ATACH PARTITION tbl\_partition\_1

FOR VALUES WITH (MODULUS 2, REMAINDER 0);

این احتمال وجود دارد که tbl\_partition\_1 حاوی رکوردهای اطلاعاتی باشد که با شرط پارتیشن بندی سازگار نباشد. در این صورت، باید ابتدا اطلاعات آن را به جدول دیگری منتقل کنید. از آنجا که کل این فرآیند نباید موجب اختلال در عملکرد دیتابیس شود، بهتر است که کل این عملیات در قالب یک تراکنش انجام شود. چیزی شبیه به کد زیر:

START TRANSACTION;

CREATE TABLE IF NOT EXISTS xx (

id UUID NOT NULL PRIMARY KEY,

value int

);

INSERT INTO xx SELECT \* FROM tbl\_partition\_1;

DELETE FROM tbl\_partition\_1;

ALTER TABLE tbl ATTACH PARTITION tbl\_partition\_1

FOR VALUES WITH (MODULUS 2, REMAINDER 0);

INSERT INTO tbl\_partition\_1 SELECT \* FROM xx;

DROP TABLE xx;

COMMIT TRANSACTION;

در انتهای همه Stored Procedure هایی که در آنها بیش از یک عملیات مربوط به جداول انجام می شود، بهتر است که عبارت COMMIT; بیاید؛ در غیر این صورت، ممکن است خاصیت اتمیک خود را از دست بدهد. COMMIT مشخص می کند که کل بدنه Stored Procedure یک Transaction است.

برای اجرای خط به خط کد در pgAdmin باید چیزی شبیه کد زیر بنویسید آن را خط به خط highlight و اجرا کنید:

BEGIN;

SELECT test(111);

FETCH ALL IN "<unnamed portal 27>";

COMMIT;

در ابزار query tool در pgAdmin می توانید از بلاک زیر استفاده کنید:

DO

$$

DECLARE

//variables

BEGIN

//some code

END;

$$ LANGUAGE plpgsql;

برای تبدیل جدولی از نوع کامپوزیت composite\_type به آرایه ای از همین نوع به صورت زیر عمل کنید:

DECLARE arr composite\_type[];

CREATE TEMP TABLE tbl OF composite\_type;

arr := ARRAY(

SELECT x

FROM tbl AS x

);