ZTB 2017:

GEOGRAPHY - używa pomiaru geodezyjnego zamiast pomiaru kartezjańskiego, reprezentuje punkty w postaci WGS 84, z punktem odniesienia SRID 4326, funkcje obliczające dystans np. ST_DISTANCE zwracają pomiary w metrach, można użyć dodatkowej ST_TRANSFORM aby zmienić układ odniesienia i otrzymać dokładniejsze dane. (sferyczne koordynaty reprezentowane kątowo)

GEOMETRY - starszy typ, ale nie można uznać go za przestarzałe ponieważ posiada większy zestaw funkcji i wsparcie zewnętrznych narzędzi, jednakże daje mniej dokładne rezultaty dla większych odległości, ponieważ opiera się

Geometry example:

```
SELECT ST_Distance(
      ST_Transform('SRID=4326;POINT(19.57 50.03)'::geometry, 3857),
      ST_Transform('SRID=4326;POINT(21.02 52.12)'::geometry, 3857)
);
```

Geography example:

```
SELECT ST_Distance(
      'SRID=4326;POINT(19.57 50.03)'::geography,
      'SRID=4326;POINT(21.02 52.12)'::geography
);
```

Błędy w transakcjach:
Serializable:

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SELECT * FROM mytab;
INSERT INTO mytab VALUES(1, 30);
COMMIT;

BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SELECT * FROM mytab;
INSERT INTO mytab VALUES(1, 50);
```

COMMIT -> błąd -> ERROR: Could not serialize access due to read/write dependencies among transactions.

Repetable reads:

```
BEGIN TRANSACTION ISOLATION LEVEL REPETABLE READ;
SELECT * FROM x;
UPDATE x SET class=4 WHERE value=30;
COMMIT;

BEGIN TRANSACTION ISOLATION LEVEL REPETABLE READ;
SELECT * FROM x;
UPDATE x SET class=5 WHERE value=30;
```

ERROR: COULD NOT SERIALIZE ACCESS DUE TO CONCURRENT UPDATE

Read committed - zawieszone przy takiej sytuacji jak wyżej, ale nie występuje błąd bo po odwieszeniu robimy update nowego stanu.
Read uncommitted - można wywołać błąd?

3. (2 pkt) Do implementacji bazy danych użyto biblioteki SQLite. Do przechowywania użyto zamiennie pliku bądź pamięci. Czasy wyszukiwania danych w przypadku użycia pliku albo pamięci były porównywalne. Zajętość RAM przez aplikacje była znacznie wyższa w przypadku pamięci. Wytłumacz to zjawisko.

Większa zajętość ram wynika z tego, że cały dataset przechowywany jest w pamięci w przypadku In-Memory database. Natomiast porównywalne czasy zapytań wynikać mogą na przykład z tego że czas operacji odczytu (fizycznej) jest pomijalnie mały w porównaniu np. do czasu wyszukiwania wynikającego z filtrów klauzuli WHERE, lub skomplikowanego sortowania. Innym powodem może być cache'owanie porcji danych na której operujemy w pamięci dla przypadku gdzie cała baza znajduje się na dysku.

4. Table "employees" has B-tree index on lastName(varchar) so it can be used to sort results in ASC order. For each results of "WHERE" query tell if this index will be used or not:

Zakładam dwa indeksy ponieważ zauważyłem, że postgres nie używa indeksu bez pattern_ops dla M%, natomiast zakładając sam pattern_ops nie używa go do sortowania.
CREATE index new_index ON samochody(marka) USING BTree;
CREATE index new_index_2 ON samochody(marka varchar_pattern_ops);

a) **lastName ILIKE 'Kowal%'**
b) **lastName LIKE 'Kowal%'**
c) **lastName LIKE '%ski'**
d) **lastName >= 'K'**

a - nie, ponieważ insensitive like nie zadziała dla liter na początku
b - tak, zadziała drugi indeks
c - nie, patterny tylko w sytuacji <stała>%
d - zadziała pierwszy indeks