



# Modern Database Flavours

Michael Schulze, Opitz Consulting Deutschland

Nahezu jedes Unternehmen betreibt Datenbanksysteme für die Datenhaltung. Die Entwicklung von Datenbanken reicht bis in die 1960er Jahre zurück, als Mainframesysteme noch dominierten. Den Durchbruch im kommerziellen Bereich brachte das relationale Datenbankmodell (SQL-basiert), das viele Jahre Bestand hatte, bis die Datenanforderungen wuchsen und neue Datenbankkonzepte wie NoSQL in den Fokus kamen. Die heutige Datenbanklandschaft ist eine Mischung aus SQL/NoSQL DBMS-Systemen, die On-Premises und in der Cloud betrieben werden. Zudem gibt es neue Entwicklungen wie „NewSQL“. Hier werden beide Ansätze berücksichtigt und auch mit KI-/AI-Methodik ergänzt. Die Datenbanklandschaften verändern sich also. Der folgende Artikel beschäftigt sich deshalb mit dieser Entwicklung und soll einen Überblick über die verschiedenen Flavours in modernen Datenbankumgebungen geben.

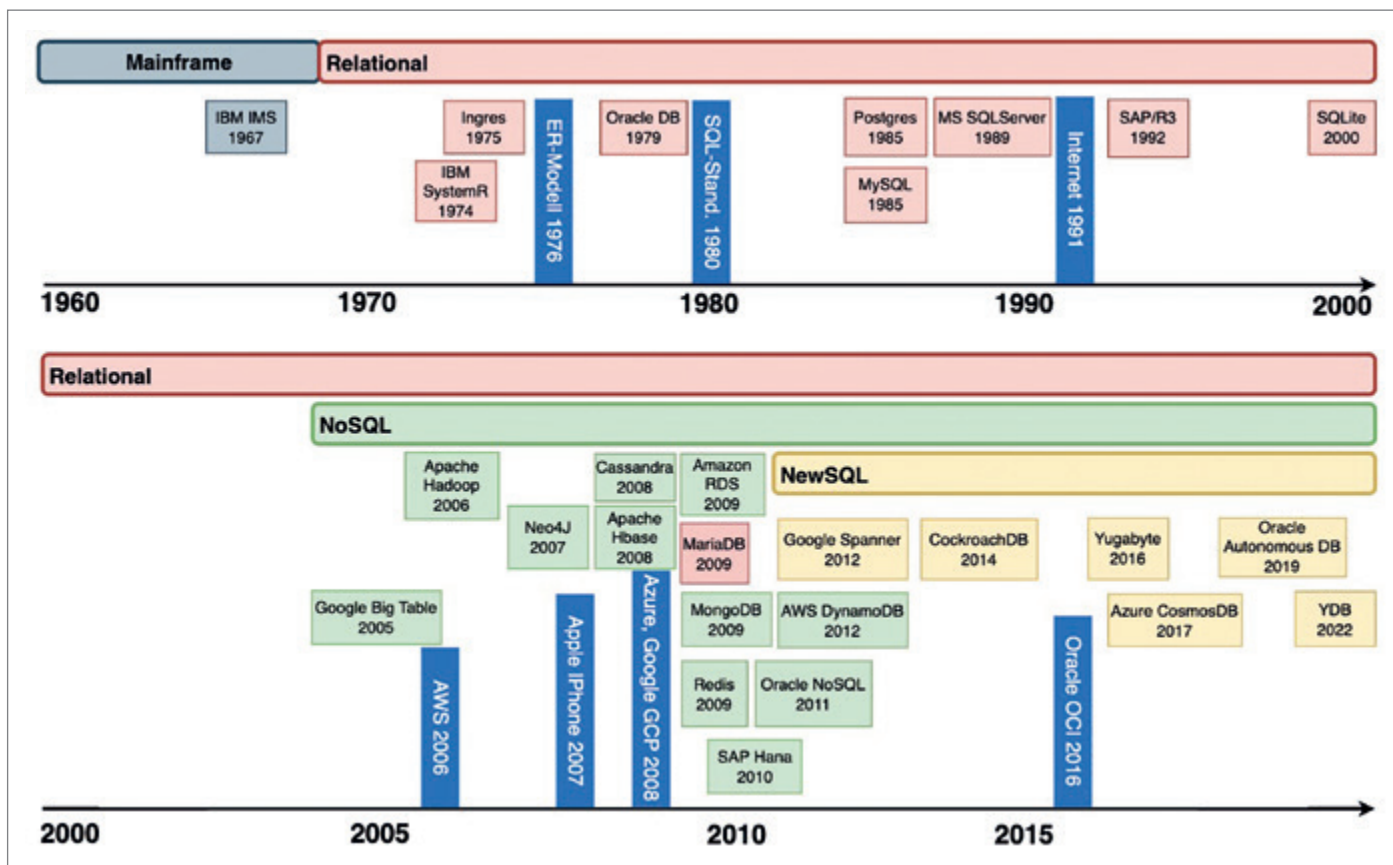


Abbildung 1: Database History 1960 bis heute (Quelle: Michael Schulze)

## Geschichte

Erste Datenbanksysteme etablierten sich in den 1960er Jahren im Mainframe-Umfeld. Hier setzte sich zum Beispiel die Firma IBM mit IMS als hierarchisches Datenbanksystem durch. Die Entwicklungen hin zu Client-Server-Umgebungen brachten mit dem relationalen Datenbankmodell, das Anfang der 1970er Jahre seinen Einzug in die IT-Landschaften hielt, neue Lösungen im Datenbankbereich hervor. Diese etablierten sich in den folgenden Jahrzehnten und sind auch heute noch zentraler Bestandteil vieler Datenbankumgebungen. Grundlage