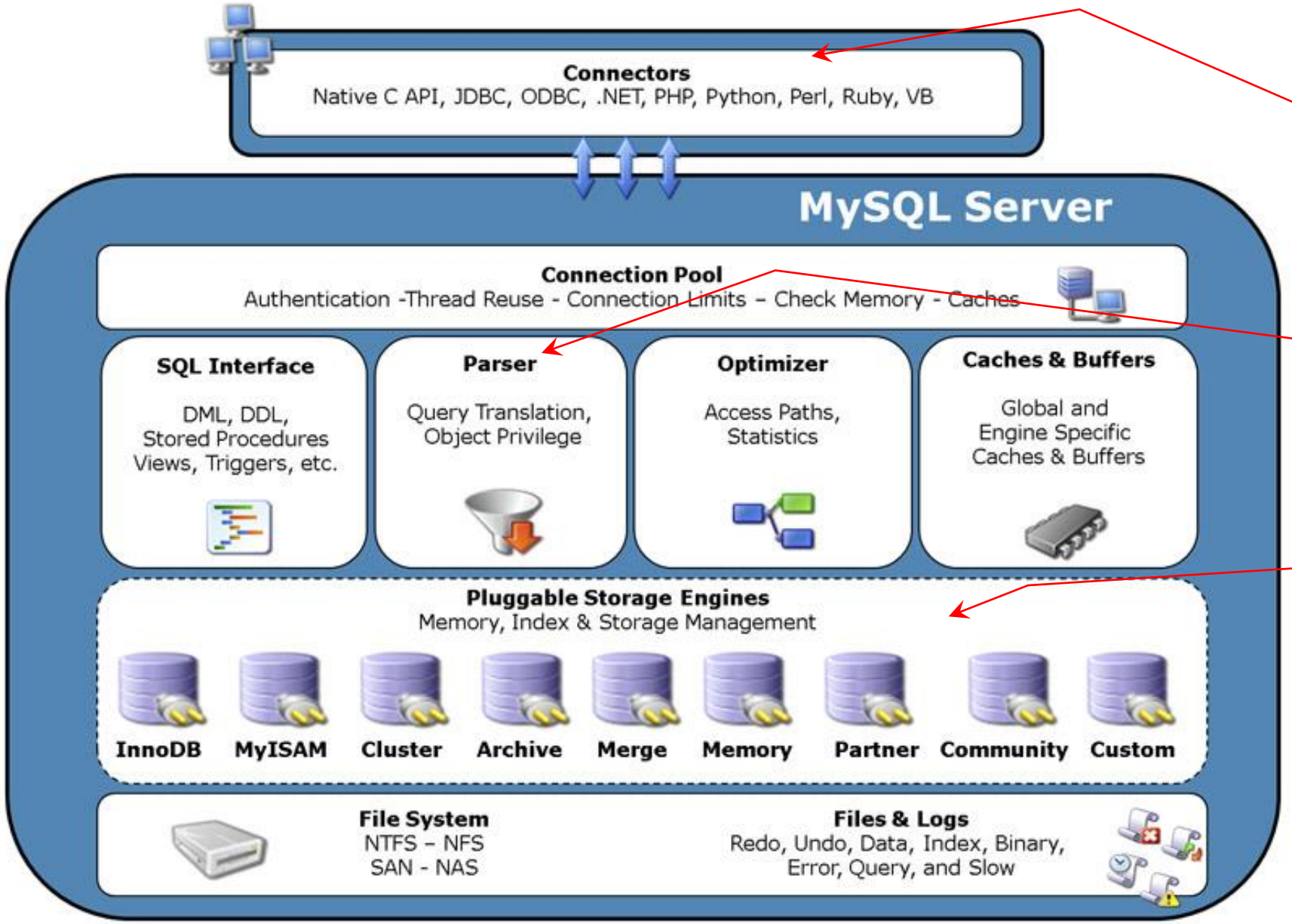
Datenbanken und SQL

komplexe Abfragen, Subselect

Wie wird SQL verarbeitet?

Beispiel: MySQL-Architektur



Wie wird SQL verarbeitet?

- SQL ist eine Interpreter-Sprache → Anweisung wird von einem „Parser“ gelesen, die „Rechtschreibung“ = Syntax geprüft, der Sinn, die Funktion = die Semantik ermittelt und in einen ausführbaren Code übersetzt und ausgeführt
- Skripte werden zeilenweise abgearbeitet
- Anweisungen werden von „innen“ nach „außen“ abgearbeitet → erst die Klammern, dann das Drumrum
- Ausdrücke und Operatoren werden nach ihrer Priorität abgearbeitet (siehe „Punktrechnung geht vor Strichrechnung“ usw.)

