Java 8 Date Time API mit PostgreSQL

Swiss PGDay 2016

Philippe Marschall

- Netcetera
- Java Entwickler seit 2007
  - viel Backend (JDBC)
- Java 8 Date Time Unterstützung in pgjdbc

Agenda

- alte JDBC Date API
- neue Java 8 Date Time API
- JDBC
- Spring JdbcTemplate
- JPA
- (pgjdbc)

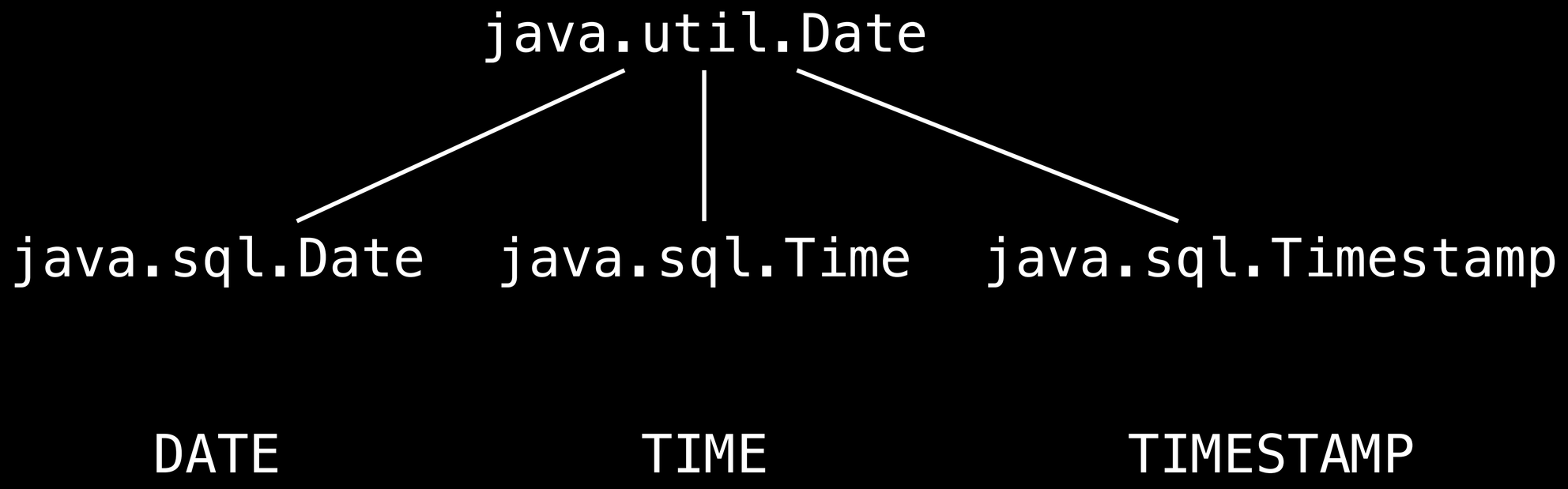
Publikum

- Architekten
- Java-Entwickler
- (DBAs)

alte Date API

- Warum brauchen wir eine neue API?

alte JDBC Date API



java.util.Date

- kein Datum, Zeitpunkt
- Anzahl Millisekunden seit 1.1.1970 00:00 GMT
- praktisch alle Methoden deprecated

java.sql.Date

- kein Subtyp
- Anzahl Millisekunden zwischen 1.1.1970 00:00 GMT bis 00:00 von Datum in JVM Zeitzone
- 24.6.2016
  - Anzahl Millisekunden zwischen 1.1.1970 00:00 GMT und 24.6.2016 00:00 CEST

Liskovsches Substitutionsprinzip

- Operationen die auf einem Supertyp funktionieren müssen auch auf einem Subtyp funktionieren
- https://de.wikipedia.org/wiki/Liskovsches_Substitutionsprinzip

JVM Zeitzone

- Betriebssystem-User Zeitzone
- kann beim JVM-Start überschrieben werden mit `-Duser.timezone`
- kann zur Laufzeit geändert werden mit `java.util.TimeZone.setDefault`

java.sql.Time

- kein Subtyp
- Anzahl Millisekunden zwischen 1.1.1970 00:00 GMT bis Zeit am 1.1.1970 in JVM Zeitzone
- 15:10
  - Anzahl Millisekunden zwischen 1.1.1970 00:00 GMT bis 1.1.1970 15:10 CEST

java.sql.Timestamp

- kein Subtyp
- `#equals` nicht symmetrisch
- Nanosekunden-Auflösung
- Anzahl Millisekunden seit 1.1.1970 00:00 GMT und Zeitpunk in JVM Zeitzone
  - plus Nanosekunden
- 24.6.2016 15:10
  - Anzahl Millisekunden seit 1.1.1970 00:00 GMT und 24.6.2016 15:10 CEST

Vor Java 8 nicht unterstützt

- (TIME WITH TIME ZONE)
- TIMESTAMP WITH TIME ZONE

TIME WITH TIME ZONE

- Postgres Dokumentation rät ab

TIMESTAMP WITH TIME ZONE

- einziger SQL Datentyp der eindeutig einen Zeitpunkt identifiziert

kein Zeitpunk

- 1. Mondlandung
- 1969-07-20 20:18:04

Zeitpunk

- 1969-07-20 20:18:04 Z
- 1969-07-20 21:18:04 Europe/Zurich
- 1969-07-20 23:18:04 Europe/Moscow
- 1969-07-20 16:18:04 America/New_York
- 1969-07-21 06:18:04 Australia/Sydney

