

DATENBANKANWENDUNG

Wintersemester 2013/2014

PD Dr. Holger Schwarz
Universität Stuttgart, IPVS
holger.schwarz@ipvs.uni-stuttgart.de

Beginn: 23.10.2013
Mittwochs: 11.45 – 15.15 Uhr, Raum 46-268 (Pause 13.00 – 13.30)
Donnerstags: 10.00 – 11.30 Uhr, Raum 46-268
11.45 – 13.15 Uhr, Raum 46-260

<http://www.lgis.informatik.uni-kl.de/cms/courses/datenbankanwendung/>



Kopplung

stat. SQL

Anfrage-
auswertung

Anfrage-
optimierung

SQL/PSM

dyn. SQL

CLI u. ODBC

JDBC u. SQL

MapReduce

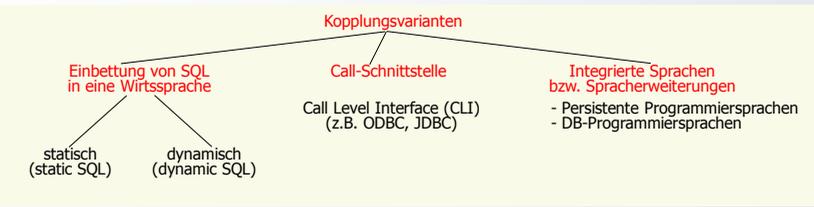


4. Anwendungsprogrammierschnittstellen

- **Kopplung mit einer Wirtssprache**
 - Übersicht und Aufgaben
- **Eingebettetes statisches SQL**
 - Cursor-Konzept
 - SQL-Programmiermodell
 - Rekursion
 - Ausnahme- und Fehlerbehandlung
- **Aspekte der Anfrageauswertung**
 - Aufgaben bei der Anfrageauswertung
 - Vorbereitung und Übersetzung
- **Optimierung von DB-Anfragen**
 - Probleme der Optimierung
 - Kostenmodell
 - Bindung und Datenunabhängigkeit
- **SQL/PSM**
- **Dynamisches SQL**
 - Eingebettetes dynamisches SQL
- **CLI und ODBC**
 - Call-Level-Interface
 - Open Data Base Connectivity (ODBC)
- **DB-Zugriff aus Java-Programmen**
 - DB-Zugriff via JDBC
 - SQLJ
- **MapReduce-Paradigma**

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Kopplung mit einer Wirtssprache

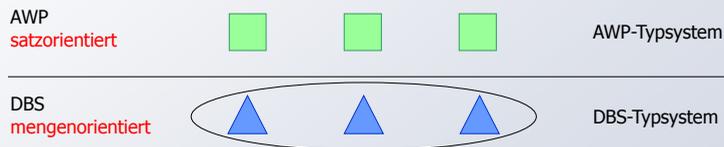


- **Einbettung von SQL (Embedded SQL, ESQL)**
 - Spracherweiterung um spezielle DB-Befehle (EXEC SQL ...)
 - komfortablere Programmierung als mit CLI
- **Statische Einbettung**
 - Vorübersetzer (Precompiler) wandelt DB-Aufrufe in Prozeduraufrufe um
 - Nutzung der normalen PS-Übersetzer für umgebendes Programm
 - SQL-Anweisungen müssen zur Übersetzungszeit feststehen
 - im SQL-Standard unterstützte Sprachen: C, COBOL, FORTRAN, Ada, PL1, Pascal, MUMPS, Java, ...
- **Dynamische Einbettung**
 - Konstruktion von SQL-Anweisungen zur Laufzeit
- **Call-Schnittstelle (prozedurale Schnittstelle, CLI)**
 - DB-Funktionen werden durch Bibliothek von Prozeduren realisiert
 - Anwendung enthält lediglich Prozeduraufrufe

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Kopplung mit einer Wirtssprache (2)

- **Integrationsansätze unterstützen typischerweise nur**
 - ein Typsystem
 - Navigation (satz-/objektorientierter Zugriff)
 - ⇒ Wünschenswert sind jedoch Mehrsprachenfähigkeit und deskriptive DB-Operationen (mengenorientierter Zugriff)
- **Relationale AP-Schnittstellen (API) bieten diese Eigenschaften**
 - erfordern jedoch Maßnahmen zur Überwindung der sog. Fehlanpassung (impedance mismatch)



- **Kernprobleme der API bei konventionellen Programmiersprachen**
 - Konversion und Übergabe von Werten
 - Übergabe aktueller Werte von Wirtssprachenvariablen (Parametrisierung von DB-Operationen)
 - DB-Operationen sind i. allg. mengenorientiert: Wie und in welcher Reihenfolge werden Zeilen/Sätze dem AP zur Verfügung gestellt?
- ⇒ Cursor-Konzept

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Eingebettetes statisches SQL

■ Beispiel für C

```

exec sql include sqla; /* SQL Communication Area */
main ()
{
exec sql begin declare section;
    char X[3];
    int GSum;
exec sql end declare section;
exec sql connect to dbname;
exec sql insert into Pers (Pnr, Name) values (4711, 'Ernie');
exec sql insert into Pers (Pnr, Name) values (4712, 'Bert');
    printf ("Anr ? "); scanf ("%s", X);
exec sql select sum (Gehalt) into :GSum from Pers where Anr = :X;
    /* Es wird nur ein Ergebnissatz zurückgeliefert */
    printf ("Gehaltssumme: %d\n", GSum)
exec sql commit work;
exec sql disconnect;
}
    
```

Anbindung einer SQL-Anweisung an die Wirtssprachen-Umgebung

- SQL-Anweisungen werden durch **exec sql** eingeleitet und durch spezielles Symbol (hier ";") beendet, um dem Compiler eine Unterscheidung von anderen Anweisungen zu ermöglichen
- Verwendung von AP-Variablen in SQL-Anweisungen verlangt Deklaration innerhalb eines **declare section**-Blocks sowie Angabe des Präfix ":" innerhalb von SQL-Anweisungen
- Kommunikationsbereich SQLCA (Rückgabe von Statusanzeigern u.ä.)
- Übergabe der Werte einer Zeile mit Hilfe der INTO-Klausel
 - INTO target-commalist (Variablenliste des Wirtsprogramms)
 - Anpassung der Datentypen (Konversion)
- Aufbau/Abbau einer Verbindung zu einem DBS: **connect/disconnect**

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Cursor-Konzept

■ Cursor-Konzept zur satzweisen Abarbeitung von Ergebnismengen

- Trennung von Qualifikation und Bereitstellung/Verarbeitung von Zeilen
- Cursor ist ein Iterator, der einer Anfrage zugeordnet wird und mit dessen Hilfe die Zeilen der Ergebnismenge einzeln (one tuple at a time) im Programm bereitgestellt werden
- Wie viele Cursor können im AWP sein?

■ Cursor-Deklaration

```

DECLARE cursor
CURSOR FOR table-exp
[ORDER BY order-item-commalist]
    
```

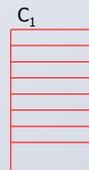
```

DECLARE C1 CURSOR FOR
SELECT Name, Gehalt, Anr
FROM Pers WHERE Anr = 'K55'
ORDER BY Name;
    
```

■ Operationen auf einen Cursor C1

```

OPEN C1
FETCH C1 INTO Var1, Var2, ..., Varn
CLOSE C1
    
```

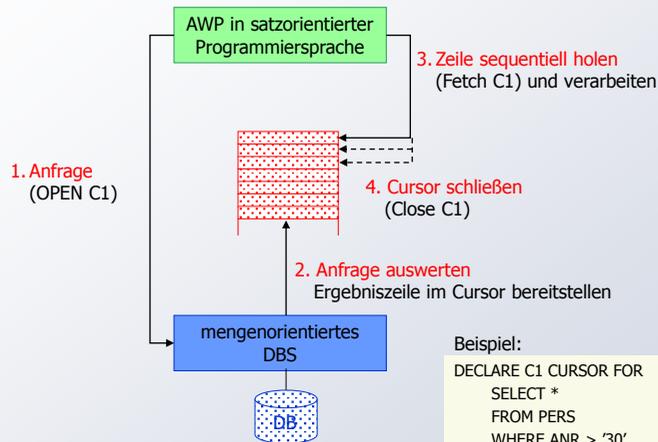


■ Reihenfolge der Ergebniszeilen

- systembestimmt
- benutzerspezifiziert (ORDER BY)

Cursor-Konzept (2)

Veranschaulichung der Cursor-Schnittstelle



Wann wird die Ergebnismenge angelegt?

- lazy:** schritt haltende Auswertung durch das DBS? Verzicht auf eine explizite Zwischenspeicherung ist nur bei einfachen Anfragen möglich
- eager:** Kopie bei OPEN? Ist meist erforderlich (ORDER BY, Join, Aggregat-Funktionen, ...)

Cursor-Konzept (3)

Beispielprogramm in C (vereinfacht)

```

exec sql begin declare section;
char X[50], Y[3];
exec sql end declare section;
exec sql declare C1 cursor for
  select Name from Pers where Anr = :Y;

printf("Bitte Anr eingeben: \n");
scanf("%d", Y);
exec sql open C1;
while (sqlcode == OK)
{
  exec sql fetch C1 into :X;
  printf("Angestellter %d\n", X);
}
exec sql close C1;
    
```

Anmerkungen

- DECLARE C1 ... ordnet der Anfrage einen Cursor C1 zu
- OPEN C1 bindet die Werte der Eingabevariablen
- Systemvariable SQLCODE zur Übergabe von Fehlermeldungen (Teil von SQLCA)

Cursor-Konzept (4)

Aktualisierung mit Bezugnahme auf eine Position

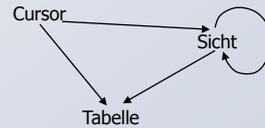
- Wenn die Zeilen, die ein Cursor verwaltet (*active set*), eindeutig Zeilen einer Tabelle entsprechen, können sie über Bezugnahme durch den Cursor geändert werden.
- Keine Bezugnahme bei INSERT möglich !

```
positioned-update ::=
    UPDATE table SET update-assignment-commalist
    WHERE CURRENT OF cursor
Positioned-delete ::=
    DELETE FROM table
    WHERE CURRENT OF cursor
```

Beispiel

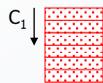
```
while (sqlcode == ok) {
    exec sql fetch C1 into :X;
    /* Berechne das neue Gehalt in Z /*
    exec sql update Pers
    set Gehalt = :Z
    where current of C1;
}
```

Vergleich: Cursor - Sicht

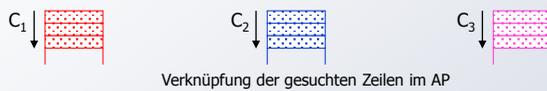


SQL-Programmiermodell für Mengenzugriff

1) ein Cursor: $\pi, \sigma, \bowtie, \cup, -, \dots$, Agg, Sort, ...



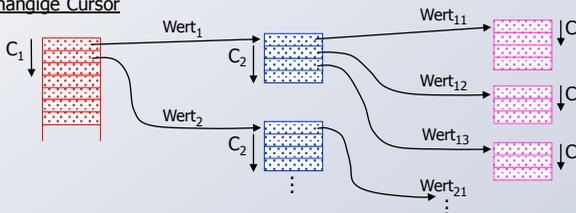
2) mehrere Cursor: π, σ , Sort, ...



3) positionsbezogene Aktualisierung

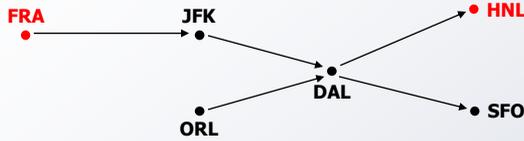


4) abhängige Cursor



- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Rekursion in SQL?



Ausschnitt aus Tabelle Flüge

Flüge	(Nr,	Ab,	An,	Ab-Zeit,	An-Zeit, ...)
AA07	FRA	JFK			
AA43	JFK	ORL			
AA07	JFK	DAL			
AA85	ORL	DAL			
AA70	DAL	HNL			
...					

