MySQL 8 XDevAPI: Neue Wege für Entwickler moderner Applikationen

Autoren: Mario Beck und Carsten Thalheimer, Oracle MySQL Deutschland

NoSQL hat viele neue Ideen in den Datenbankbereich gebracht, z.B. die Handhabung unstrukturierter Daten, simple APIs als Alternative zu SQL oder die Aufweichung strenger Anforderungen an Konsistenz zugunsten von Performance. MySQL 8 unterstützt viele dieser Konzepte und ermöglicht es so, das Beste aus beiden Welten in einem Datenbanksystem zu nutzen: z.B. MySQL als DocumentStore oder eine native CRUD API zum performanten Zugriff auf relationale Daten oder die Power von SQL nutzen, um nicht-strukturierte Daten dennoch effektiv in der Datenbank auswerten zu können.

Eigentlich beginnt die Geschichte bereits vor drei Jahren mit MySQL 5.7 und der Einführung des neuen Datentyps JSON, automatisch generierter, indizierbarer Spalten und einer Menge neuer SQL-Funktionen zur Manipulation von JSON-Daten. Seit MySQL 5.7 ist es möglich, Daten einer relationalen Tabelle im JSON-Format an die Applikation zu liefern:

```
mysql> SELECT json_object("family", name, "first", vorname)
FROM foo WHERE id=2;
+-----------------------------------------------+
| json_object("family", name, "first", vorname) |
+-----------------------------------------------+
| {"first": "Micky", "family": "Maus"}          |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

Unstrukturierte Daten können als JSON-Dokumente abgespeichert werden:

```
mysql> SELECT * FROM restaurants LIMIT 1\G
*************************** 1. row ***************************
doc: {"_id": "564b3259666906a86ea90a99", "name": "Dj Reynolds
Pub And Restaurant", "grades": [{"date": {"$date":
```

```
1409961600000}, "grade": "A", "score": 2}, {"date": {"$date":
1374451200000}, "grade": "A", "score": 11}, {"date": {"$date":
1343692800000}, "grade": "A", "score": 12}, {"date": {"$date":
1325116800000}, "grade": "A", "score": 12}], "address":
{"coord": [-73.98513559999999, 40.7676919], "street": "West
57 Street", "zipcode": "10019", "building": "351"}, "borough":
"Manhattan", "cuisine": "Irish", "restaurant_id": "30191841"}
_id: 564b3259666906a86ea90a99
1 row in set (0.00 sec)
```

Vor allem jedoch können seit MySQL 5.7 beide Welten verbunden werden und relationale Daten mit unstrukturierten Daten leicht kombiniert werden:

```
mysql> SHOW CREATE TABLE products\G
*************************** 1. row ***************************
       Table: products
Create Table: CREATE TABLE `products` (
  `id` int(11) NOT NULL,
  `name` varchar(20) DEFAULT NULL,
  `price` int(11) DEFAULT NULL,
  `attributes` json DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB …
1 row in set (0.00 sec)
```

Das neue CLI: MySQL Shell

MySQL 8 geht einen Schritt weiter und bietet ein neues Protokoll (X-Protokoll) und eine neue API zur Nutzung der Datenbank (X DevAPI). Das neue X-Protokoll wird von allen aktuellen Konnektoren in MySQL unterstützt (Java, php, C, C++, Python, JavaScript, .NET). Es kann nun wahlweise das alte (SQL-) Protokoll oder das neue X-Protokoll verwendet werden (siehe Abbildung1).

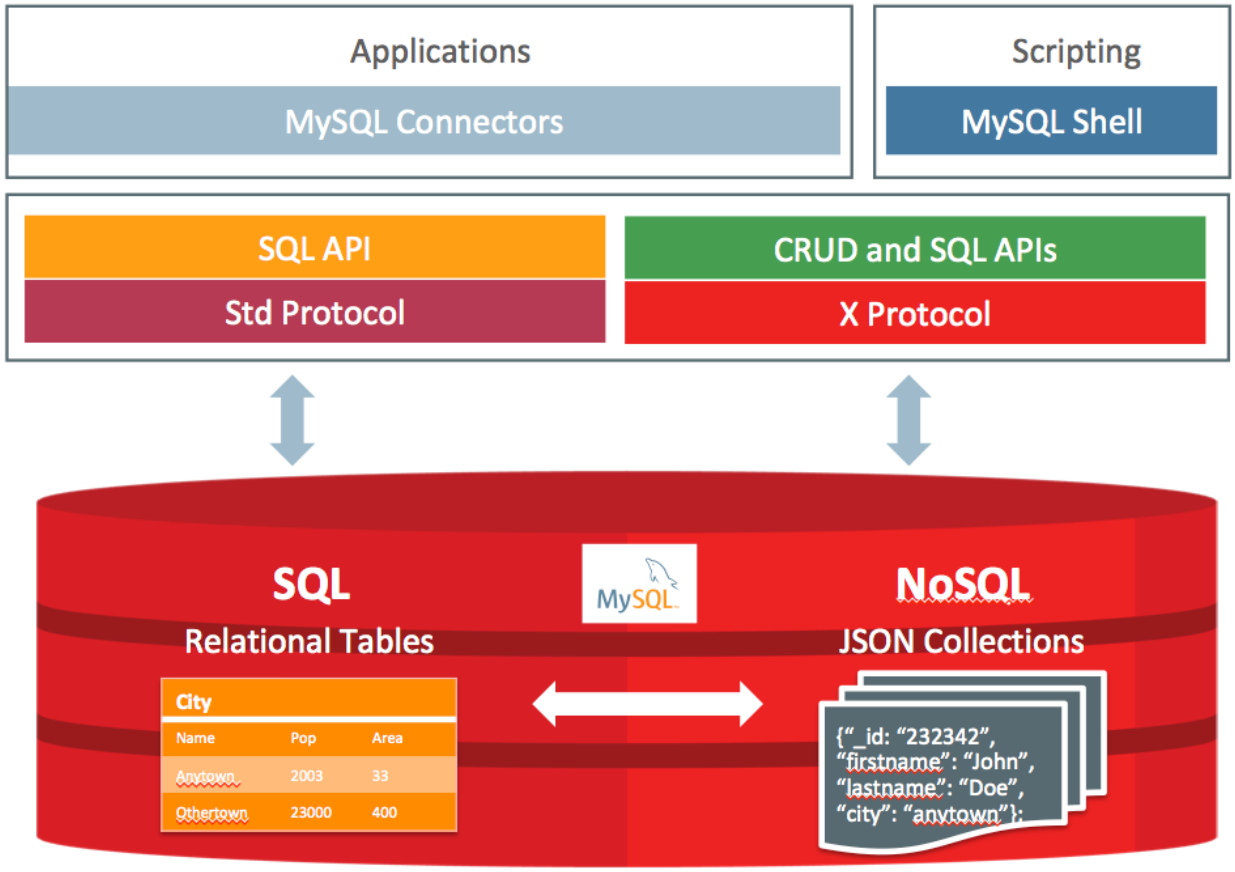

Abbildung 1: MySQL Architektur

Das neue CLI unterstützt ebenfalls beide Protokolle und ist somit eine Obermenge des alten „mysql“ CLI. Mit neueren MySQL 5.7 und MySQL 8 wird schwerpunktmäßig die neue MySQL Shell „mysqlsh“ anzutreffen sein. Bisher ist dieser Client allerdings noch ein separates zu installierendes Paket.
Beim Verbindungsaufbau wählt man bereits über den anzusprechenden Port das gewünschte Protokoll. Der Server kann über die Variable @@mysqlx_port beliebig konfiguriert werden (default 33060):

```
$ mysqlsh root@localhost:33060
Creating a session to 'root@localhost:33060'
Please provide the password for 'root@localhost:33060': ****
Save password for 'root@localhost:33060'? [Y]es/[N]o/Ne[v]er
(default No): y
…
Your MySQL connection id is 8 (X protocol)
…
```

```
Type '\help' or '\?' for help; '\quit' to exit.