für das relationale Datenbankmodell ist das Entity-Relationship-Modell (ER-Modell), das erstmals kontextbezogene Beziehungen und damit Objektabbildungen der realen Welt in Datenbankumgebungen ermöglichte. Das relationale Datenbankmodell wird in zweidimensionalen Tabellen (Spalten/Zeilen) organisiert. Diese können in Relation zu weiteren Tabellen stehen und damit Zusammenhänge definieren. Relationale Datenbanksysteme haben einen Fokus auf Transaktionssicherheit, die insbesondere im Multiuser-

betrieb im Online Transaction Processing (OLTP) benötigt werden.

Unternehmen wie beispielsweise Oracle oder IBM etablierten das relationale Datenbankmodell im kommerziellen Enterprise-Bereich im Client-Server-Umfeld. Die verschiedenen relationalen DB-Lösungen führten zur Notwendigkeit einer Standardisierung ihrer Abfragesprache. Dieser Meilenstein wurde 1980 durch die Einführung des SQL-Standards erreicht und führte zur Verbreitung von relationalen Datenbanken. In den 1980er bis 2000er-Jahren wurden weitere Enterprise DBMS-Systeme entwickelt. Zu nennen sind hier: PostgreSQL, Microsoft SQL Server, SAP/R3 und MySQL, aber auch sehr leichtgewichtige Datenbanken wie SQLite, die sich insbesondere für kleinere Projekte eignen. Ein wichtiger Meilenstein für die weitere Verbreitung von Datenbanksystemen war 1991 die Einführung des Internets. Daraus resultierten erste Cloud-Ausprägungen (1999: Salesforce SaaS) [1]. Verschiedene Player entwickelten ab 2003 Datenbanklösungen, die den Fokus auf die Verwaltung von sehr großen Datenbeständen (Bigdata) hatten (zum Beispiel Google BigTable, Apache Ha-