Flug von FRA nach HNL?

```

SELECT Ab, An, ...
FROM Flüge
WHERE Ab = 'FRA' AND An = 'HNL'
  
```

Flug von FRA nach HNL? (Anzahl der Teilstrecken (hier 3) bekannt)

```

SELECT X.Ab, X.An, Y.An, Z.An.
FROM Flüge X, Flüge Y, Flüge Z
WHERE X.Ab = 'FRA'
AND
AND
AND
AND Z.An = 'HNL'
  
```

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Beispiel: Stücklistenauflösung

Tabelle (Struktur (Otnr, Utnr, Anzahl))

- Aufgabe: Ausgabe aller Endprodukte sowie deren Komponenten
- Max. Schachtelungstiefe sei bekannt (hier: 2)

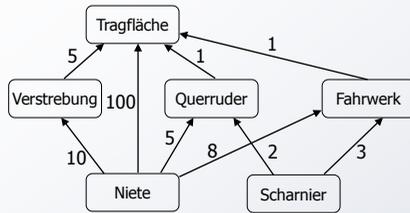
```

exec sql begin declare section;
char T0[10], T1[10], T2[10]; int Anz;
exec sql end declare section;
exec sql declare C0 cursor for
select distinct Otnr from Struktur S1
where not exists (select * from Struktur S2
where S2.Utnr = S1.Otnr);
exec sql declare C1 cursor for
select Utnr, Anzahl from Struktur
where Otnr = :T0;
exec sql declare C2 cursor for
select Utnr, Anzahl from Struktur
where Otnr = :T1;
exec sql open C0;
while (1) {
exec sql fetch C0 into :T0;
if (sqlcode == notfound) break;
printf (" %s\n", T0);
exec sql open C1;
while (2) {
exec sql fetch C1 into :T1, :Anz;
if (sqlcode == notfound) break;
printf (" %s: %d\n", T1, Anz);
exec sql open (C2);
while (3) {
exec sql fetch C2 INTO :T2, :Anz;
if (sqlcode == notfound) break;
printf (" %s: %d\n", T2, Anz); }
exec sql close (C2); } /* end while (2) */
exec sql close C1; } /* end while (1) */
exec sql close (C0);
  
```

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Beispiel: Stücklistenauflösung (2)

Gozinto-Graph

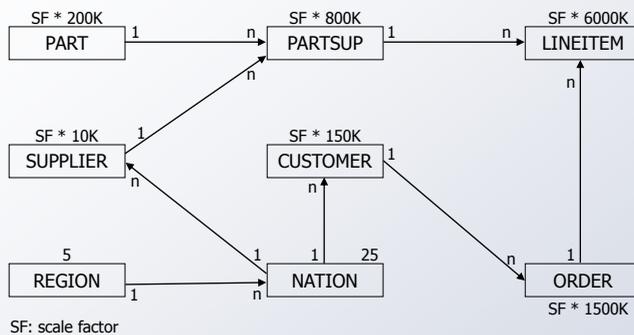


Struktur	(Otrn,	Utrn,	Anzahl)
T	T	V	5
T	T	N	100
T	T	Q	1
T	T	F	1
V	V	N	10
Q	Q	N	5
Q	Q	S	2
F	F	N	8
F	F	S	3

Strukturierte Ausgabe aller Teile von Endprodukten

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Beispiel einer komplexen SQL-Anfrage



Anfrage Q2 beim TPC-H: Minimum Cost Supplier Query

The query finds, in a given region, for each part of a certain type and size, the supplier who can supply it at minimum cost. If several suppliers in that region offer the desired part type and size at the same (minimum) cost, the query lists the parts from suppliers with the 100 highest account balances. For each supplier, the query lists the supplier's account balance, name and nation, the part's number and manufacturer, the supplier's address, phone number and comment information.

Beispiel einer komplexen SQL-Anfrage (2)

■ SQL-Umsetzung von Q2

```

SELECT      S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS,
              S_PHONE, S_COMMENT
FROM        PART, SUPPLIER, PARTSUPP, NATION, REGION
WHERE       P_PARTKEY = PS_PARTKEY
              AND S_SUPPKEY = PS_SUPPKEY
              AND P_SIZE = [size]
              AND P_TYPE LIKE '%[type]%'
              AND S_NATIONKEY = N_NATIONKEY
              AND N_REGIONKEY = R_REGIONKEY
              AND R_NAME = '[region]'
              AND PS_SUPPLYCOST =
              (SELECT MIN(PS_SUPPLYCOST)
               FROM   PARTSUPP, SUPPLIER, NATION, REGION
               WHERE  P_PARTKEY = PS_PARTKEY
                   AND S_SUPPKEY = PS_SUPPKEY
                   AND S_NATIONKEY = N_NATIONKEY
                   AND N_REGIONKEY = R_REGIONKEY
                   AND R_NAME = '[region]')
ORDER BY   S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY;
    
```

4-15

Zentrale Aspekte der Anfrageverarbeitung

■ Deskriptive, mengenorientierte DB-Anweisungen

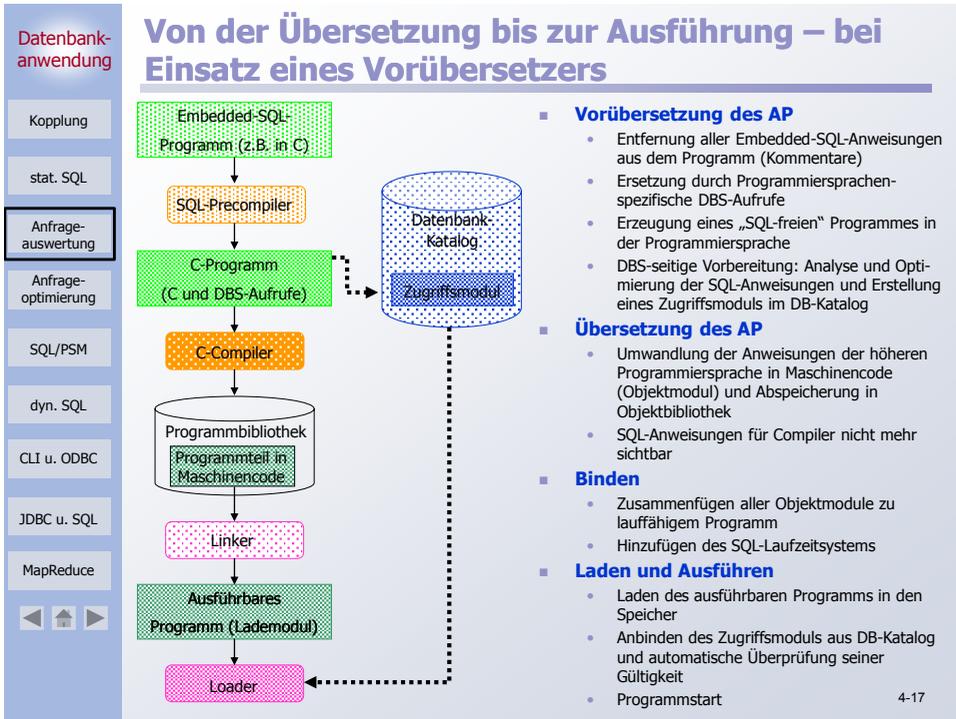
- **Was**-Anweisungen sind in zeitoptimale Folgen interner DBMS-Operationen umzusetzen
- Bei navigierenden DB-Sprachen bestimmt der Programmierer, **wie** eine Ergebnismenge (abhängig von existierenden Zugriffspfaden) satzweise aufzusuchen und auszuwerten ist
- **Jetzt**: Anfrageauswertung/-optimierung des DBMS ist im Wesentlichen für die effiziente Abarbeitung verantwortlich

■ Wirtssprachen-Einbettung und Übersetzung

- **Direkte Einbettung**
 - keine syntaktische Unterscheidung zwischen Programm- und DB-Anweisungen
 - DB-Anweisung wird als Zeichenkette A ins AP integriert, z. B. **exec sql** open C1
 - **verlangt Maßnahmen bei der AP-Übersetzung**, typischerweise Einsatz eines Vorübersetzters PC (Precompiler)²
- **Aufruftechnik**
DB-Anweisung wird durch expliziten Funktionsaufruf an das Laufzeitsystem des DBMS übergeben, z. B. CALL DBS ('SELECT SUM(Gehalt) into :Gsum ...')
- Es sind prinzipiell keine DBMS-spezifischen Vorkehrungen bei der AP-Übersetzung erforderlich!
- Verschiedene Formen der Standardisierung:
Call-Level-Interface (CLI), JDBC

* Sonst ist einerweiterer Compiler C der Wirtssprache bereitzustellen, der sowohl Anweisungen der Wirtssprache als auch der DB-Sprache behandeln kann.

4-16



Übersetzung, Optimierung und Codegenerierung von DB-Anweisungen

■ DB-Anweisung liegt als Zeichenkette vor

- Bei Einsatz eines **Interpreters** fällt der folgende Aufwand zur Laufzeit an
- **Vorübersetzung** verschiebt die „Vorbereitung“ in die Übersetzungszeit
- Erforderliche Verarbeitungsschritte bis zur Auswertung:

1. Lexikalische und syntaktische Analyse

- Erstellung eines Anfragegraphs (AG) als Bezugsstruktur für die nachfolgenden Übersetzungsschritte
- Überprüfung auf korrekte Syntax (Parsing)

2. Semantische Analyse

- Feststellung der Existenz und Gültigkeit der referenzierten Tabellen, Sichten und Attribute
- Einsetzen der Sichtdefinitionen in den AG
- Ersetzen der externen durch interne Namen (Namensauflösung)
- Konversion vom externen Format in interne Darstellung

3. Zugriffs- und Integritätskontrolle

- Sollen aus Leistungsgründen, soweit möglich, schon zur Übersetzungszeit erfolgen
- Zugriffskontrolle erfordert bei Wertabhängigkeit Generierung von Laufzeitaktionen
 - Durchführung einfacher Integritätskontrollen (Kontrolle von Formaten und Konversion von Datentypen)
 - Generierung von Laufzeitaktionen für komplexere Kontrollen

4-19

Übersetzung, Optimierung und Codegenerierung von DB-Anweisungen (2)

4. Standardisierung und Vereinfachung

- dienen der effektiveren Übersetzung und frühzeitigen Fehlererkennung
- Überführung des AG in eine Normalform
 - Elimination von Redundanzen

5. Restrukturierung und Transformation

- Restrukturierung zielt auf globale Verbesserung des AG ab; bei der Transformation werden ausführbare Operationen eingesetzt
 - Anwendung von heuristischen Regeln (**algebraische Optimierung**) zur Restrukturierung des AG
 - Transformation führt Ersetzung und ggf. Zusammenfassen der logischen Operatoren durch Planoperatoren durch (**nicht algebraische Optimierung**): Meist sind mehrere Planoperatoren als Implementierung eines logischen Operators verfügbar
 - Bestimmung alternativer Zugriffspläne (**nicht-algebraische Optimierung**): Meist sind viele Ausführungsreihenfolgen oder Zugriffspfade auswählbar
 - Bewertung der Kosten und Auswahl des günstigsten Ausführungsplanes
- ⇒ Schritte 4 + 5 werden als **Anfrageoptimierung** zusammengefasst

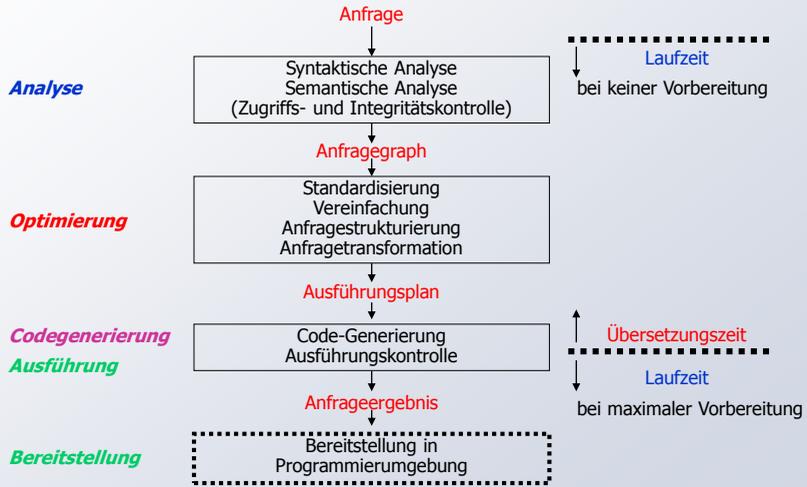
6. Code-Generierung

- Generierung eines zugeschnittenen Programms für die vorgegebene (SQL-) Anweisung
- Erzeugung eines ausführbaren Zugriffsmoduls
- Verwaltung der Zugriffsmodule in einer DBMS-Bibliothek

4-20

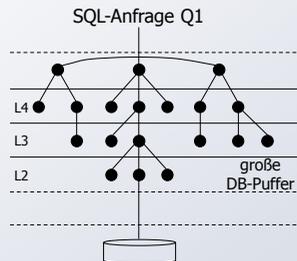
Anfrageverarbeitung - Kostenaspekte

- **Wie teilen sich die Kosten der Transaktionsverarbeitung auf?**
DB-System : Kommunikationssystem : Anwendung
- **Wann fallen Kosten für die Anfrageverarbeitung an?**



Anfrageverarbeitung – maximale Vorbereitung

- **SQL-Anfragebeispiel**
Q1: Select B.Titel, B.E-Jahr, A.Name
FROM Bücher B, Autoren A
Where B.AutorID = A.AutorID And A.Name = „S*“ And B.Fach = „DBS“
- **Zur Laufzeit ersetzt Zugriffsmodul Schicht L5**



- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Anfrageverarbeitung – maximale Vorbereitung (2)

Zugriffsmodul für Q1 (erstellt zum Übersetzungszeitpunkt)

```

Open Scan (IndexB(Fach), Fach='DBS', Fach>'DBS')           /*SCB1*/
Sort Access (SCB1) ASC AutorID Into T1 (AutorID, Titel, ...)
Close Scan (SCB1)

Open Scan (IndexA(Name), Name>='S', Name>'S')             /*SCB2*/
Sort Access (SCB2) ASC AutorID Into T2 (AutorID, Name)
Close Scan (SCB2)

Open Scan (T1, BOF, EOF)                                     /*SCB3*/
Open Scan (T2, BOF, EOF)                                     /*SCB4*/
While Not Finished
Do
    Fetch Tuple (SCB3, Next, None)
    Fetch Tuple (SCB4, Next, None)
    . . .
End
    
```

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Auswertung von DB-Anweisungen (3)

Welche Auswertungstechnik soll gewählt werden?