 MySQL  localhost:33060+ ssl  JS > \sql
Switching to SQL mode... Commands end with ;

 MySQL  localhost:33060+ ssl  SQL > show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| json_restaurants   |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.0008 sec)
MySQL  localhost:33060+ ssl  SQL > \py
Switching to Python mode...

 MySQL  localhost:33060+ ssl  Py > \js
Switching to JavaScript mode...
```

Das neue Passwort-Management fällt beim Anmelden sofort auf: MySQL Shell kann verwendete Passwörter in einem sicheren Passwort-Speicher lagern und bei Bedarf wiederverwenden. Dabei kann zwischen dem üblichen Login-Path (plattformübergreifend) oder OS-abhängigen Keystores (OS-X Keychain, Windows Credential Manager) gewählt werden.
MySQL Shell hat drei Modi (SQL, JavaScript und Python), zwischen denen mit den neuen Backslash-Direktiven \js, \sql und \py gewechselt werden kann. Der SQL-Mode verhält sich nahezu identisch zum alten MySQL CLI. Der neue JavaScript Mode birgt mehr Neuerungen und wird uns durch die weiteren Beispiele begleiten. In Applikationen anderer Sprachen (php, Java, .NET, ...) sind die neuen API-Methoden jeweils vergleichbar aufgebaut. Dieser Artikel beschränkt sich auf die JavaScript-Notation. Alle Beispiele sind übrigens auf mysqlsh/MySQL 8.0.12 ausgeführt worden.

Nach dem Login ist Javascript der Default-Modus. Es sind einige globale Objekte automatisch definiert: `session` beschreibt die aktuelle Verbindung. `db` beschreibt die aktuell gewählte Datenbank. `dba` wird verwendet, um z.B. einen InnoDB Cluster zu administrieren. Ausgehend von diesen globalen Objekten können dann Benutzerobjekte erzeugt und manipuliert werden.

MySQL als DocumentStore

Als DocumentStore werden keine strukturierten Tabellen von Anwender erzeugt sondern lediglich Collections, die beliebig strukturierte JSON-Dokumente aufnehmen.

```
$ mysqlsh root@localhost:33060/demo
Creating a session to 'root@localhost:33060/demo'
…
Type '\help' or '\?' for help; '\quit' to exit.

 MySQL  localhost:33060+ ssl  demo  JS >
db.createCollection("posts")
<Collection:posts>

 MySQL  localhost:33060+ ssl  demo  JS >
db.posts.add({"title":"MySQL 8.0 rocks", "text":"My first
post!", "code": "42"})
Query OK, 1 item affected (0.0851 sec)

 MySQL  localhost:33060+ ssl  demo  JS >
db.posts.add({"title":"Polyglot database", "text":"Developing
both SQL and NoSQL applications"})
Query OK, 1 item affected (0.0169 sec)
```

Die Daten werden weiterhin in MySQL unter der InnoDB Storage Engine gespeichert. Dort erscheint die Collection als eine gewöhnliche, relationale Tabelle mit zwei Spalten: Einer JSON-Spalte, die das Dokument enthält und einer automatisch generierten Spalte mit dem _id Feld, welches jedem Dokument einen eindeutigen Identifier gibt. Bleiben wir aber zunächst auf der DocumentStore-Seite des Geschehens. Die üblichen CRUD-Operation (Create, Read, Update, Delete) haben in

der DocumentStore API ihre Entsprechung in den Methoden collection.add(), collection.find(), collection.modify() und collection.remove(). Einige Beispiele dazu sind

```
db.posts.find().limit(1)
db.posts.find("code='42'")