doop). Diese waren unter anderem auch eine Grundlage für weitere Entwicklungen hin zu eigenständigen Public-Cloud-Lösungen. So etablierte die Firma Amazon mit Amazon Webservices (AWS) 2006 seine eigene Cloudlösung [2]. Dadurch war es nun besser möglich, standortunabhängig zu arbeiten. Die neuen Möglichkeiten brachten jetzt auch zunehmend mobile Endgeräte in den Fokus. Als ein Initiator dafür kann man hier 2007 die Marktvorstellung des ersten Apple iPhones sehen [3]. Wachsende Mobilität wirkte sich auch auf die Applikationsanforderungen aus. Verbunden mit dem zunehmenden Einsatz von Sensorik, IoT und vielem mehr, stiegen auch die Datenanforderungen stetig an. Aus dieser Notwendigkeit heraus mussten relationale Datenbanken durch neue Datenbanksysteme ergänzt werden. NoSQL-Datenbanken (Not only SQL) konnten besser zur Bewältigung großer Datenmengen bezüglich Lesegeschwindigkeit, Verteilung und Verfügbarkeit genutzt werden, als das relationale Modell. Sie verbreiteten sich ab 2005 insbesondere im Datawarehouse (DWH)- und BI-Umfeld mit verschiedenen Ausprägungen, auf die später im Artikel noch

```

SQL> -- CREATE
SQL> set feedback on
SQL> CREATE TABLE test(
  2   type VARCHAR2(10),
  3   action VARCHAR2(5),
  4   descr VARCHAR2(10)
  5* );

Table TEST created.

SQL> set feedback off
SQL> INSERT INTO test VALUES ('SQL','C','CREATE');
SQL> INSERT INTO test VALUES ('SQL','R','READ');
SQL> INSERT INTO test VALUES ('SQL','U','UPDATE');
SQL> INSERT INTO test VALUES ('SQL','D','DELETE');
SQL> commit;
SQL> -- READ
SQL> SELECT * FROM test;

TYPE      ACTION      DESCR
-----
SQL       C           CREATE
SQL       R           READ
SQL       U           UPDATE
SQL       D           DELETE
SQL> -- UPDATE
SQL> UPDATE test SET descr = 'update' WHERE action = 'U';
SQL> commit;
SQL> SELECT * FROM test where action = 'U';

TYPE      ACTION      DESCR
-----
SQL       U           update
SQL> -- DELETE
SQL> DELETE FROM test WHERE action = 'D';
SQL> commit;
SQL> SELECT * FROM test where action = 'D';

SQL> set feedback on
SQL> DROP table test;

Table TEST dropped.

```

Abbildung 2: SQLPlus-Output (Quelle: Michael Schulze)

genauer eingegangen wird. Es folgten weitere Public-Cloudlösungen insbesondere 2008: Microsoft Azure, Google Cloud Platform (GCP) und 2016 die Oracle Cloud Infrastructure (OCI) [4] (siehe Abbildung 1). Alle Lösungen brachten auf Grundlage ihrer Infrastruktur neue SQL- und NoSQL-Cloud-Datenbanken mit, die einfache bis verwaltete Services boten. So entstanden auch DB-Services, die Bestandteile von SQL und NoSQL beinhalteten. Zu nennen sind hier etwa die Oracle Autonomous

Database, Microsoft Cosmos DB, Google Cloud Spanner und moderne PostgreSQL-Ausprägungen wie Yugabyte und YDB. Diese neuen Lösungen werden auch als „NewSQL“ bezeichnet. Zudem hat sich ein Trend der Nutzung von Multicloud-Lösungen etabliert. Hier bieten die Cloudhersteller schon verschiedene Möglichkeiten an. Als ein Beispiel ist der „Oracle Database Service for Azure“ zu nennen, der seit 2023 einen verwalteten Datenbank-Service in der Oracle-Cloud für Azure-

Kunden über einen direkten Interconnect der beiden Clouds ermöglicht. Zudem lassen sich in jüngster Zeit Entwicklungen hin zu mehr Automatisierung im Datenbankbetrieb durch Nutzung und Integration von KI-/AI-Methoden erkennen.

Nach diesem Ausflug in die Historie und der Entwicklung von Datenbanksystemen möchte ich auf zwei wesentliche Konzepte eingehen und die Fakten und Ausprägungen in Kurzform beschreiben. Ein kleines Beispiel soll zudem die technischen Unterschiede jeder Ausprägung noch besser darstellen.

## Relationale Datenbanksysteme (SQL)

Eigenschaften relationaler Datenbanksysteme sind strukturierte Datendefinitionen. Daten werden in zweidimensionalen Tabellen (rowstore) gehalten, diese stehen in Beziehung zueinander und ermöglichen referenzielle Integrität. Ein wesentlicher Bestandteil ist die Structured Query Language (SQL), die als Abfragesprache verwendet wird. Relationale Datenbanken verfolgen das Konzept der Transaktions-sicherheit, Konsistenz und Datenintegrität (ACID). Hier spielen die Einhaltung von Faktoren wie Atomarität, Konsistenz, Isolation und Dauerhaftigkeit der Datenerhaltung eine zentrale Rolle. Deshalb sind die Systeme sehr gut für den Multiuser-Betrieb mit vielen Transaktionen (Schreib- und Lesezugriffen) geeignet.