Spektrum von Verfahren mit folgenden Eckpunkten:

- **Maximale Vorbereitung** (siehe Beispiel)
 - Für die DB-Anweisungen von AP wird ein zugeschnittenes Programm (Zugriffsmodul) zur Übersetzungszeit (ÜZ) erzeugt
- **Keine Vorbereitung**
 - Technik ist typisch für Call-Schnittstellen (dynamisches SQL)
 - Allgemeines Programm (Interpreter) akzeptiert DB-Anweisungen als Eingabe und erzeugt durch Aufrufe des Zugriffssystems das Ergebnis

Kosten der Auswertung

- Vorbereitung (ÜZ) + Ausführung (LZ)
- Vorbereitung erfolgt durch „Übersetzung“ (Ü)
- Ausführung
 - Laden und Abwicklung (A) des Zugriffsmoduls
 - Sonst: Interpretation (I) der vorliegenden Struktur

Aufteilung der Kosten

	Übersetzungszeit			Laufzeit
Vorbereitung	Analyse	Optimierung	Code-Gen.	Ausführung
Zugriffsmodul				
Ausführungsplan				
Anfragegraph				
keine				

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung**
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Anfrageoptimierung*

- **Von der Anfrage (Was?) zur Auswertung (Wie?)**
 ⇒ **Ziel: kostengünstiger Auswertungsweg**
- **Arten von Optimierern**
 - **regelbasiert** (siehe Optimierung von Operationen der Relationalalgebra)
 - **kostenbasiert** (Kostenmodell erforderlich, höhere Genauigkeit)
- **Einsatz einer großen Anzahl von Techniken und Strategien**
 - logische Transformation von Anfragen
 - Auswahl von Zugriffspfaden
 - Berücksichtigung von Clusterbildung
 - optimierte Speicherung von Daten auf Externspeichern
- **Schlüsselproblem**
 - genaue Optimierung ist im allgemeinen „nicht berechenbar“
 - Fehlen von genauer statistischer Information
 - breiter Einsatz von Heuristiken (Daumenregeln)
- **Optimierungsziel**
 - „entweder Maximierung des Outputs bei gegebenen Ressourcen
 oder Minimierung der Ressourcennutzung für gegebenen Output“
 - Durchschnittsmaximierung?
 - Antwortzeitminimierung für gegebene Anfragesprache, Mix von Anfragen verschiedenen Typs und gegebener Systemumgebung!

* Jarke, M., Koch, J.: Query Optimization in Database Systems, in: ACM Computing Surveys 16:2, 1984, pp. 111-152

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung**
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Anfragegraph und Ausführungsplan

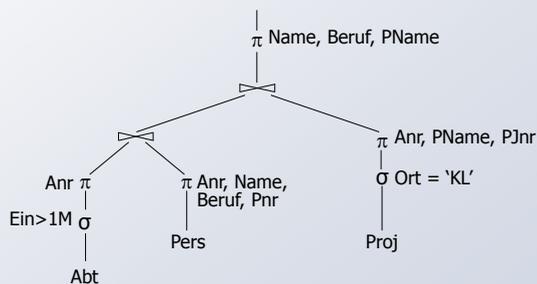
■ Problemendarstellung – Beispiel

```

SQL
SELECT P.Name, P.Beruf, J.PName
FROM Pers P, Abt A, Proj J
WHERE A.Ein > 1000000 AND J.Ort = 'KL'
      AND A.Anr = P.Anr AND A.Anr = J.Anr
    
```

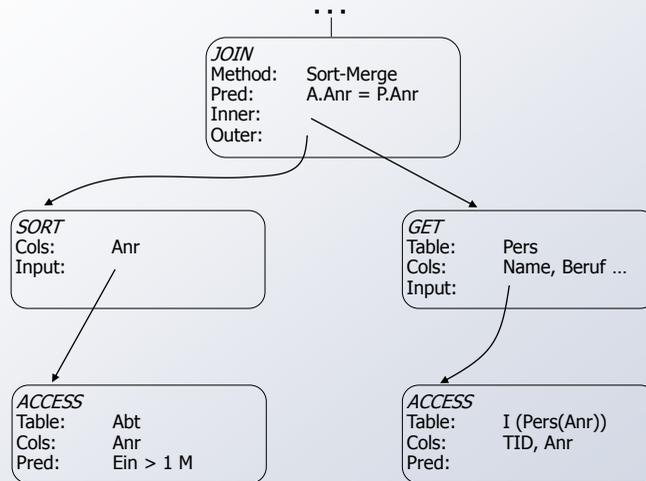
Zugehöriger Anfragegraph

(nach Restrukturierung oder Regelbasierter Optimierung)



Anfragegraph und Ausführungsplan (2)

- **Ausschnitt aus einem möglichen Ausführungsplan**



Bewertung von Ausführungsplänen – Grundsätzliche Probleme

- **Kostenbasierte Anfrageoptimierung beruht i. Allg. auf zwei „fatalen“ Annahmen**

1. Alle Datenelemente und alle Attributwerte sind gleichverteilt
 2. Die Werte verschiedener Attribute sind stochastisch unabhängig
- ⇒ Beide Annahmen sind (im allgemeinen Fall) falsch!

- **Beispiel**

(GEHALT ≥ '100K') AND (ALTER BETWEEN 21 AND 30)

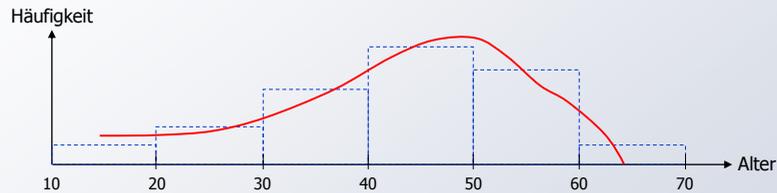
Bereiche: 10K – 1M 16 – 65

1. Lineare Interpolation:
2. Multiplikation von Wahrscheinlichkeiten

Bewertung von Ausführungsplänen – Grundsätzliche Probleme (2)

■ Lösung

- Verbesserung der Statistiken/Heuristiken (Histogramme)



- Berechnung/Bewertung von noch mehr Ausführungsplänen?

Obwohl die Kostenabschätzungen meist falsch sind ...

Berechnung der Zugriffskosten

■ Optimiereraufgabe

- erstellt Kostenvoranschlag für jeden „aussichtsreichen“ Ausführungsplan
- Einsatz einer gewichteten Kostenformel:

$$C = \# \text{physischer Seitenzugriffe} + W * (\# \text{Aufrufe des Zugriffssystems})$$
- gewichtetes Maß für E/A- und CPU-Auslastung
- W ist das Verhältnis des Aufwandes von ZS-Aufruf zu Seitenzugriff

■ Ziel der Gewichtung:

Minimierung der Kosten in Abhängigkeit des Systemzustandes

- System ist "I/O-bound": ⇒ sehr kleiner W-Wert

$$W_{I/O} = \frac{\# \text{ Instr. pro ZS - Aufruf}}{\# \text{ Instr. pro E/A} + \text{Zugriffszeit} \cdot \text{MIPS - Rate}}$$

$$\text{Bsp. } W_{I/O} = \frac{1000 \text{ I.}}{2500 \text{ I.} + 6 \text{ msec} \cdot 10^9 \text{ I./sec}} = 0,000016$$

- System ist „CPU-bound“: ⇒ relativ großer W-Wert

$$W_{CPU} = \frac{\# \text{ Instr. pro ZS - Aufruf}}{\# \text{ Instr. pro E/A}}$$

$$\text{Bsp. } W_{CPU} = \frac{1000}{2500} = 0,4$$

Berechnung der Zugriffskosten (2)

■ Fiktives Beispiel: Entscheidung zwischen zwei Ausführungsplänen

- $C1 = 200 + W * 1000$
- $C2 = 100 + W * 2000$
- System ist "I/O-bound":
- System ist "CPU-bound":

4-31

Kostenmodell – statistische Werte

■ Statistische Größen für Segmente

M_S Anzahl der Datenseiten des Segmentes S

L_S Anzahl der leeren Seiten in S

■ Statistische Größen für Tabellen

N_R Anzahl der Tupel der Tabelle R (Card(R))

$T_{R,S}$ Anzahl der Seiten in S mit Tupel von R

C_R Clusterfaktor (Anzahl Tupel pro Seite)

■ Statistische Größen pro Index I auf Attributen A einer Tabelle R:

j_I Anzahl der Attributwerte / Schlüsselwerte im Index
(=Card($\pi_A(R)$))

B_I Anzahl der Blattseiten (B*-Baum)

...

⇒ **Statistiken müssen im DB-Katalog gewartet werden**

■ Aktualisierung bei jeder Änderung sehr aufwendig

- zusätzliche Schreib- und Log-Operationen
- DB-Katalog wird zum Sperr-Engpass

■ Alternative

- Initialisierung der statistischen Werte zum Lade- oder Generierungszeitpunkt von Tabellen und Indexstrukturen
- periodische Neubestimmung der Statistiken durch eigenes Kommando/ Dienstprogramm (DB2: RUNSTATS)

4-32

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce
- ◀ ▶

Kostenmodell – Berechnungsgrundlagen

Mit Hilfe der statistischen Werte kann der Anfrageoptimierer jedem Verbundterm im Qualifikationsprädikat einen Selektivitätsfaktor ($0 \leq SF \leq 1$) zuordnen (erwarteter Anteil an Tupel, die das Prädikat erfüllen): $Card(\sigma_p(R)) = SF(p) \cdot Card(R)$

■ **Selektivitätsfaktor SF bei:**

$A_i = a_i$	$SF = \begin{cases} 1/j_i & \text{wenn Index auf } A_i \\ 1/10 & \text{sonst} \end{cases}$
$A_i = A_k$	$SF = \begin{cases} 1 / \text{Max}(j_i, j_k) & \text{wenn Index auf } A_i, A_k \\ 1/j_i & \text{wenn Index auf } A_i \\ 1/j_k & \text{wenn Index auf } A_k \\ 1/10 & \text{sonst} \end{cases}$
$A_i \geq a_i$ (oder $A_i > a_i$)	$SF = \begin{cases} (a_{\max} - a_i) / (a_{\max} - a_{\min}) & \text{wenn Index auf } A_i \\ & \text{und Wert interpolierbar} \\ 1/3 & \text{sonst} \end{cases}$
$A_i \text{ BETWEEN } a_i \text{ AND } a_k$	$SF = \begin{cases} (a_k - a_i) / (a_{\max} - a_{\min}) & \text{wenn Index auf } A_i \\ & \text{und Wert interpolierbar} \\ 1/4 & \text{sonst} \end{cases}$
$A_i \text{ IN } (a_1, a_2, \dots, a_r)$	$SF = \begin{cases} r/j_i & \text{wenn Index auf } A_i \\ & \text{und } SF < 0.5 \\ 1/2 & \text{sonst} \end{cases}$