# modifying documents
db.posts.modify("code='42'").set("code", "43")
db.posts.modify("true").set("cool", "yes")
db.posts.modify("true").unset("cool")
db.posts.modify("title like 'MySQL%'").set("color",['red',
'blue', 'black'])
db.posts.find("'red' in color")
```

Für performante Zugriffe ist es natürlich auch wichtig, sekundäre Indizes zu erzeugen. Dazu gibt es die Methode collection.createIndex:

```
db.posts.createIndex('title_idx', {fields: [{field: "$.title",
type: "TEXT(20)"}]} )
```

Das Index-Feld wird als Array übergeben. Daran ist bereits abzulesen, dass auch Compound-Indizes, also Indizes, die aus mehreren Feldern aufgebaut sind, erzeugt werden können. Nun ist es wieder interessant, das Geschehen auf SQL-Seite zu verfolgen. Auch im JavaScript-Mode ist es möglich, traditionelle SQL-Kommandos zum Server zu schicken:

```
session.sql("DESC posts")
session.sql("SHOW CREATE TABLE posts")
```

Die der Collection zugrunde liegende Tabelle hat eine weitere Spalte erhalten. Deren Daten werden automatisch generiert aus der Expression `doc->$.title`, enthalten also den Wert des `title` Attributs der JSON-Dokumente (oder NULL, falls das Attribut nicht im Dokument existiert). Außerdem ist diese Spalte indiziert. Der Optimizer in MySQL wird diesen Index automatisch verwenden, wenn Queries mit dem Prädikat `doc->$.title="XYZ"` versehen sind. Entwickler müssen also diese Index-Spalte nicht berücksichtigen. Das übernimmt der Optimizer automatisch!

Transaktionen im DocumentStore

Da die Daten in der normalen InnoDB Engine gespeichert werden, stehen auch alle Eigenschaften des InnoDB im DocumentStore zur Verfügung. So auch Transaktionen: Es ist leicht möglich, Änderungen an Dokumenten oder ganzen Collections im Rahmen einer Transaktion zusammenzufassen und diese gemeinsam per COMMIT auszuführen oder durch ein ROLLBACK zurückzurollen. Der MySQL DocumentStore ist automatisch voll ACID-compliant! Das folgende Beispiel verschiebt ein JSON-Dokument von einer Collection in eine Zweite. Dank der Transaktionsklammer ist sichergestellt, dass auch collection-übergreifend entweder alle oder keines der Statements ausgeführt wird.

```
Db.createCollection(„archive“)
session.startTransaction()
t=db.posts.find(„code=‘43‘“).execute().fetchAll()
db.archive.add(t)
db.posts.remove(„code=‘43‘“)
db.posts.find()
db.archive.find()
session.rollback()
db.posts.find()
db.archive.find()
```

Natürlich kann ein Entwickler je nach Anwendungsfall auf diese Eigenschaften verzichten, um zum Beispiel zugunsten der Performance auf 100%ige Durability zu verzichten (flush_logs_at_trx_commit=2). Entscheidend ist, dass alle Möglichkeiten zur Verfügung stehen und somit verschiedene Anwendungsfälle abgedeckt werden können.

Komplexes Reporting im DocumentStore

Ein häufiges Problem beim Einsatz von NoSQL-Datenbanken besteht in der geschickten Auswertung von Daten, dem Reporting. Hier kann MySQL besondere Vorteile geltend machen. Zum einen unterstützt die X-DevAPI bereits die üblichen Filter-Prädikate und Aggregationen. (Die folgenden Beispiele basieren auf

Beispieldaten von Restaurants, die bei Document-Datenbanken gern genutzt werden. Diese können in MySQL als JSON-Dokumente geladen werden.)

