IN3020/4020 – Database Systems
Spring 2021, Week 2.2a

SQL QUERIES (SELECT and a bit more)

Egor V. Kostylev (with M. Naci Akkøk)

Based upon slides by E. Thorstensen from Spring 2019

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Datatypes

- Handbook for data types (Chapters 8 and 9): https://www.postgresql.org/docs/9.2/sql.htm
- Few hints:
  - Exact vs. approximate (inexact) numeric types
  - Timestamps vs. intervals (time-zones are complicated)
  - Enums are not SQL-standard, will mean update if requirement is changed
  - Binary blobs are small and nice, large files should be directly on the disk (storage) if they aren´t super important
- See also: https://wiki.postgresql.org/wiki/Don%27t_Do_This

Arrays (Lists)

(more here https://www.postgresql.org/docs/9.1/arrays.html)

- SQL supports arrays as data type; they are lists, actually

```
CREATE TABLE sal_emp (
   name text, pay_by_quarter integer[4],
   schedule text[][]
);
```

- There is quite a number of operations for arrays
- Arrays can also be used in queries:
  `ARRAY(X, Y, 3)` creates an array

Array Operations

(more here https://www.postgresql.org/docs/9.2/static/functions-array.html)

- Pick an element:
  `SELECT codon[2] FROM genomesequence` …
- Concatenation:
  `g1.codon || 'ACU', g1.codon || g2.codon`
- Number of elements:
  … `WHERE cardinality(codon) > 100` ..
- Compare exact content:
  `g1.codon = g2.codon, g1.codon <> g2.codon`
- Compare with every element in the array: `ANY, ALL`
  `WHERE codon[3] = ANY(array['GGU', 'UGG', 'UAA'])`…
- «Flatten out» an array:
  `SELECT Chromosomenr, unnest(codon) FROM genomesequence;`

Views

- Queries stored for use later
- Can be nested — a view can use other views
- Can be a spaghetti if not properly structured and documented (like any other function, procedure or API library, really)

Triggers

- A trigger is executed («triggered») when an event occurs in a table.
- Think of listeners and such:
  «when (or on) button pressed then execute...»

- Events are **`INSERT, UPDATE, DELETE`** (part of DML)
- Very flexible mechanism for doing a lot of good and, if not careful, a lot of bad

Trigger example (continued)

```
CREATE TABLE employees(
   id int4 serial primary key,
   first_name varchar(40)NOT NULL,
   last_name varchar(40)NOT NULL
);

CREATE TABLE employee_audits (
   id int4 serial primary key,
   employee_id int4 NOT NULL,
   last_name varchar(40)NOT NULL,
   changed_on timestamp(6)NOT NULL
)
```

Trigger example (continued)

```
CREATE OR REPLACE FUNCTION log_last_name_changes ()
   RETURNS trigger AS $llnc$
   BEGIN
   IF NEW.last_name <> OLD.last_name THEN
       INSERT INTO employee_audits(employee_id,
   last_name,changed_on)
       VALUES(OLD.id,OLD.last_name,now());
   END IF;
RETURN NEW;
END;

$llnc$ language plpgsql;

CREATE TRIGGER last_name_changes
BEFORE UPDATE ON employees
```

Triggers – Hints

- Handy for logging, for complicated constraints and various house-keeping needs
- Can be complicated with many triggers and complex logic
- Especially if a cascade (i.e., a trigger changes another table with its own triggers): “Trigger hell” is a concept (unfortunately)