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce
- ◀ ▶

Kostenmodell – Berechnungsgrundlagen (2)

■ **Berechnung von Ausdrücken**

- $SF(p(A) \wedge p(B)) = SF(p(A)) \cdot SF(p(B))$
- $SF(p(A) \vee p(B)) = SF(p(A)) + SF(p(B)) - SF(p(A)) \cdot SF(p(B))$
- $SF(\neg p(A)) = 1 - SF(p(A))$

■ **Join-Selektivitätsfaktor (JSF)**

- $Card(R \bowtie S) = JSF * Card(R) * Card(S)$
- bei (N:1)-Verbunden (verlustfrei): $Card(R \bowtie S) = \text{Max}(Card(R), Card(S))$

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung**
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Beispiel: Einfache Anfrage

SQL-Anfrage

```
SELECT NAME, GEHALT
FROM PERS
WHERE BERUF = 'PROGRAMMIERER'
AND GEHALT BETWEEN 100.000 AND 200.000
```

Vorhandene Zugriffspfade

- Tabelle PERS als Heap-Struktur (ungeordnete Speicherung im Segment)
- $I_{PERS}(BERUF)$ mit 100 verschiedenen Berufen
- $I_{PERS}(GEHALT)$ mit Gehaltsspanne von 10 K bis 1 M
- LINK von FAEHIGKEIT nach PERS (hierarchischer Zugriffspfad)

Zugriffsmöglichkeiten

- Scans über die vorhandenen Zugriffspfade oder Speicherungsstrukturen
- **Optionen:** Start- (OPEN), Stopp- (CLOSE), Suchbedingung (FETCH NEXT)

Statistische Kennwerte

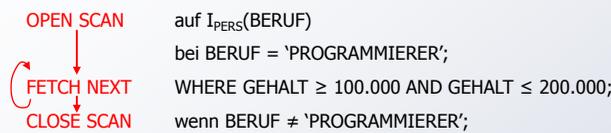
- Der Optimizer findet folgende Parameter im DB-Katalog:
- N = # der Tupel in Tabelle PERS
- m = Anzahl der Seiten von Segment S
- C = durchschnittliche Anzahl von PERS-Tupeln pro Seite
- j_i = Index-Kardinalität (Anzahl der Attributwerte für A_i)
- h = Höhe der B*-Bäume + Information über Clusterbildung

Annahmen

- Jeder 10. Programmierer hat ein Gehalt zwischen 100 K und 200 K
- Jeder 2. Angestellte mit Gehalt zwischen 100 K und 200 K ist Programmierer

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung**
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

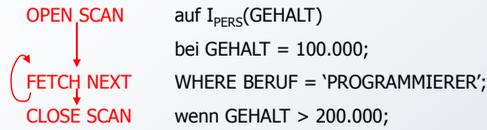
Methode 1: Scan über $I_{PERS}(BERUF)$



Kosten

- Clusterbildung auf $I_{PERS}(BERUF)$
$$K \approx h + \frac{N}{C \cdot j_{BERUF}} + W \cdot \frac{N}{j_{BERUF} \cdot 10}$$
- keine Clusterbildung
$$K \approx h + \frac{N}{j_{BERUF}} + W \cdot \frac{N}{j_{BERUF} \cdot 10}$$

Methode 2: Scan über $I_{PERS}(GEHALT)$



Kosten

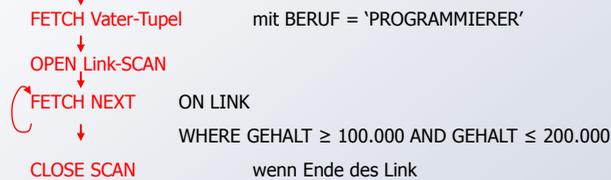
- Clusterbildung auf $I_{PERS}(GEHALT)$ $K \approx h + \frac{N}{10 \cdot C} + W \cdot \frac{N}{10 \cdot 2}$
- keine Clusterbildung $K \approx h + \frac{N}{10} + W \cdot \frac{N}{10 \cdot 2}$

Methode 3: Benutze einen hierarchischen Zugriffspfad (LINK) von einer anderen Tabelle



Annahme

Schneller Zugriff auf Tabelle FAEHIGKEIT als Einstieg in LINK möglich, z.B. über $I_{FAEHIGKEIT}(BERUF)$



Kosten

- Clusterbildung auf Link $K \approx h + \frac{N}{C \cdot j_{BERUF}} + W \cdot \frac{N}{j_{BERUF} \cdot 10}$
- keine Clusterbildung $K \approx h + \frac{N}{j_{BERUF}} + W \cdot \frac{N}{j_{BERUF} \cdot 10}$

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung**
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Übersetzung vs. Interpretation

Was heißt „Binden“?

AP: SELECT Pnr, Name, Gehalt
FROM Pers
WHERE Beruf = 'Programmierer'

DB-Katalog: SYSTAB: Tabellenbeschreibungen: Pers, . . .
SYSATTR: Attributbeschreibungen: Pnr, Name, Gehalt, . . .
SYSINDEX: IPers(Beruf), . . .
SYSAUTH: Nutzungsrechte
SYSINT/RULES: Integritätsbedingungen, Zusicherungen, . . .

Zeitpunkt des Bindens



Übersetzungskosten:

- unerheblich für Antwortzeit

Zugriffe (zur LZ):

- effizient
- datenabhängig!

{ **Ausgleich**
gesucht! }

Interpretation:

- erheblich für Antwortzeit (AZ)

Zugriffe (zur LZ):

- teuer
- datenunabhängig!

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung**
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Übersetzung vs. Interpretation (2)

Bindezeitpunkt

macht die für die Abwicklung einer DB-Anweisung erforderlichen Operationen von DB-Schema abhängig!

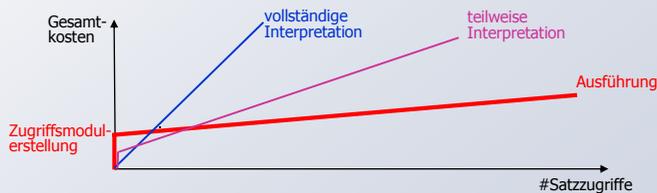
Maximale Vorbereitung einer DB-Anweisung

- Vorübersetzung und frühes Binden sind eine gute Idee
- aufwendige Optimierung möglich, aber auch maximale Auswirkungen von Schemaänderungen
- Änderungen des DB-Zustandes nach der Übersetzung werden nicht berücksichtigt (neue Zugriffspfade, geänderte Statistiken etc.)

⇒ Invalidierung des Zugriffsmoduls und erneute Erstellung

Interpretation einer DB-Anweisung

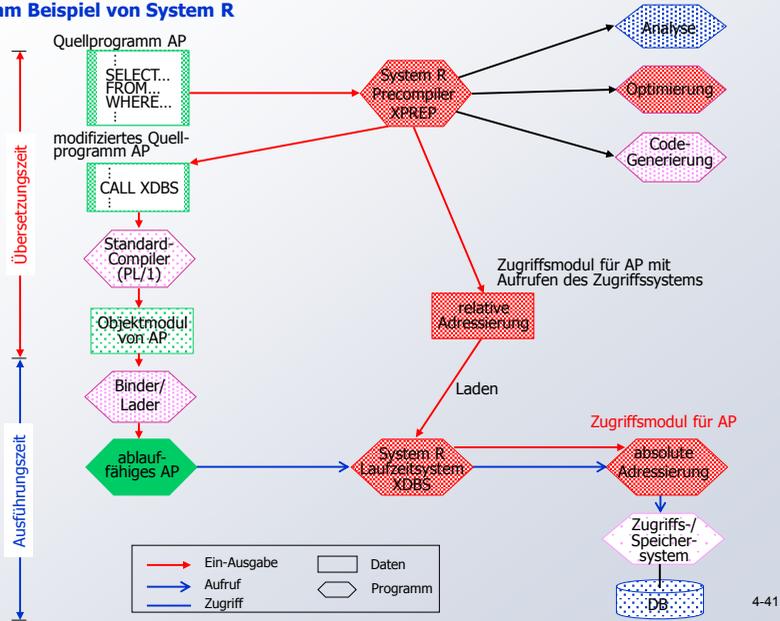
- Interpreter wertet Anweisung (als Zeichenfolge) zur Laufzeit aus
- Aktueller DB-Zustand wird automatisch berücksichtigt
- sehr hohe Ausführungskosten bei Programmschleifen sowie durch häufige Katalogzugriffe
- interessant vor allem für Ad-hoc-Anfragen bzw. dynamisches SQL



- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Vorbereitung und Ausführung von DB-Anweisungen

am Beispiel von System R



- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

SQL/PSM

■ PSM (Persistent Stored Modules)

- zielt auf **Leistungsverbesserung** vor allem in Client/Server-Umgebung ab
 - Ausführung mehrerer SQL-Anweisungen durch ein EXEC SQL
 - Entwerfen von Routinen mit mehreren SQL-Anweisungen
- erhöht die **Verarbeitungsmächtigkeit** des DBS
 - Prozedurale Erweiterungsmöglichkeiten (der DBS-Funktionalität aus Sicht der Anwendung)
 - Einführung neuer Kontrollstrukturen
- erlaubt **reine SQL-Implementierungen** von komplexen Funktionen
 - Sicherheitsaspekte
 - Leistungsaspekte
- ermöglicht **SQL-implementierte Klassenbibliotheken** (SQL-only)

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM**
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

SQL/PSM (2)

■ Beispiel

- ins AWP eingebettet


```

      . . .
      EXEC SQL INSERT INTO Pers VALUES (...);
      EXEC SQL INSERT INTO Abt VALUES (...);
      . . .
      
```
- Erzeugen einer SQL-Prozedur
 - CREATE PROCEDURE proc1 ()


```

          {
              BEGIN
              INSERT INTO Pers VALUES (...);
              INSERT INTO Abt VALUES (...);
              END;
          }
          
```
- Aufruf aus AWP


```

      . . .
      EXEC SQL CALL proc1 ( );
      . . .
      
```

■ Vorteile

- **Vorübersetzte Ausführungspläne** werden gespeichert, sind wiederverwendbar
- **Anzahl der Zugriffe** des Anwendungsprogramms auf die DB wird reduziert
- Prozeduren sind als **gemeinsamer Code** für verschiedene Anwendungsprogramme nutzbar
- Es wird ein **höherer Isolationsgrad** der Anwendung von der DB erreicht

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM**
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

SQL/PSM – Prozedurale Spracherweiterungen

- | | |
|--|--|
| <ul style="list-style-type: none"> • Compound statement • SQL variable declaration • If statement • Case statement • Loop statement • While statement • Repeat statement • For statement • Leave statement • Return statement • Call statement • Assignment statement • Signal/resignal statement | <ul style="list-style-type: none"> • BEGIN ... END; • DECLARE var CHAR (6); • IF subject (var <> 'urgent') THEN ... ELSE ...; • CASE subject (var) WHEN 'SQL' THEN ... WHEN ...; • LOOP <SQL statement list> END LOOP; • WHILE i<100 DO ... END WHILE; • REPEAT ... UNTIL i<100 END REPEAT; • FOR result AS ... DO ... END FOR; • LEAVE ...; • RETURN 'urgent'; • CALL procedure_x (1,3,5); • SET x = 'abc'; • SIGNAL divison_by_zero |
|--|--|

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL**
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Dynamisches SQL

■ Festlegen/Übergabe von SQL-Anweisungen zur Laufzeit

- Benutzer stellt Ad-hoc-Anfrage
- AP berechnet dynamisch SQL-Anweisung
- SQL-Anweisung ist aktueller Parameter von Funktionsaufrufen an das DBMS
⇒ **Dynamisches SQL** erlaubt Behandlung solcher Fälle

■ Mehrere Sprachansätze

- Eingebettetes dynamisches SQL
- Call-Level-Interface (CLI): kann ODBC-Schnittstelle¹ implementieren
- Java Database Connectivity² (JDBC) ist eine dynamische SQL-Schnittstelle zur Verwendung mit Java
 - JDBC ist gut in Java integriert und ermöglicht einen Zugriff auf relationale Datenbanken in einem objektorientierten Programmierstil
 - JDBC ermöglicht das Schreiben von Java-Applets, die von einem Web-Browser auf eine DB zugreifen können
 ⇒ **Funktionalität ähnlich, jedoch nicht identisch**

■ Gleiche Anforderungen bei allen Sprachansätzen (LZ)

- Zugriff auf Metadaten
- Übergabe und Abwicklung dynamisch berechneter SQL-Anweisungen
- Optionale Trennung von Vorbereitung und Ausführung
 - einmalige Vorbereitung mit Platzhalter (?) für Parameter
 - n-malige Ausführung
- Explizite Bindung von Platzhaltern (?) an Wirtsvariable
 - Variable sind zur ÜZ nicht bekannt!
 - Variablenwert wird zur Ausführungszeit vom Parameter übernommen

1. Die Schnittstelle Open Database Connectivity (ODBC) wird von Microsoft definiert.
 2. 'de facto'-Standard für den Zugriff auf relationale Daten von Java-Programmen aus: Spezifikation der JDBC-Schnittstelle 4-45 unter <http://java.sun.com/products/jdbc>

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL**
- CLI u. ODBC
- JDBC u. SQL
- MapReduce

Eingebettetes dynamisches SQL (EDSQL)

■ Wann wird diese Schnittstelle gewählt?