Zeitpunkt

- alte JDBC API braucht Zeitpunkte für Sachen die keine Zeitpunkte sind
  - `DATE`
  - `TIME`
  - `TIMESTAMP`
- nur `TIMESTAMP WITH TIME ZONE` ist ein Zeitpunkt

TIMESTAMP WITH TIME ZONE

- “brauchen wir nicht”
- “wir haben nur eine Zeitzone”

Stille Datenabschneidung

- mit `TIMESTAMP [ WITHOUT TIME ZONE ]` können Daten beim speichern verändert werden wenn eine andere Zeitzone als UTC verwendet wird
- Sommer-Winterzeitumstellung
- Schweiz hat zwei Zeitzonen: CET und CEST

Stille Datenabschneidung

- 2016-10-30 03:00 → 2016-10-30 02:00
- 2016-10-30T02:55+02:00[Europe/Zurich]
- 2016-10-30T02:00+01:00[Europe/Zurich]
- 2016-10-30T02:05+01:00[Europe/Zurich]

TIMESTAMP WITH TIME ZONE

- PostgreSQL speichert als UTC
- originale Zeitzone nicht mehr verfügbar

neue Java Date Time API

- seit Java SE 8
- inspiriert von Joda-Time
- viele Klassen, “Framework”
- Domain-driven Design
  - keine spezifischen Typen für JDBC
- aka JSR-310

Java 8 API Date Time API

- relevant für ANSI SQL/JDBC
  - `LocalDate`
  - `LocalTime`
  - `LocalDateTime`
  - `OffsetDateTime`

LocalDate

- Datum ohne Zeit
  - Jahr
  - Monat
  - Tag
- keine Zeit, keine Zeitzone → kein Zeitpunkt

`LocalTime`

- Zeit ohne Datum
  - Stunde
  - Minute
  - Sekunde
  - Nanosekunde
- kein Datum, keine Zeitzone → kein Zeitpunkt

LocalDateTime

- Datum und Zeit
  - Datum: LocalDate
  - Zeit: LocalTime
- keine Zeitzone → kein Zeitpunkt

OffsetDateTime

- Zeitpunk
- LocalDateTime mit fixem Abstand zu GMT
  - Datum: LocalDate
  - Zeit: LocalTime
  - Abstand

JDBC 4.2

ANSI SQL	Java SE 8
`DATE`	`LocalDate`
`TIME`	`LocalTime`
`TIMESTAMP`	`LocalDateTime`
`TIME WITH TIMEZONE`	`OffsetTime`
`TIMESTAMP WITH TIMEZONE`	`OffsetDateTime`

Interval

- keine Entsprechung in Java 8 Date Time API

JDBC 4.2

- `PreparedStatement#setObject(int, Object)`
- `ResultSet#getObject(int, Class)`
- `ResultSet#getObject(String, Class)`
- keine Typenkonvertierung

#[sg]etObject

- in JDBC spezifizierte Datentypen
- Postgres-Erweiterungen:
  - `PGobject: PGbox, PGcircle, PGinterval, PGline, PGlseg, PGmoney, PGpath, PGpoint, PGpolygon`
  - `UUID`
  - `InetAddress`

`CallableStatement`

- Postgres Implementation von `#getObject` unterstützt nicht Java 8 Date Time API Typen

<<Beispiel>>

JdbcTemplate

- Spring ≥ 4.1
- Java 8 Datentypen für
  - Bind-Parameter
  - Rückgabe-Typen
- keine Typenkonvertierung

<<Beispiel>>

JPA

Java SE	Java EE	JDBC	JPA
7	7	4.1	2.1
8	(8)	4.2	(2.2)

JPA 2.2

- Java 8 Date Time API Unterstützung geplant (JPA_SPEC-63)
- Fahrplan unklar

`AttributeConverter`

- erlauben Konvertierung von Basistypen auf beliebige Typen
- AttributeConverter haben keinen Zugriff auf `ResultSet` oder `PreparedStatement`
  - `OffsetDateTime` nicht möglich

AttributeConverter

- Konvertierung über alte Datentypen
- selber schreiben
- diverse Open-source libraries
  - `com.github.marschall:threeten-jpa`
  - `org.jadira.usertype:usertype.extended`

Konvertierung über alte Datentypen

- kann aufgrund von Konvertierung über die lokale Zeitzone zu Problemen führen
- Zeit existiert in lokaler Zeitzone nicht

Hibernate

- 5.0.0: hibernate-java8 Modul
- 5.2.0: von Haus aus
- konvertiert über alte Datentypen (wie `AttributeConverter`)

Hibernate

- direkter Zugriff auf Java 8 Typen
  - `com.github.marschall:threeten-jpa-jdbc42-hibernate`
- bringt auch OffsetDateTime Unterstützung

<<Beispiel>>

JPA Fazit

- proprietäre Hibernate API für beste Resultate
  - `OffsetDateTime` nur mit Hibernate
- schlimmster Fall: gleiche Probleme wie bisher
  - JVM in UTC kann helfen

Treiber-Version

- ≥ 9.4.1208 / “JDBC42”
  - nicht “jre7” oder “jre6”
- kein Zusammenhang zur Datenbank-Version
- https://jdbc.postgresql.org/download.html
- org.postgresql:postgresql
- nicht von Linux-Distribution

Code

https://github.com/marschall/pgday2016