Neben „Online Transaction Processing“ (OLTP)-Datenbanken mit aktivem Datenbestand und Größenordnungen im GB- bis TB-Bereich gibt es weitere Ausprägungen wie Online Analytical Processing (OLAP). Diese oft großen Datenbanken im Terabyte-/Petabyte-Bereich sind mehrdimensional organisiert und führen mehrere (aktive) Datenbestände durch ETL-Prozesse in passive Datenbestände zusammen, die der Analyse dienen. OLAP-Datenbanken werden deshalb vorrangig im Datawarehouse-/BI-Umfeld eingesetzt. Hier liegt der Fokus mehr auf den lesenden Zugriffen komplexer SQL-Statements. Prominente Hersteller von relationalen Datenbanken sind zum Beispiel Oracle, Microsoft, IBM oder SAP.

Vorteile relationaler SQL-Datenbanken liegen in der ACID-konformen Datenkonsistenz, der flexiblen Datenstrukturie-






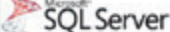








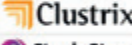




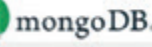



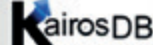

<b>Columnar</b>	 cassandra	 DATASTAX	 APACHE HBASE		DWH, Big Data
<b>Spatial</b>	 snowflake	 ORACLE	 Microsoft SQL Server		Geodata
<b>Object</b>	 Couchbase	 ZooDB The ZOO Database			Object orientiert
<b>Key-Value</b>	 redis	 amazon DynamoDB	 ORACLE NOSQL DATABASE		Simple data model
<b>NewSQL</b>	 CockroachDB	 VOLTDB	 nuODB	 Clustrix  SingleStore	Wieder ACID + SQL
<b>Graph</b>	 neo4j	 Amazon Neptune	 JanusGraph		Social Media
<b>Document</b>	 mongoDB.	 Couchbase	 amazon DynamoDB		Read++, JSON
<b>Time Series</b>	 influxdb	 KairosDB	 ClickHouse		Timestamp + Data, Sensoren, Kurse..

Abbildung 3: Overview NoSQL Databases (Quelle: Michael Schulze)

zung mit komplexen Datenbeziehungen, der Verwendung von SQL und der Datensicherheit. Ein klarer Nachteil, bedingt durch den Fokus auf die Transaktionssicherheit, sind fehlende horizontale Skalierungsmöglichkeiten, höherer Wartungsaufwand sowie Performanceverluste durch wachsende und komplexe Datenbestände mit vielen Objekten.

Um die Unterschiede von SQL/NoSQL noch besser zu veranschaulichen habe ich ein Create-Read-Update-Delete (CRUD)-Beispiel vorbereitet, das elementare Grundlagen beinhaltet und auf beide DB-Konzepte exemplarisch angewendet wird.

## SQL-Beispiel

Im folgenden SQL-Beispiel wurde als Usecase eine Oracle-Datenbank und SQLPlus als Schnittstelle genutzt. Das Beispiel ist geläufig und wird daher nur kurz beschrieben (siehe Abbildung 2).

CREATE: Grundlage für alle weiteren SQL-Befehle im Skript ist die Erstellung eines Tabellen-Objekts. Ist dieses vorhanden, können nun Datensätze durch INSERT hinzugefügt werden. Dieser Schritt muss durch ein commit; abgeschlossen werden, damit die Transaktion(en) festgeschrieben und die Änderungen für andere Benutzer sichtbar gemacht werden.

READ: Das Auslesen von Daten erfolgt mittels SELECT. Damit können Daten durch SQL-Syntax Daten gefiltert werden.

UPDATE: Hierbei wird ein vorhandener Datensatz manipuliert, auch diese Transaktion wird erst für andere sichtbar, wenn ein commit; erfolgt.

DELETE: Als letzter Schritt im CRUD-Beispiel erfolgt das Löschen von Daten. Zudem wird das Tabellenobjekt durch DROP wieder aus der Datenbank entfernt.