- Sie unterstützt auch andere Wirtssprachen als C
- Sie ist im Stil statischem SQL ähnlicher; sie wird oft von Anwendungen gewählt, die dynamische und statische SQL-Anweisungen mischen
- Programme mit EDSQL sind kompakter und besser lesbar als solche mit CLI oder JDBC

■ EDSQL

- besteht im wesentlichen aus 4 Anweisungen:
 - DESCRIBE
 - PREPARE
 - EXECUTE
 - EXECUTE IMMEDIATE

■ SQL-Anweisungen werden vom Compiler wie Zeichenketten behandelt

- Deklaration DECLARE STATEMENT
- Anweisungen enthalten Platzhalter für Parameter (?) statt Programmvariablen

Eingebettetes dynamisches SQL (2)

Trennung von Vorbereitung und Ausführung

```
exec sql begin declare section;
char Anweisung [256], X[3];
exec sql end declare section;
exec sql declare SQLAnw statement;
```

```
/* Zeichenkette kann zugewiesen bzw. eingelesen werden */
Anweisung = 'DELETE FROM Pers WHERE Anr = ?';
```

```
/* Prepare-and-Execute optimiert die mehrfache Verwendung
einer dynamisch erzeugten SQL-Anweisung */
```

```
exec sql prepare SQLAnw from :Anweisung;
exec sql execute SQLAnw using 'K51';
scanf ("%s ", X);
exec sql execute SQLAnw using :X;
```

Bei einmaliger Ausführung EXECUTE IMMEDIATE ausreichend

```
scanf ("%s ", Anweisung);
exec sql execute immediate :Anweisung;
```

Cursor-Verwendung

- SELECT-Anweisung nicht Teil von DECLARE CURSOR, sondern von PREPARE-Anweisung
- OPEN-Anweisung (und FETCH) anstatt EXECUTE

```
exec sql declare SQLAnw statement;
exec sql prepare SQLAnw from
"SELECT Name FROM Pers WHERE Anr=?";
exec sql declare C1 cursor for SQLAnw;
exec sql open C1 using 'K51';
...
```

4-47

Eingebettetes dynamisches SQL (3)

Dynamische Parameterbindung

```
Anweisung = 'INSERT INTO Pers VALUES (?, ?, ...)';
exec sql prepare SQLAnw from :Anweisung;
vname = 'Ted';
nname = 'Codd';
exec sql execute SQLAnw using :vname, :nname, ...;
```

Zugriff auf Beschreibungsinformation wichtig

- wenn Anzahl und Typ der dynamischen Parameter nicht bekannt ist
- Deskriptorbereich ist eine gekapselte Datenstruktur, die durch das DBMS verwaltet wird (kein SQLDA vorhanden)

```
Anweisung = 'INSERT INTO Pers VALUES (?, ?, ...)';
exec sql prepare SQLAnw from :Anweisung;
exec sql allocate descriptor 'Eingabeparameter';
exec sql describe input SQLAnw into sql descriptor 'Eingabeparameter';
exec sql get descriptor 'Eingabeparameter' :n = count;
```

```
for (i = 1; i < n; i++)
{
exec sql get descriptor 'Eingabeparameter' value :i
:attrtyp = type, :attrlänge = length, :attrname = name;
...
exec sql set descriptor 'Eingabeparameter' value :i
data = :d, indicator = :ind;
}
```

```
exec sql execute SQLAnw
using sql descriptor 'Eingabeparameter';
```

4-48

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC**
- JDBC u. SQL
- MapReduce

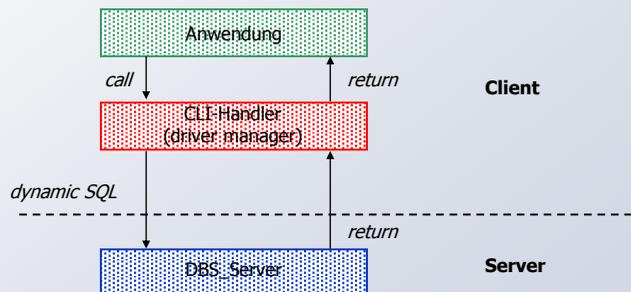
Call-Level-Interface

■ Spezielle Form von dynamischem SQL

- Schnittstelle ist als Sammlung von Prozeduren/Funktionen realisiert
- Direkte Aufrufe der Routinen einer standardisierten Bibliothek
- Keine Vorübersetzung (Behandlung der DB-Anweisungen) von Anwendungen
 - Vorbereitung der DB-Anweisung geschieht erst beim Aufruf zur LZ
 - Anwendungen brauchen nicht im Quell-Code bereitgestellt werden
 - Wichtig zur Realisierung von kommerzieller AW-Software bzw. Tools

⇒ Schnittstelle wird sehr häufig in der Praxis eingesetzt

■ Einsatz typischerweise in Client/Server-Umgebung



4-49

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC**
- JDBC u. SQL
- MapReduce

Call-Level-Interface (2)

■ Vorteile von CLI

- Schreiben portabler Anwendungen
 - keinerlei Referenzen auf systemspezifische Kontrollblöcke wie SQLCA/SQLDA
 - kann die ODBC-Schnittstelle implementieren
- Systemunabhängigkeit
 - Funktionsaufrufe zum standardisierten Zugriff auf den DB-Katalog
- Mehrfache Verbindungen zur selben DB
 - unabhängige Freigabe von Transaktionen in jeder Verbindung
 - nützlich für AW mit GUIs (graphical user interfaces), die mehrere Fenster benutzen
- Optimierung des Zugriffs vom/zum Server
 - Holen von mehreren Zeilen pro Zugriff
 - Lokale Bereitstellung einzelner Zeilen (Fetch)
 - Verschicken von zusammengesetzten SQL-Anweisungen
 - Client-Programme können Stored Procedures (PSM) aufrufen

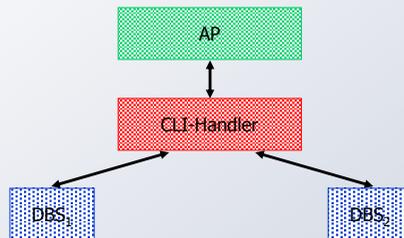
4-50

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC**
- JDBC u. SQL
- MapReduce

Call-Level-Interface (3)

Wie kooperieren AP und DBMS?

- maximale gegenseitige Kapselung
- Zusammenspiel AP/CLI und DBMS ist nicht durch Übersetzungsphase vorbereitet
 - keine DECLARE SECTION
 - keine Übergabebereiche
- Wahl des DBMS zur Laufzeit
- vielfältige LZ-Abstimmungen erforderlich



Konzept der Handle-Variablen wesentlich

- „Handle“ (internes Kennzeichen) ist letztlich eine Programmvariable, die Informationen repräsentiert, die für ein AP durch die CLI-Implementierung verwaltet wird
- gestattet Austausch von Verarbeitungsinformationen

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC**
- JDBC u. SQL
- MapReduce

Call-Level-Interface (4)

4 Arten von Handles

- **Umgebungs-kennung** repräsentiert den globalen Zustand der Applikation
- **Verbindungs-kennung**
 - separate Kennung: n Verbindungen zu einem oder mehreren DBMS
 - Freigabe/Rücksetzen von Transaktionen
 - Steuerung von Transaktionseigenschaften (Isolationsgrad)
- **Anweisungs-kennung**
 - mehrfache Definition, auch mehrfache Nutzung
 - Ausführungszustand einer SQL-Anweisung; sie fasst Informationen zusammen, die bei statischem SQL in SQLCA, SQLDA und Positionsanzeigern (Cursor) stehen
- **Deskriptorkennung** enthält Informationen, wie Daten einer SQL-Anweisung zwischen DBMS und CLI-Programm ausgetauscht werden

CLI-Standardisierung in SQL3:

- ISO-Standard wurde 1996 verabschiedet
- starke Anlehnung an ODBC bzw. X/Open CLI
- Standard-CLI umfasst über 40 Routinen: Verbindungskontrolle, Ressourcen-Allokation, Ausführung von SQL-Befehlen, Zugriff auf Diagnoseinformation, Transaktionsklammerung, Informationsanforderung zur Implementierung

Standard-CLI: Beispiel

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC**
- JDBC u. SQL
- MapReduce

```
#include „sqlcli.h“
#include <string.h>
...
{
SQLCHAR * server;
SQLCHAR * uid;
SQLCHAR * pwd;
HENV henv; // environment handle
HDBC hdbc; // connection handle
HSTMT hstmt; // statement handle
SQLINTEGER id;
SQLCHAR name [51];

/* connect to database */
SQLAllocEnv (&henv);
SQLAllocConnect (henv, &hdbc);
if (SQLConnect (hdbc, server, uid,
pwd, ...) != SQL_SUCCESS)
return (print_err (hdbc, ...));

/* create a table */
SQLAllocStmt (hdbc, &hstmt);
{
SQLCHAR create [] = "CREATE TABLE
NameID (ID integer,
Name varchar (50))";

if (SQLExecDirect (hstmt, create, ...)
!= SQL_SUCCESS
return (print_err (hdbc, hstmt));
}
/* commit transaction */
SQLTransact (henv, hdbc, SQL_COMMIT);

/* insert row */
{ SQLCHAR insert [] = "INSERT INTO
NameID VALUES (?, ?)";
if (SQLPrepare (hstmt, insert, ...) !=
SQL_SUCCESS)
return (print_err (hdbc, hstmt));

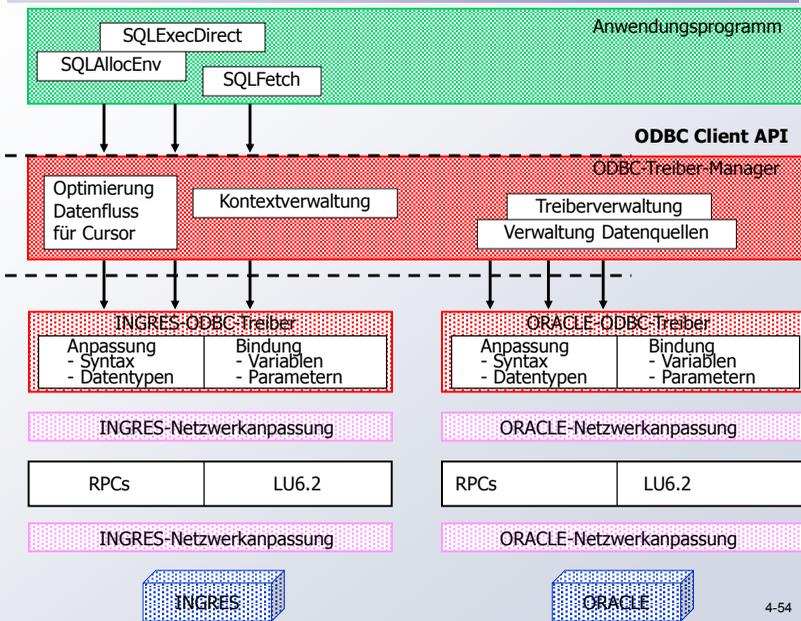
SQLBindParam (hstmt, 1, ..., id, ...);
SQLBindParam (hstmt, 2, ..., name,
...);
id = 500; strcpy (name, "Schmidt");

if (SQLExecute (hstmt) != SQL_SUCCESS)
return (print_err (hdbc, hstmt) ); ;

/* commit transaction */
SQLTransact (henv, hdbc, SQL_COMMIT);
}
}
```

Beispiel Microsoft – ODBC-Architektur -

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC**
- JDBC u. SQL
- MapReduce



- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC**
- JDBC u. SQL
- MapReduce

Überwindung der Heterogenität mit ODBC

- **ODBC (Open Database Connectivity) ist**
 - eine durch die Firma Microsoft definierte und von ihr favorisierte Architektur, die aus funktionaler Sicht Heterogenität (eingermaßen) überwindet,
 - jedoch z.T. erhebliche Leistungseinbußen gegenüber einer DBMS-Hersteller-spezifischen Anbindung verzeichnet.
- **ODBC umfasst u.a.**
 - eine Programmierschnittstelle vom CLI-Typ und
 - eine Definition des unterstützten SQL-Sprachumfangs (im Vergleich zu ISO SQL2).
- **DBMS-Anbieter**
 - implementieren sogenannte ODBC-Treiber (Umsetzung von Funktionen und Daten auf herstellerspezifische Schnittstellen),
 - die gemäß den ebenfalls in der ODBC-Architektur definierten Windows-internen Schnittstellen in die Windows-Betriebssysteme integriert werden können.
- **ODBC**
 - wird von praktisch allen relevanten DBMS-Herstellern unterstützt und
 - stellt einen **herstellerspezifischen De-facto-Standard** dar,
 - der für die **unterschiedlichen Windows-Betriebssysteme** auf der Anwendungsseite Gültigkeit hat.