```
db.restaurants.find().fields(„$.cuisine“,
„count(,*‘)“).groupBy(„$.cuisine“).sort(„$.cuisine“).execute()
```

Bemerkenswert ist hier die einfache Lesbarkeit der Query.
Der zweite Aspekt ist die Möglichkeit, die Daten jederzeit mit den umfangreichen Möglichkeiten von SQL auszuwerten. Auch in SQL kann die oben stehende Query ausgeführt werden und direkt eine Document-Collection auswerten:

```
SELECT doc->”$.cuisine”, count(*) FROM restaurants
GROUP BY doc->”$.cuisine” ORDER BY doc->”$.cuisine”;
```

Aber auch die komplexeren SQL-Konstrukte stehen nun zur Auswertung von Document-Collections zur Verfügung. MySQL 8 unterstützt beispielsweise auch einfache und rekursive Common Table Expressions („WITH-Queries“) und auch Window-Functions. Die bestbewertesten Restaurants je Cuisine können mit einem SQL-Statement ermittelt werden:

```
WITH cte1 AS
 (SELECT doc->>”$.name” AS name,
         doc->>”$.cuisine” AS cuisine,
         (SELECT AVG(score) FROM
             json_table(doc, “$.grades[*]”
                  COLUMNS (score INT PATH “$.score”)) AS r
         ) AS avg_score FROM restaurants
  )
SELECT *,
       RANK() OVER (PARTITION BY cuisine ORDER BY avg_score)
AS `rank`
FROM cte1
ORDER BY `rank`, avg_score DESC
LIMIT 30;
```

Die vergleichbare Auswertung mit Bordmitteln einer DocumentStore-Datenbank ist erheblich schwieriger oder teilweise schlicht nicht möglich.

X-DevAPI für relationale Daten

Die neue API unterstützt nicht nur DocumentStore. Es können auch relationale Tabellen mit den nativen Methoden erzeugt und manipuliert werden. Hierbei beschränkt sich die X-DevAPI auf die gewöhnlichen CRUD-Funktionen (Create, Read, Update, Delete). DDL, höherwertige komplexe Aufgaben wie Join-Operationen oder die Möglichkeiten über Common-Table-Expressions, Window-Functions oder einige Aggregationen werden weiterhin über SQL abgewickelt mit Hilfe der Methode `session.sql()`.

```
Session.sql("CREATE TABLE boroughs
             (name VARCHAR(20) PRIMARY KEY)")
db.boroughs.insert("name").values("Manhattan")
db.boroughs.insert("name").values("Staten Island")
i=db.boroughs.insert("name").values("Bronx")
i.values("Brooklyn").values("Queens").values("Missing")
db.boroughs.select().orderBy("name")
```

Mischung von Documents und relationalen Daten

Da auch Document-Collections im Hintergrund als Tabellen mit einfacher Struktur abgebildet werden, ist es möglich, Document-Collections und relationale Tabellen in der gleichen Datenbank zu speichern und diese gemeinsam in einer Applikation zu nutzen. Dies kann sogar soweit getrieben werden, dass Fremdschlüsselbeziehungen zwischen Document-Collections und relationalen Tabellen erstellt werden können. Dazu wird zunächst das gewünschte Attribut in eine automatisch generierte Spalte der Document-Collection herausgezogen:

```
session.sql("ALTER TABLE restaurants ADD COLUMN borough
varchar(20) AS (json_unquote(doc->'$.borough')) STORED")
```

Und mit Hilfe dieser Spalte wird nun die Fremdschlüsselbeziehung definiert:

```
session.sql("ALTER TABLE restaurants ADD FOREIGN KEY (borough)
REFERENCES boroughs(name);")
```

In diesem Beispiel erzwingt die Fremdschlüsselbeziehung, dass nur noch JSON-Dokumente gespeichert werden können, deren Borough-Attribut auch in der referenzierten Tabelle enthalten ist. Ein kleiner Test:

```
db.restaurants.add({name:"Marios Pizza", borough:"Milan"})
ERROR: 1452: Cannot add or update a child row: a foreign key
constraint fails…

db.restaurants.add({name:"Marios Pizza", borough:"Brooklyn"})
Query OK, 1 item affected (0.0453 sec)
```

Natürlich sind viele weitere Vorteile einer Mischung aus relationalen und dokumentbasierten Tabellen denkbar, z.B. JOIN-Operationen, Fremdschlüssel, Views, die ein Reporting aus JSON-Documents vereinfachen usw.

Vorteile der X-DevAPI

Ein wichtiger Vorteil der neuen API besteht in der sauberen Trennung von Statement und Parametern. Während bei SQL die Applikation einen String erzeugt, der sowohl Befehle als auch Parameter enthält (z.B. `INSERT INTO foo VALUES („42“);`) besteht bei der X-DevAPI eine saubere Trennung zwischen Methoden und Parametern. Dies ist u.a. der wirksamste Schutz gegen SQL-Injection, denn Anwender können nun nicht mehr Parameter eingeben, die versehentlich (oder absichtlich) als Statement interpretiert werden können.

Ein weiterer Vorteil der X-DevAPI besteht in der besseren Lesbarkeit von Code und der natürlicheren Verbindung von Applikation und Datenhaltung. Objekte können nun unkompliziert als JSON gespeichert und zurückgelesen werden. Eine oft komplexe Konvertierung in ein relationales Modell mittels ORM (Object-Relational-Mapping) kann häufig vermieden werden.

Ein dritter großer Vorteil der neuen API besteht in der nahtlosen Integration des „alten“ SQL. Damit ist ein unkomplizierter Umstieg in die neue API möglich. Daten können jederzeit durch `session.sql(...)` mit SQL-Statements manipuliert werden und die Übersichtlichkeit und Einfachheit der CRUD-API kann problemlos mit der Komplexität und Macht von SQL beim Reporting verbunden werden. Zu jedem

Zeitpunkt kann das Beste aus beiden Welten mit einem Datenbestand genutzt werden.

Document-Modell im RDBMS?

Als letztes stellt sich noch die Frage nach dem Vorteil, wenn ein Datenbanksystem sowohl für relationale Datenmodelle als auch für document-basierte Datenmodelle genutzt werden kann. Hier gibt es zwei Aspekte:

Die seit vielen Jahren vorhandenen und weiterentwickelten Fähigkeiten eines RDBMS wie ACID-Transaktionen, Foreign Keys, Methoden zum komplexen Reporting stehen schlagartig auch für den DocumentStore zur Verfügung. Während DocumentStore-Datenbanken mühsam alle vorhandenen Errungenschaften nachbilden müssen, stehen diese Features vollumfänglich in MySQL bereits zur Verfügung. Zusätzlich ergeben sich aus der Kombination von relationalen Daten und JSON-Documents viele Anwendungsfälle, die mit nur einer Technologie nicht einfach zu realisieren wären.

Der zweite Aspekt betrifft den Betrieb des Datenbanksystems. MySQL bietet ein ausgereiftes und weitentwickeltes Ökosystem von Tools und Erweiterungen, die den Betrieb vereinfachen und teilweise auch erst ermöglichen. Dies reicht von Online-Backup über diverse Monitoring-Lösungen bis hin zu Security-Erweiterungen wie transparenter Datenverschlüsselung, Auditing, feingranulare Rechteverwaltung und mehr. Alle diese Funktionalitäten stehen automatisch in bewährter Qualität zur Verfügung, um auch den DocumentStore zu betreiben. GDPR-Compliance (DSGVO) kann mit den gleichen Mitteln erreicht werden wie auch für bereits vorhandene relationale Datenbanken. Betrieb und Support erfolgen mit den gleichen Tools und Service-Providern wie bisher. Dies ist eine signifikante Vereinfachung für den Betrieb. Trotzdem erhalten Entwickler alle Möglichkeiten, die neuen Ideen der NoSQL-Datenbanken einzusetzen.

Kontakt:

M. Beck, Mario.Beck@Oracle.com

C. Thalheimer, Carsten.Thalheimer@Oracle.com