## NoSQL-Datenbanksysteme

NoSQL-Datenbanken organisieren die Daten unstrukturiert/teilstrukturiert und haben weniger Abhängigkeiten. Das ermöglicht die Verwaltung sehr großer Datenbestände, horizontale Skalierungsmöglichkeiten (Verteilung auf mehrere Systeme) und dadurch Performancevorteile. Es gibt keine Abhängigkeiten zu Schemaobjekten und auch keine Abfragesprache wie SQL, alle notwendigen Informationen werden direkt mit der entsprechenden Datenbankaktion mitgegeben. NoSQL-Datenbanken verfolgen das BASE-Konzept, das einen starken Fokus auf Verfügbarkeit und Geschwindigkeit setzt. Durch die Anwendung der horizontalen Skalierung müssen die Daten nach Änderung auf die einzelnen Knoten verteilt werden,

was einen kleinen Zeitversatz zur Folge hat. Deshalb ist hier keine Transaktionssicherheit gewährleistet, die Daten sind „eventually consistent“ und man muss softwareseitig Wege finden, um deren Konsistenz sicherzustellen. Aus diesem Grund eignen sich NoSQL-Datenbanken auch eher für einfache Datenmodelle mit vielen Daten und mit einem starken Lesefokus, zum Beispiel einem Dashboard. NoSQL-Datenbanken existieren in verschiedenen Ausprägungen wie etwa document, key-value, graph, column-store (für DWH) und vielen weiteren. Bekannte Vertreter/Produkte sind hier beispielsweise MongoDB, Cassandra, Redis, Couchbase und Oracle NoSQL.

Vorteile von NoSQL-Datenbanken liegen klar in der horizontalen Skalierung. Resultierend daraus, haben sie eine hohe Abfrageperformance auch bei wachsenden Datenbeständen, weniger Abhängigkeiten, besserer Verfügbarkeit und Flexibilität. Aufgrund der Skalierung sind NoSQL-Systeme ideal für Cloud-Umgebungen geeignet. Nachteile sind in der Inkonsistenz bei Datenabruf, der Ineffizienz bei komplexen Abfragen und der fehlenden Abfragesprache zu sehen.

Die folgende Übersicht klassifiziert aktuelle NoSQL-Datenbanksysteme hinsichtlich Ausprägung und Anwendungsfall (siehe Abbildung 3).

```

test> db.test.insertMany([
... { type: "NoSQL", action: "C", descr: "CREATE"},
... { type: "NoSQL", action: "R", descr: "READ"},
... { type: "NoSQL", action: "U", descr: "UPDATE"},
... { type: "NoSQL", action: "D", descr: "DELETE"},
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6578abd3e360e063e0d33562"),
    '1': ObjectId("6578abd3e360e063e0d33563"),
    '2': ObjectId("6578abd3e360e063e0d33564"),
    '3': ObjectId("6578abd3e360e063e0d33565")
  }
}
test> db.test.find({ action: "R" })
[
  {
    _id: ObjectId("6578abd3e360e063e0d33563"),
    type: 'NoSQL',
    action: 'R',
    descr: 'READ'
  }
]
test> db.test.updateOne({ action: "U" }, {$set: { action: "update" }})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test> db.test.deleteMany({ action: "D" })
{ acknowledged: true, deletedCount: 1 }
test> db.test.deleteMany({})
{ acknowledged: true, deletedCount: 3 }
    
```

Abbildung 4: mongosh-Output (Quelle: Michael Schulze)

onen werden dem Befehl direkt als JSON-String mitgegeben. `db.test.insertMany` erzeugt die Collection „test“. In dieser Datenstruktur werden die Daten gespeichert. Ein „commit“ oder ähnliches gibt es hier nicht, da NoSQL auf Verfügbarkeit (BASE) und nicht auf Transaktionssicherheit (ACID) konzipiert ist. Mit `db.test.insertOne` können einzelne Daten hinzugefügt werden. Zudem ist es möglich, eine leere Collection mit `db.createCollection("test")` zu erzeugen.

READ: Das Auslesen von Daten erfolgt mittels `db.test.find`. Damit ist eine Daten Selektion möglich. Als Ergebnis werden alle passenden Daten und die entsprechende ObjectID zurückgegeben.

UPDATE: Mit `db.test.updateOne` werden Daten in der Collection „test“ aktualisiert. Als Parameter werden das Selektionskriterium und die Änderungswerte angegeben. Ferner gibt es mit `db.<collection>.updateMany` die Möglichkeit, viele Daten en bloc zu ändern.

DELETE: Als letzter Schritt im Beispiel, wird das Löschen von Daten beschrieben. Mit `db.test.deleteOne` und Parameter `<ObjectID>` werden Daten aus der Collection-Struktur entfernt. Ein Selektionskriterium kann an dieser Stelle nicht angegeben werden. Diese Möglichkeit gibt es aber mit `db.<collection>.deleteMany`.