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC**
- JDBC u. SQL
- MapReduce

Beispiel ODBC

```

RETCODE retcode;                                     /* Return Code */
HENV henv; HDBC hdbc;                                /* Environment und Connection Handle */
HSTMT hstmt;                                         /* Statement Handle */
UCHAR szName[33], szAbtName[33]; long IBonus;
retcode = SQLAllocEnv (&henv);                       /* Anlegen Anwendungskontext */
retcode = SQLAllocConnect (henv, &hdbc);             /* Anlegen Verbindungskontext */
retcode = SQLAllocStmt (hdbc, &hstmt);               /* Anlegen Anweisungskontext */
retcode = SQLConnect (hdbc, „DEMO-DB“, SQL_NTS, „PePe“, SQL_NTS, „GEHEIM“, SQL_NTS);
/* Verbindung aufbauen */
retcode = SQLSetConnect Option (hdbc, SQL_ACCESS_MODE, SQL_MODE_READ_ONLY);
/* Eigenschaften */
retcode = SQLExecDirect (hstmt, „UPDATE Mitarbeiter SET Bonus = 0.2 * Gehalt“, SQL_NTS);
/* Ausführen */
retcode = SQLExecDirect (hstmt, „SELECT M.Name, M.Bonus, A.Abname
FROM Mitarbeiter M, Abteilung A
WHERE A.AbtNr = M.AbtNr“, SQL_NTS);
retcode = SQLBindCol (hstmt, 1, SQL_C_DEFAULT, szName, 33, &cbName);
retcode = SQLBindCol (hstmt, 2, SQL_C_DEFAULT, szAbtName, 33, &cbAbtName);
/* Variablen binden */
retcode = SQLBindCol (hstmt, 3, SQL_C_DEFAULT, szBonus, sizeof(long), &cbBonus);
retcode = SQLFetch (hstmt);                          /* Zeile anfordern */
retcode = SQL Transact (henv, hdbc, SQL_COMMIT);
/* Freigabe der dynamisch angeforderten Kontexte */
retcode = SQLFreeStmt (hstmt); retcode = SQLDisconnect (hdbc);
retcode = SQLFreeConnect (hdbc); retcode = SQLFreeEnv (henv);
    
```

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL**
- MapReduce

DB-Zugriff via JDBC

- **Java Database Connectivity Data Access API (JDBC)***
 - unabhängiges, standardisiertes CLI, basierend auf SQL:1999
 - bietet Schnittstelle für den Zugriff auf (objekt-) relationale DBMS aus Java-Anwendungen
 - besteht aus zwei Teilen
 - Core Package: Standardfunktionalität mit Erweiterungen (Unterstützung von SQL:1999-Datentypen, flexiblere ResultSets, ...)
 - Optional Package: Ergänzende Funktionalität (Connection Pooling, verteilte Transaktionen, ...)

- **Allgemeines Problem**
 Verschiedene DB-bezogene APIs sind aufeinander abzubilden



- **Überbrückung/Anpassung durch Treiber-Konzept**
 - setzen JDBC-Aufrufe in die DBMS-spezifischen Aufrufe um
 - Treiber werden z.B. vom DBMS-Hersteller zur Verfügung gestellt
 - Treiber-Unterstützung kann auf vier verschiedene Arten erfolgen

* Standard: JDBC API 3.0 Specification Final Release <http://java.sun.com/products/jdbc>

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL**
- MapReduce

JDBC – wichtige Funktionalität

- **Laden des Treiber**
 - kann auf verschiedene Weise erfolgen, z.B. durch explizites Laden mit dem Klassenlader:
`Class.forName (DriverClassName)`
- **Aufbau einer Verbindung**
 - Connection-Objekt repräsentiert die Verbindung zum DB-Server
 - Beim Aufbau werden URL der DB, Benutzername und Paßwort als Strings übergeben:
`Connection con = DriverManager.getConnection (url, login, pwd);`
- **Anweisungen**
 - Mit dem Connection-Objekt können u.a. Metadaten der DB erfragt und Statement-Objekte zum Absetzen von SQL-Anweisungen erzeugt werden
 - Statement-Objekt erlaubt das Erzeugen einer SQL-Anweisung zur direkten (einmaligen) Ausführung
`Statement stmt = con.createStatement();`
 - PreparedStatement-Objekt erlaubt das Erzeugen und Vorbereiten von (parametrisierten) SQL-Anweisungen zur wiederholten Ausführung
`PreparedStatement pstmt = con.prepareStatement ("select * from personal where gehalt >= ?");`
 - Ausführung einer Anfrageanweisung speichert ihr Ergebnis in ein spezifiziertes ResultSet-Objekt
`ResultSet res = stmt.executeQuery ("select name from personal");`
- **Schließen von Verbindungen, Statements usw.**
 - `Stmt.close();`
 - `Con.close();`

JDBC - Anweisungen

Anweisungen (Statements)

- Sie werden in einem Schritt vorbereitet und ausgeführt
- Sie entsprechen dem Typ EXECUTE IMMEDIATE im dynamischen SQL
- JDBC-Methode erzeugt jedoch ein Objekt zur Rückgabe von Daten

executeUpdate-Methode

wird zur direkten Ausführung von UPDATE-, INSERT-, DELETE- und DDL-Anweisungen benutzt

```
Statement stat = con.createStatement ();
int n = stat.executeUpdate ("update personal
    set gehalt = gehalt * 1.1
    where gehalt < 5000.00");
// n enthält die Anzahl der aktualisierten Zeilen
```

executeQuery-Methode

führt Anfragen aus und liefert Ergebnismenge zurück

```
Statement stat1 = con.createStatement ();
ResultSet res1 = stat1.executeQuery (
    "select pnr, name, gehalt from personal where
    gehalt >=" + gehalt);
// Cursor-Zugriff und Konvertierung der DBMS-Datentypen in
// passende Java-Datentypen erforderlich (siehe Cursor-Behandlung)
```

4-59

JDBC – Prepared-Anweisungen

PreparedStatement-Objekt

```
PreparedStatement pstmt;
double gehalt = 5000.00;
pstmt = con.prepareStatement (
    "select * from personal where gehalt >= ?");
```

- Vor der Ausführung sind dann die aktuellen Parameter einzusetzen mit Methoden wie setDouble, setInt, setString usw. und Indexangabe pstmt.setDouble (1, gehalt);
- Neben setXXX () gibt es Methoden getXXX () und updateXXX () für alle Basistypen von Java

Ausführen einer Prepared-Anweisung als Anfrage

```
ResultSet res1 = pstmt.executeQuery ();
```

Vorbereiten und Ausführung

einer Prepared-Anweisung zur DB-Aktualisierung

```
pstmt = con.prepareStatement (
    "delete from personal
    where name = ?");
// set XXX-Methode erlaubt die Zuweisung von aktuellen Werten
pstmt.setString (1, "Maier")

int n = pstmt.executeUpdate ();

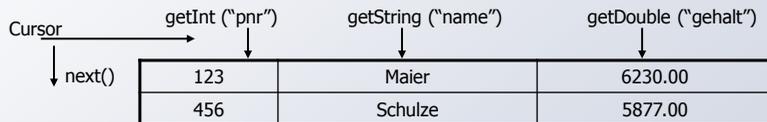
// Methoden für Prepared-Anweisungen haben keine Argumente
```

4-60

JDBC – Ergebnismengen und Cursor

Select-Anfragen und Ergebnisübergabe

- Jede JDBC-Methode, mit der man Anfragen an das DBMS stellen kann, liefert ResultSet-Objekte als Rückgabewert
- ```
ResultSet res = stmt.executeQuery (
 "select pnr, name, gehalt from personal where gehalt >=" + gehalt);
```
- Cursor-Zugriff und Konvertierung der DBMS-Datentypen in passende Java-Datentypen erforderlich
  - JDBC-Cursor ist durch die Methode next() der Klasse ResultSet implementiert



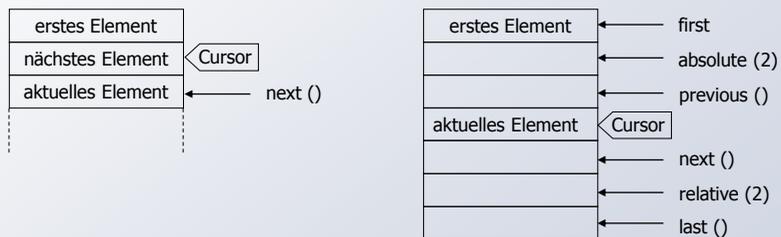
- Zugriff aus Java-Programm

```
while (res.next()) {
 System.out.print (res.getInt ("pnr") + "\\t");
 System.out.print (res.getString ("name") + "\\t");
 System.out.println (res.getDouble ("gehalt"));
}
```

## JDBC – Ergebnismengen und Cursor (2)

### JDBC definiert drei Typen von ResultSets

- **ResultSet: forward-only**  
Default-Cursor vom Typ INSENSITIVE: nur next()
- **ResultSet: scroll-insensitive**
  - Scroll-Operationen sind möglich, aber DB-Aktualisierungen verändern ResultSet nach seiner Erstellung nicht



## JDBC – Ergebnismengen und Cursor (3)

### ■ ResultSet: scroll-sensitive

- Scroll-Operationen sind möglich, wobei ein nicht-INSENSITIVE Cursor benutzt wird
- Semantik der Operation, im Standard nicht festgelegt, wird vom darunterliegenden DBMS übernommen, die vom Hersteller definiert wird!
- Oft wird die sogen. KEYSSET\_DRIVEN-Semantik\* (Teil des ODBC-Standards) implementiert.

### ■ Aktualisierbare ResultSets

```
Statement s1 = con1.createStatement (ResultSet.TYPE_SCROLL_
 SENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet res= s1.executeQuery (. . .); . . .
res.updateString ("name", "Müller"); . . .
res.updateRow ();
```

- Zeilen können in aktualisierbaren ResultSets geändert und gelöscht werden. Mit res.insertRow () wird eine Zeile in res und gleichzeitig auch in die DB eingefügt.

\* Bei Ausführung der Select-Anweisung wird der ResultSet durch eine Menge von Zeigern auf die sich qualifizierenden Zeilen repräsentiert. Änderungen und Löschungen nach Erstellen des ResultSet werden dadurch sichtbar gemacht, Einfügungen aber nicht!