Theoretische Grundlage ist die Relationenalgebra

- Relationenalgebra bzw. relationale Algebra enthält Operationen/ Regeln, über die Relationen = Tabellen = Mengen miteinander verknüpft werden können (Erfinder ist Codd)
- Was bedeutet der Begriff „Algebra“?
- die Verknüpfung von Relationen erzeugt neue Relationen
- grundlegende Operationen sind:
  - Projektion
  - Selektion
  - Kreuzprodukt
  - Vereinigung
  - Differenz
  - Umbenennung

Projektion

Attribut 1	Attribut 2	Attribut 3	Attribut 4	Attribut 5	Attribut 6
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Projektion

Attribut 1	Attribut 3	Attribut 4	Attribut 6
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

SELECT * FROM <table>

SELECT attribut1, attribut2 FROM <table>

Selektion

- Selektion
  - SELECT * FROM … WHERE … (→ Selektionsbedingung)

- Varianten und Operatoren
  - Vergleichsoperatoren = < >
  - Verknüpfungsoperatoren AND, OR
  - Listenabfrage IN
  - mit Platzhalter %, _, [] und LIKE
  - mit TOP n bzw. TOP n PERCENT
  - mit Begrenzung der Zeilenanzahl für alle folgenden Abfragen: SET ROWCOUNT n (Rücksetzen mit n=0)
  - Funktionen (für SELECT-Befehle)

- Unterdrückung doppelter/ mehrfacher Datensätze
  - SELECT DISTINCT * FROM … WHERE

Selektion

Attribut 1	Attribut 2	Attribut 3	Attribut 4	Attribut 5	Attribut 6
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Selektion

Attribut 1	Attribut 2	Attribut 3	Attribut 4	Attribut 5	Attribut 6
2					
3					
5					
6					
7					
10					

SELECT * FROM <table>

SELECT attribut1, attribut2 FROM <table>
WHERE <bedingung>

Logische Verarbeitungsreihenfolge der SELECT-Anweisung

- Die folgenden Schritte beschreiben die logische Verarbeitungs- oder Bindungsreihenfolge der SELECT-Anweisung.
- Diese Reihenfolge bestimmt, wann die in einem Schritt definierten Objekte in nachfolgenden Schritten für die Klauseln verfügbar gemacht werden.
  - Wenn der Abfrageprozessor z. B. eine Bindung mit den in der FROM-Klausel definierten Tabellen oder Sichten herstellen (bzw. darauf zugreifen) kann, werden diese Objekte und die dazugehörigen Spalten für alle nachfolgenden Schritten verfügbar gemacht.
  - Umgekehrt kann auf die in dieser Klausel definierten Spaltenaliase oder abgeleiteten Spalten nicht durch vorhergehende Klauseln verwiesen werden, da die SELECT-Klausel Schritt 8 ist.
  - Allerdings kann durch nachfolgende Klauseln wie der ORDER BY-Klausel darauf verwiesen werden.
  - Die tatsächliche physische Ausführung der Anweisung wird durch den Abfrageprozessor bestimmt und die Reihenfolge kann von dieser Liste abweichen.

- Bindungsreihenfolge:
  (1) from
  (2) ON
  (3) JOIN
  (4) where
  (5) GROUP BY
  (6) WITH CUBE oder WITH ROLLUP
  (7) HAVING
  (8) SELECT
  (9) DISTINCT
  (10) ORDER BY
  (11) TOP bzw. LIMIT

Logische Verarbeitungsreihenfolge der SELECT-Anweisung

```
SELECT    b.KUNDENNR,
          k.NAME,
          k.ORT,
          COUNT(b.kundennr) 'Anzahl Bestellungen'

FROM BESTELLUNG b JOIN KUNDE k ON b.KUNDENNR = k.KUNDENNR

GROUP BY b.KUNDENNR, k.name, k.ORT

ORDER BY COUNT(b.kundennr) DESC, k.NAME ASC
```

Logische Verarbeitungsreihenfolge der SELECT-Anweisung

Ergebnisse | Meldungen

	KUNDENNR	NAME	ORT	Anzahl der Bestellungen für diesen Kunden in dieser T...
1	4	Falkner	Wiesbaden	3
2	17	Grunpeter	Bonn	3
3	72	Hart	Hannover	3
4	63	Kampmann	Köln	3
5	23	Knutt	Aachen	3
6	29	Mann	Weimar	3
7	47	Masur	Hamburg	3
8	39	Palk	Bonn	3
9	73	Ure	Köln	3
10	9	Voglin	Berlin	3
11	2	Adler	Hannover	2
12	13	Badel	Köln	2
13	42	Clement	Berlin	2
14	8	Dietrich	Wiesbaden	2
15	83	Durdan	Frankfurt...	2
16	36	Fabrizi	Köln	2
17	12	Fromkess	Bonn	2
18	15	Front	Hamburg	2
19	48	Gross	Berlin	2
20	25	Grunert	Hamburg	2
21	43	Handl	Köln	2
22	61	Hecht	Wiesbaden	2
23	88	Hordern	Hamburg	2