Abbildung 5: Oracle Autonomous Database (Quelle: Michael Schulze)

### NewSQL – eine Mischung aus beiden Konzepten

Zielstellungen von „NewSQL“-Lösungen sind JSON-Integration, Schnittstellen zu anderen Datenbanken, das Aufbrechen von monolithischen Strukturen relationaler Datenbanken sowie mögliche horizontale Skalierung im Cloud-Umfeld mit Transaktionssicherheit. Diese Datenbanken etablieren sich seit einiger Zeit.

Als Vertreter sind hier unter anderem die Oracle Autonomous Database (siehe Abbildung 5), Microsoft Cosmos DB, Google Cloud Spanner und moderne Postgres-Ausprägungen wie Yugabyte und YDB zu nennen.

### NoSQL-Beispiel

Es folgt in Analogie das CRUD-Beispiel in einer NoSQL-Variante (dokumentenbasiert). Als Datenbank wurde hier Mon-

goDB (mongosh CLI) [5] genutzt (siehe Abbildung 4).

CREATE: Wichtig ist hier zu wissen, dass es hier keine Abhängigkeiten zu Schemaobjekten gibt. Alle notwendigen Informati-

### Datenbank-Cloud-Entwicklung und weiterer Ausblick

Aktuelle Datenbankumgebungen müssen immer größer werdenden Daten-



Abbildung 6: Gartner Magic Quadrant for Cloud DBMS-Systems (2023) (Quelle: Gartner)

mengen gerecht werden, skalierbar, verfügbar und kosteneffizient sein. Sie sind insgesamt auch stark von den sich ändernden Anforderungen an die Applikationen abhängig. Man erkennt im Gartner Quadrant (2023) [6], dass bei Cloud-Datenbanklösungen hier zunehmend horizontal skalierbare NoSQL-Lösungen neben relationalen Datenbanken eine große Rolle spielen (siehe Abbildung 6). Der Move-to-Cloud-Prozess ist in vielen Unternehmen schon vollzogen worden, beziehungsweise in naher Zukunft fest geplant. Dem Trend, dass Kunden zunehmend Multicloud-Lösungen flexibel nutzen, folgen auch die Cloud-Anbieter.

Als ein Beispiel ist der „Oracle Database Service for Azure“ zu nennen, der seit 2023 einen verwalteten DB-Service in der Oracle-Cloud für Azure-Kunden über einen direkten Interconnect der beiden Cloudlösungen ermöglicht.

Die weitere Automatisierung von Datenbanken durch Nutzung und Integration von KI-/AI-Methoden wird zukünftig zu schnelleren und effizienteren Datenban-

ken führen, die sich selbst monitoren und automatisch agieren.

Beispiele dafür sind die „Oracle Autonomous Database“, die schon einen Großteil von Automatismus beinhaltet. Mit der Integration von „select AI“ sind hier weitere Schritte in Richtung KI/AI geplant [7].

## Quellen

- [1] <https://www.salesforceben.com/salesforce-history/>
- [2] <https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/>
- [3] <https://www.apple.com/de/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/>
- [4] <https://www.oracle.com/a/ocom/docs/cloud/oracle-cloud-infrastructure-platform-overview-wp.pdf>
- [5] <https://www.mongodb.com/docs/manual/crud/>
- [6] <https://www.gartner.com/doc/reprints?id=1-2G077EWH&ct=231221>
- [7] <https://blogs.oracle.com/machine-learning/post/introducing-natural-language-to-sql-generation-on-autonomous-database>

## Über den Autor

Michael Schulze ist ein Infrastrukturexperte mit mehr als 25 Jahren Berufserfahrung in Unix- und Oracle-Projekten. Sein Spektrum bei Opitz Consulting Deutschland GmbH umfasst als Solution-Architekt die Bewertung, Konzeption und Erstellung von Infrastrukturmgebungen im Datenbank- und Middleware-Umfeld. Ein starker Fokus liegt hier auf Oracle-Produkten wie der Oracle-Datenbank, HA-Lösungen, Engineered Systems, KVM-Virtualisierung, Automatisierungs- und Cloudlösungen. Als Autor für das Red Stack-Magazin und DOAG-Konferenz-Speaker liefert er regelmäßig Beiträge für die Oracle-Community in den verschiedenen Kontexten.



Michael Schulze  
michael.schulze@opitz-consulting.com