## JDBC – Zugriff auf Metadaten

### ■ Allgemeine Metadaten

- Welche Information benötigt ein Browser, um seine Arbeit beginnen zu können?
- JDBC besitzt eine Klasse DatabaseMetaData, die zum Abfragen von Schema- und anderer DB-Information herangezogen wird

### ■ Informationen über ResultSets

- JDBC bietet die Klasse ResultSetMetaData

```
ResultSet rs1 = stmt1.executeQuery ("select * from personal");
ResultSetMetaData rsm1 = rs1.getMetaData ();
```

- Es müssen die Spaltenanzahl sowie die einzelnen Spaltennamen und ihre Typen erfragt werden können (z. B. für die erste Spalte)

```
int AnzahlSpalten = rsm1.getColumnCount ();
String SpaltenName = rsm1.getColumnName (1);
String TypName = rsm1.getColumnTypeName (1);
```

- Ein Wertzugriff kann dann erfolgen durch

```
rs1.getInt (2), wenn
rsm1.getColumnTypeName (2)
den String "Integer" zurückliefert.
```

## JDBC – Fehler

### ■ Fehlerbehandlung

- Spezifikation der Ausnahmen, die eine Methode werfen kann, bei ihrer Deklaration (throw exception)
- Ausführung der Methode in einem try-Block, Ausnahmen werden im catch-Block abgefangen

```
try{
 . . . Programmcode, der Ausnahmen verursachen kann
}
catch (SQLException e) {
 System.out.println ("Es ist ein Fehler aufgetreten :\n");
 System.out.println ("Msg: " + e.getMessage ());
 System.out.println ("SQLState: " + e.getSQLState ());
 System.out.println ("ErrorCode: " + e.getErrorCode ());
};
```

## JDBC – Transaktionen

### ■ Transaktionen

- Bei Erzeugen eines Connection-Objekts (z.B. con1) ist als Default der Modus **autocommit** eingestellt
- Um Transaktionen als Folgen von Anweisungen abwickeln zu können, ist dieser Modus auszuschalten

```
con1.setAutoCommit(false);
```

- Für eine Transaktion können sogen. Konsistenzebenen (isolation levels) wie TRANSACTION\_SERIALIZABLE, TRANSACTION\_REPEATABLE\_READ usw. eingestellt werden

```
con1.setTransactionIsolation (Connection.TRANSACTION_SERIALIZABLE);
```

### ■ Beendigung oder Zurücksetzen

```
con1.commit();
con1.rollback();
```

### ■ Programm kann mit mehreren DBMS verbunden sein

- selektives Beenden/Zurücksetzen von Transaktionen pro DBMS
- kein globales atomares Commit möglich

## DB-Zugriff via JDBC – Beispiel 1

Kopplung

stat. SQL

Anfrage-auswertung

Anfrage-optimierung

SQL/PSM

dyn. SQL

CLI u. ODBC

JDBC u. SQL

MapReduce



```
import java.sql.*;
public class Select {
 public static void main (String [] args) {
 Connection con = null;
 PreparedStatement pstmt;
 ResultSet res;
 double gehalt = 5000.00;
 try {
 Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
 con = java.sql.DriverManager.getConnection (
 "jdbc:odbc:personal", "user", "passwd");
 pstmt = con.prepareStatement (
 "select pnr, name, gehalt from personal where gehalt >= ?");
 pstmt.setDouble (1, gehalt);
 ...
 res = pstmt.executeQuery ();
 while (res.next ()) {
 System.out.print (res.getInt ("pnr") + "\t");
 System.out.print (res.getString ("name") + "\t");
 System.out.println (res.getDouble ("gehalt"));
 }
 res.close ();
 pstmt.close ();
 } // try
 catch (SQLException e) {
 System.out.println (e);
 System.out.println (e.getSQLState ());
 System.out.println (e.getErrorCode ());
 }
 catch (ClassNotFoundException e) {
 System.out.println (e);
 }
 } // main
} // class Select
```

4-67

## DB-Zugriff via JDBC – Beispiel 2

Kopplung

stat. SQL

Anfrage-auswertung

Anfrage-optimierung

SQL/PSM

dyn. SQL

CLI u. ODBC

JDBC u. SQL

MapReduce



```
import java.sql.*;
public class Insert {
 public static void main (String [] args) {
 Connection con = null;
 PreparedStatement pstmt;
 try {
 Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
 con = java.sql.DriverManager.getConnection ("jdbc:odbc:personal", " ", " ");
 pstmt = con.prepareStatement ("insert into personal values (?, ?, ?)");
 pstmt.setInt (1, 222);
 pstmt.setString (2, "Schmitt");
 pstmt.setDouble (3, 6000.00);
 pstmt.executeUpdate ();
 pstmt.close ();
 con.close ();
 } // try
 catch (SQLException e) {
 System.out.println (e);
 System.out.println (e.getSQLState ());
 System.out.println (e.getErrorCode ());
 }
 catch (ClassNotFoundException e) {System.out.println (e);
 }
 }
}
...
pstmt = con.prepareStatement (
 "update personal set gehalt = gehalt * 1.1 where gehalt < ?");
pstmt.setDouble (1, 10000.00);
pstmt.executeUpdate ();
pstmt.close ();
...
pstmt = con.prepareStatement ("delete from personal where pnr = ?");
pstmt = setInt (1,222);
pstmt.executeUpdate ();
pstmt.close ();
```

4-68

# SQLJ

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL**
- MapReduce

## ■ SQLJ

- Teil 0 der SQLJ-Spezifikation beschreibt die Einbettung von SQL in Java-Anwendungen (ANSI-Standard)
- besitzt bessere Lesbarkeit, Verständlichkeit und Wartbarkeit durch kurze und prägnante Schreibweise
- zielt auf die Laufzeiteffizienz von eingebettetem SQL ab, ohne die Vorteile des DB-Zugriffs via JDBC aufzugeben

## ■ Im Folgenden werden nur einige Unterschiede zu eingebettetem SQL und JDBC aufgezeigt

### ■ Verbindung zum DBMS

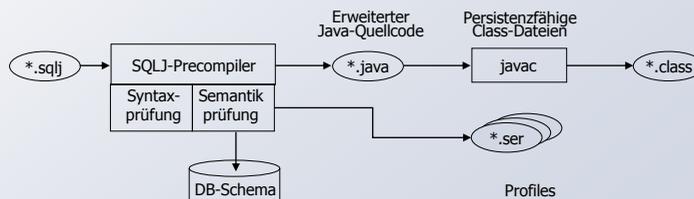
- erfolgt über sog. Verbindungskontexte (ConnectionContext)
- Sie basieren auf JDBC-Verbindungen und werden auch so genutzt (URL, Nutzernamen, Paßwort)
- SQLJ-Programm kann **mehrere Verbindungskontexte** über verschiedene JDBC-Treiber aufbauen; sie erlauben den parallelen Zugriff auf mehrere DBMS oder aus mehreren Threads/Prozessen auf das gleiche DBMS

# SQLJ (2)

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL**
- MapReduce

## ■ Abbildung auf JDBC durch Precompiler

- Überprüfung der Syntax sowie (gewisser Aspekte) der Semantik von SQL-Anweisungen (Anzahl und Typen von Argumenten usw.) zur Übersetzungszeit, was Kommunikation mit dem DBMS zur Laufzeit erspart
- Ersetzung der SQLJ-Anweisungen durch Aufrufe an das SQLJ-Laufzeitmodul (Package `sqlj.runtime.*`)
- Erzeugung sog. Profiles, serialisierbare Java-Klassen, welche die eigentlichen JDBC-Anweisungen enthalten
- Abwicklung von DB-Anweisungen vom SQLJ-Laufzeitmodul dynamisch über die Profiles, die wiederum über einen JDBC-Treiber auf die DB zugreifen
- Anpassung an ein anderes DBMS geschieht durch Austausch der Profiles (sog. Customizing)



## SQLJ (3)

### ■ SQL-Anweisungen sind im Java-Programm Teil einer SQLJ-Klausel

```
#SQL {
 select p.onr into :persnr
 from personal p
 where p.beruf = :beruf
 and p.gehalt > :gehalt};
```

- Austausch von Daten zwischen SQLJ und Java-Programm erfolgt über Wirtssprachenvariablen
- Parameterübergabe kann vorbereitet werden
- ist viel effizienter als bei JDBC (mit ?-Platzhaltern)

### ■ Iteratoren

- analog zu JDBC-ResultSets
- Definition von Iteratoren (Cursor), aus denen entsprechende Java-Klassen generiert werden, über die auf die Ergebnismenge zugegriffen wird

### ■ SQLJ und JDBC

- Ebenso wie statische und dynamische SQL-Anweisungen in einem Programm benutzt werden können, können SQLJ-Anweisungen und JDBC-Aufrufe im selben Java-Programm auftreten.

4-71

## SQLJ (4)

### ■ Nutzung eines Iterators in SQLJ

```
import java.sql.*
...
#SQL iterator GetPersIter (int personalNr, String nachname);
Get PersIter iter1;
#SQL iter1 = {select p.pnr as "personalNr", p.name as "nachname"
 from personal p
 where p.beruf = :Beruf
 and p.gehalt = :Gehalt};

int Id;
String Name;
while (iter1.next ()) {
 Id = iter1.personalNr ();
 Name = iter1.nachname ();
 ... Verarbeitung ...
}
iter1.close ();
```

Die as-Klausel wird benutzt, um die SQL-Attributnamen im Ergebnis mit den Spaltennamen im Iterator in Beziehung zu setzen.

### ■ SQLJ liefert für eine Anfrage ein SQLJ-Iterator-Objekt zurück

- SQLJ-Precompiler generiert Java-Anweisungen, die eine Klasse GetPersIter\* definieren
- GetPersIter kann als Ergebnisse Zeilen mit zwei Spalten (Integer u. String) aufnehmen
- Deklaration gibt den Spalten Java-Namen (personalNr u. nachname) und definiert implizit Zugriffsmethoden personalNr() u. n achname(), die zum Iterator-Zugriff benutzt werden

\* Sie implementiert das Interface sqll.runtime.NamedIterator (spezialisiert vom Standard-Java-Interface java.util.Iterator)

4-72

## SQLJ (5)

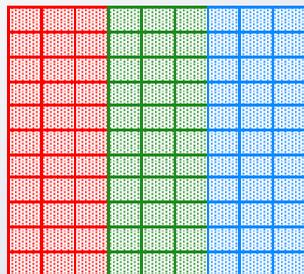
- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL**
- MapReduce

- **DB-seitige Nutzung von Java mit SQLJ**
  - Teil 1 des SQLJ-Standards beschreibt, wie man Stored Procedures mit Java realisieren kann
  - Sprache für Stored Procedures bisher nicht standardisiert, Wildwuchs von Implementierungen
  - erste Sprache für portable Stored Procedures
  - automatisiertes Installieren/Entfernen von Stored Procedures in/aus DBMS (Einsatz sog. Deployment Descriptors)
- **DB-seitige Verwendung von Java-Datentypen**
  - Teil 2 des SQLJ-Standards beschreibt Verfahren, wie Java-Datentypen als SQL-Datentypen verwendet werden können
  - Umgekehrt können für herkömmliche SQL-Typen Wrapper-Klassen automatisch generiert werden

## MapReduce Paradigm\*

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce**

- **Parallel programming\*\***
  - Identify sets of tasks that can run concurrently and/or partitions of data that can be processed concurrently.
  - This is not always possible. See a Fibonacci function  $F_{k+2} = F_k + F_{k+1}$
  - A common situation: A large amount of consistent data, which can be decomposed into (~) equal-sized partitions



Subarray 1 Subarray 2 Subarray 3 ...

\* Michael Stonebraker: SQL databases v. NoSQL databases. Commun. ACM 53(4): 10-11 (2010)  
 \*\* David DeWitt; Michael Stonebraker: „MapReduce: A major step backwards“. databasecolumn.com  
<http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/>.

## MapReduce Paradigm (2)

- A common implementation technique is called *Master/Worker*
- *The Master*
  - initializes the array and splits it up for the available Workers
  - sends each Worker its subarray
  - receives the result from each Worker
- *The Worker*
  - receives the subarray from the Master
  - performs processing on its subarray
  - returns results to the Master
- ↳ **Static load balancing only useful if all tasks are performing the same amount of work on identical machines**
- *Dynamic algorithms* are more flexible, though more computationally expensive; they consider the network load before allocating a new process to a processor

## MapReduce Paradigm (3)

- **Programming model**
  - Input & Output: each a set of key/value pairs
  - Programmer specifies two functions:
 

```
map(in_key, in_value) -> list(out_key, intermediate_value)
```

    - Processes input key/value pair
    - Produces set of intermediate pairs

```
reduce(out_key, list(intermediate_value)) -> list(out_value)
```

    - Combines all intermediate values for a particular key
    - Produces a set of merged output values (usually just one)
  - Inspired by similar primitives in LISP and other languages
- **Model is widely applicable**
  - distributed grep, distributed sort, Web link-graph traversal
  - term-vector per host, Web access log stats, inverted index construction
  - document clustering, machine learning, statistical machine translation
  - ...