Ergebnisse einer SELECT-Abfrage

- Was ist das Ergebnis einer SELECT-Abfrage? Welche Ergebnisse kann ein SELECT liefern?
  - select address, district, postal_code from address where city_id = 312;
    liefert eine durch Projektion und Selektion bestimmte Ergebnistabelle (tabellenwertiges Ergebnis)
  - select district from address where postal_code like '27%';
    liefert eine durch Projektion und Selektion bestimmte Ergebnisspalte (Liste)
  - SELECT SUM(amount) FROM payment;
    liefert einen einzigen Wert als Ergebnis der Anwendung einer Aggregatfunktion auf die Projektion einer Spalte (skalares Ergebnis)
  - SELECT SUM(amount), avg(amount) FROM payment;
    liefert zwei Werte als Ergebnis der Anwendung je einer Aggregatfunktion auf die Projektion zweier Spalten der Tabelle (mehrwertiges Ergebnis)
- die SELECT-Ausgabe der Spalten einer Ergebnistabelle und die Ausgabe des Ergebnisses einer oder mehrerer Aggregatfunktionen kann nie gleichzeitig in einem SELECT-Befehl erfolgen
- Wie können diese Ergebnisse weiter verarbeitet werden in einer Anweisung?

Verwendung von Unterabfragen = subselect

SELECT: Abfragen auf eine oder mehrere Tabellen mit Unterabfragen (= subselect)

- wenn Abfragen (= SELECT) direkt die Ergebnisse einer anderen Abfrage für die Formulierung eines logischen Ausdrucks im WHERE nutzen, spricht man von einem subselect oder auch subquery
- Subselects können **einen** oder **mehrere** Werte bzw. Zeilen zurückliefern --> siehe „Ergebnisse einer SELECT-Abfrage“
- Subselects können in UPDATE, DELETE, INSERT und SELECT verwendet werden

SELECT: Abfragen auf eine oder mehrere Tabellen mit Unterabfragen (= subselect)

Beispiel: Bildung eines logischen Ausdrucks mit dem Vergleichsoperator >= sowie einem SELECT und einer Aggregatfunktion auf eine Spalte
(möglich sind =, < >, >, > =, <, ! >, ! < oder < =)

```
select *
from payment
where amount >= (select avg(amount) from payment);
```

SELECT: Abfragen auf eine oder mehrere Tabellen mit Unterabfragen (= subselect)

Beispiel: Bildung eines logischen Ausdrucks mit dem Listenoperator IN sowie einem SELECT und einer Projektion auf eine Spalte (und Selektion von Datensätzen)

```
SELECT
 *
FROM customer
WHERE address_id IN (select address_id from address where district like 'd%');

--                          hier wird die Liste für IN erzeugt
```

SELECT: Abfragen auf eine oder mehrere Tabellen mit Unterabfragen (= subselect)

Beispiel: Bildung eines logischen Ausdrucks mit EXISTS und einem SELECT, das zwei Tabellen verknüpft

```
SELECT
        *

FROM bestellung

WHERE EXISTS
(SELECT * FROM kunde WHERE
kunde.kundennr=bestellung.kundennr AND kunde.plz LIKE '4%')
```

Praktisches Beispiel: INSERT mit SUBSELECT

Beispiel: Verwendung eines subselects, um ausgewählte Daten aus einer Tabelle in eine andere zu kopieren

```
INSERT INTO tbl_mitarbeiter_tmp (mitarbeiternummer,
mitarbeitername, maschinenberechtigung, abteilungsnummer,
mitarbeiterPLZ)
SELECT mitarbeiternummer, mitarbeitername,
maschinenberechtigung, abteilungsnummer,
mitarbeiterPLZ FROM tbl_mitarbeiter
WHERE abteilungsnummer = 9
```

Unterscheidung von subselects

- subselects können in
  - **korrelierende subselects** und
  - **nicht korrelierende subselects** unterschieden werden (selbständige s.)
- bei einem nicht korrelierenden subselect funktioniert die Unterabfrage vollständig unabhängig von der übergeordneten Abfrage, kann also zunächst völlig unabhängig entwickelt und getestet werden
  BEISPIEL:
- HINWEISE:
  - oft kann man das gleiche Ergebnis mit Unterabfrage oder (unterschiedlichen) JOIN erreichen → aber die Laufzeit/ Performance kann sehr unterschiedlich sein
  - möglichst wenig oder keine Unterabfragen verwenden und dafür JOIN verwenden

korrelierende und nicht korrelierende subselects

- bei einem nicht korrelierenden subselect funktioniert die Unterabfrage vollständig unabhängig von der übergeordneten Abfrage, kann also zunächst völlig unabhängig entwickelt und getestet werden
  BEISPIEL: siehe Skript
- die Ergebnisse der selbständigen subselects wird mit den bekannten Vergleichsoperatoren bzw. mit Funktionen in der übergeordneten Funktion verarbeitet.

- bei korrelierenden subselects wird zwischen der übergeordneten Abfrage und dem subselect eine Verbindung über Aliasse hergestellt
  BEISPIEL: siehe Skript