## MapReduce Paradigm (4)

### ■ Example: Count word occurrences

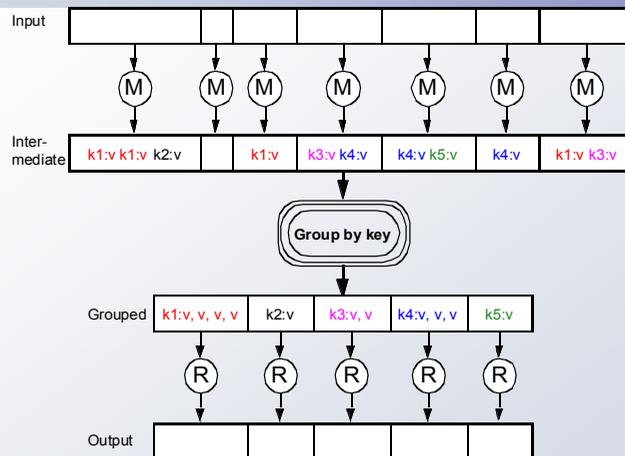
```

// Pseudo-code for "word counting"
map(String input_key, String input_value):
 // input_key: document name
 // input_value: document contents
 for each word w in input_value:
 EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
 // output_key: a word
 // output_values: a list of counts
 int word_count = 0;
 for each v in intermediate_values:
 word_count += ParseInt(v);
 Emit(output_key, AsString(word_count));

```

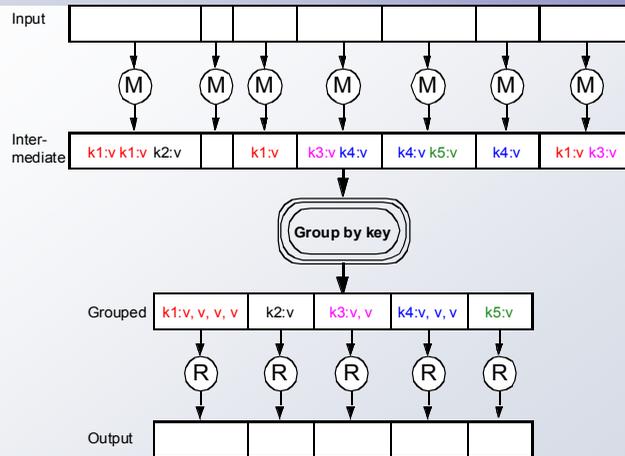
## Logical Structure of a MapReduce Application



### ■ Map function

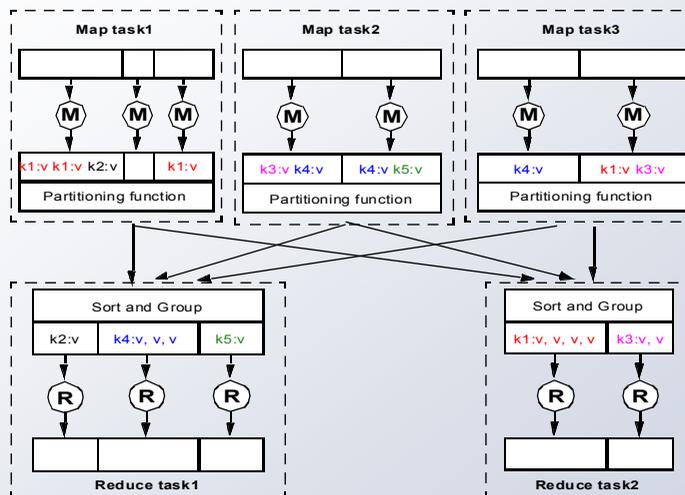
- Each worker processes several parts
- Maps to key/value pairs
- Each Map task partitions the data for the Reduce process
- Standard partitioning function: Hash(key) modulo (# Reduce tasks)

## Logical Structure of a MapReduce Application (2)



- **Group by key**
  - Partitioned intermediate results are sorted by key
  - Reduce tasks operate on sorted (partitioned) data
- **Reduce function**
  - is called for each key value
  - produces (sorted) output as final result

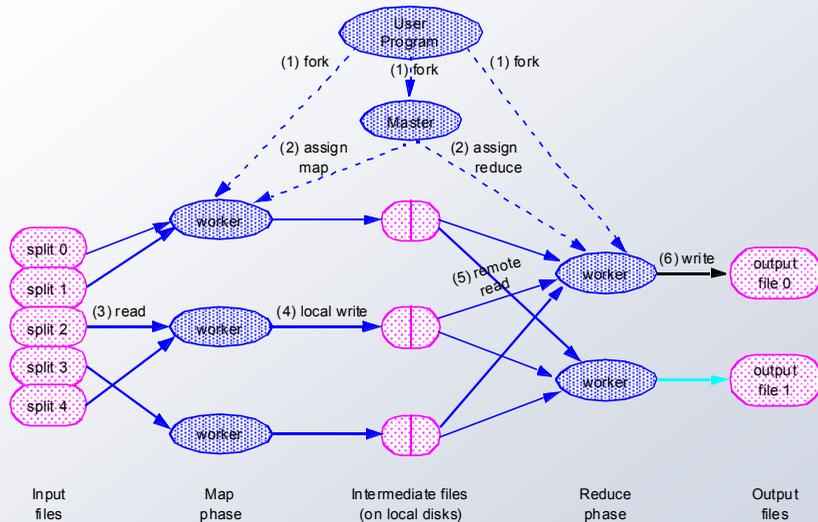
## Assignment of MapReduce Tasks





- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce**

## MapReduce Framework



- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce**

## MapReduce Framework (2)

### ■ Distribution of components\*

- **Map** invocations are distributed across multiple machines by automatically partitioning the input data set into a set of M splits or shards, which can be processed in parallel.
- **Reduce** invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function (e. g.,  $\text{hash}(\text{key}) \bmod R$ ). The number of partitions (R) and the partitioning function are specified by the user.

### ■ MapReduce execution overview

1. The MapReduce library in the user program first shards the input files into M pieces of typically 16 to 64 MB per piece. It then starts up many copies of the program on a cluster of machines.
2. One of the copies of the program is special: the master. The rest are workers that are assigned work by the master. There are M Map tasks and R Reduce tasks to assign. The master picks idle workers and assigns each one a Map task or a Reduce task.
3. A worker who is assigned a map task reads the contents of the corresponding input shard. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.

\* Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. Sixth Symposium on Operating System Design and Implementation, OSDI 2004: 137-150

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce**

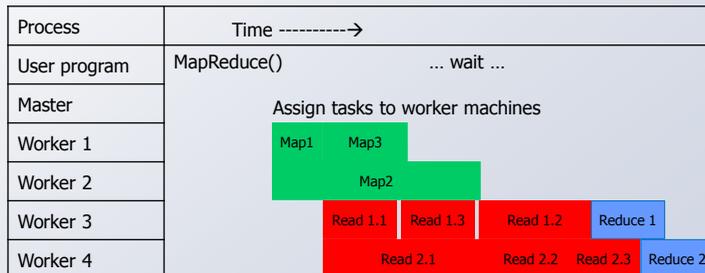
## MapReduce Framework (3)

4. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the Reduce workers.
5. When a Reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the Map workers. When a Reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. If the amount of intermediate data is too large to fit in memory, an external sort is used.
6. The Reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for this Reduce partition.
7. When all Map tasks and Reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce**

## Implementation Overview

- **Typical cluster:**
  - 100s/1000s of 2-CPU x86 machines, 2-4 GB of memory
  - Storage is on local IDE disks
  - GFS: distributed file system manages data
  - Job scheduling system: jobs made up of tasks, tasks assigned to machines
- **Framework implementation is a C++ library linked into user programs**
- **Task granularity and pipelining**
  - Fine granularity tasks: many more Map tasks than machines
    - Minimizes time for fault recovery
    - Can pipeline shuffling with Map execution
    - Better dynamic load balancing
  - Often use 200,000 Map /5000 Reduce tasks w/ 2000 machine



- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce**



## Implementation Overview (2)

- **Storage system (Google File System)\***
  - Distribution and optimization of large files
  - High throughput of disks
  - Appropriate distribution and allocation of workers
  - Tuning of chunk size and split size

→ **Storage system is basis for MapReduce performance**
- **Fault tolerance or delay of computing nodes**
  - On worker failure
    - Detect failure via periodic heartbeats
    - Re-execute completed and in-progress Map tasks
    - Task completion committed through master
  - Master failure
    - Could handle, but don't yet (master failure unlikely)

→ **Robust: lost 1600 of 1800 machines once, but finished successful**

\* Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System, Proc. 19th ACM Symposium on Operating Systems Principles, pp. 29-43, Lake George, NY, Oct. 2003

- Kopplung
- stat. SQL
- Anfrage-auswertung
- Anfrage-optimierung
- SQL/PSM
- dyn. SQL
- CLI u. ODBC
- JDBC u. SQL
- MapReduce**



## Implementation Overview (3)

### ■ Use of MapReduce inside Google

|                             | August 2004 | March 2006 | September 2007 |
|-----------------------------|-------------|------------|----------------|
| Number of jobs (1000s)      | 29          | 171        | 2.217          |
| Avg. Completion time (secs) | 634         | 874        | 395            |
| Machine years used          | 217         | 2.002      | 11.081         |
| Map input data (TB)         | 3,288       | 52.254     | 403,152        |
| Map output data (TB)        | 758         | 6.743      | 34,774         |
| Reduce output data (TB)     | 193         | 2.970      | 14,028         |
| Avg. machines per job       | 157         | 268        | 394            |
| Unique implementations      |             |            |                |
| Map                         | 395         | 1958       | 4083           |
| Reduce                      | 269         | 1208       | 2418           |

→ **By combining multi-level execution and columnar data layout, Dremel is capable of running aggregate queries over trillion-row tables in secs\***

### Hadoop\*

- Open-source framework written in Java
- based on the MapReduce algorithm and GFS
- scalable for data-intensive computations of computer clusters

\* Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis: Dremel: Interactive Analysis of Web-Scale Datasets. PVLDB 3(1): 330-339 (2010)

## Zusammenfassung

- **Cursor-Konzept zur satzweisen Verarbeitung von Datenmengen**
  - Anpassung von mengenorientierter Bereitstellung und satzweiser Verarbeitung von DBMS-Ergebnissen
  - Operationen: DECLARE CURSOR, OPEN, FETCH, CLOSE
  - Erweiterungen: Scroll-Cursor, Sichtbarkeit von Änderungen
- **Statisches (eingebettetes) SQL**
  - hohe Effizienz, gesamte Typprüfung und Konvertierung erfolgen durch Precompiler
  - relativ einfache Programmierung
  - Aufbau aller SQL-Befehle muss zur Übersetzungszeit festliegen
  - es können zur Laufzeit nicht verschiedene Datenbanken dynamisch angesprochen werden
- **Interpretation einer DB-Anweisung**
  - allgemeines Programm (Interpreter) akzeptiert Anweisungen der DB-Sprache als Eingabe und erzeugt mit Hilfe von Aufrufen des Zugriffssystems Ergebnis
  - hoher Aufwand zur Laufzeit (v.a. bei wiederholter Ausführung einer Anweisung)
- **Übersetzung, Code-Erzeugung und Ausführung einer DB-Anweisung**
  - für jede DB-Anweisung wird ein zugeschnittenes Programm erzeugt (Übersetzungszeit), das zur Laufzeit abgewickelt wird und dabei mit Hilfe von Aufrufen des Zugriffssystems das Ergebnis ableitet
  - Übersetzungsaufwand wird zur Laufzeit soweit wie möglich vermieden

4-89

## Zusammenfassung (2)

- **Kostenmodell**
  - Minimierung der Kosten in Abhängigkeit des Systemzustandes
  - Problem: Aktualisierung der statistischen Kenngrößen
- **Anfrageoptimierung: Kernproblem** der Übersetzung mengenorientierter DB-Sprachen
  - **"fatale" Annahmen:**
    - Gleichverteilung aller Attributwerte
    - Unabhängigkeit aller Attribute
  - **Kostenvoranschläge für Ausführungspläne:**
    - CPU-Zeit und E/A-Aufwand
    - Anzahl der Nachrichten und zu übertragende Datenvolumina (im verteilten Fall)
  - gute Heuristiken zur Auswahl von Ausführungsplänen sehr wichtig
- **PSM**
  - zielt ab auf Leistungsverbesserung vor allem in Client/Server-Umgebung
  - erhöht die Verarbeitungsmächtigkeit des DBMS
- **Dynamisches SQL**
  - Festlegung/Übergabe von SQL-Anweisungen zur Laufzeit
  - hohe Flexibilität, schwierige Programmierung
- **Unterschiede in der SQL-Programmierung zu eingebettetem SQL**
  - explizite Anweisungen zur Datenabbildung zwischen DBMS und Programmvariablen
  - klare Trennung zwischen Anwendungsprogramm und SQL (⇒ **einfacheres Debugging**)

4-90