10. Große Objekte

- **Neue Anforderungen für große Objekte (*Large Objects*)**
  - vordefinierte Datentypen
  - große operationale Unterschiede zu Basistypen

- **Spezifikation von LOBs**[1]

- **Unterstützung für LOBs mit Einschränkungen**
  - Auswertung von Prädikaten
  - Verarbeitung und Indexierung

- **Lokator-Konzept**
  - „Verweis" auf in DB gespeicherten LOB
  - Kapselung des Zugriffs

- **Dateireferenzen**

- **Speicherungsstrukturen für LOBs**
  - Segmente fester Größe (Exodus)
  - Segmente mit einem festen Wachstumsmuster (Starburst)
  - Segmente variabler Größe (EOS)
  - Zugriff über B*-Baum, Zeigerliste, . . .

- **DB-Anbindung externer Daten**
  - DataLinks-Konzept
  - Referentielle Integrität, Zugriffskontrolle, Transaktionskonsistenz, Koordiniertes Backup und Recovery

---

1. Die Realisierungsbeispiele beziehen sich auf DB2 - Universal Database

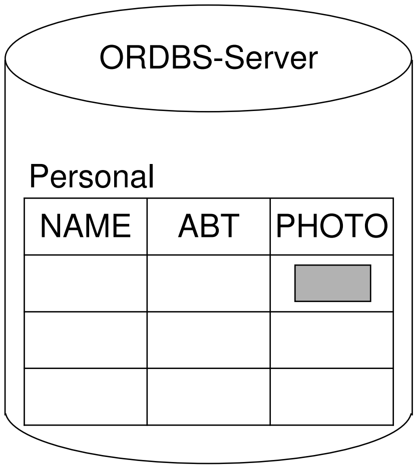
Große Objekte

- **Anforderungen**
  - idealerweise keine Größenbeschränkung
  - allgemeine Verwaltungsfunktionen
  - zugeschnittene Verarbeitungsfunktionen, . . .

- **Beispiele für große Objekte (heute bis n (=2) GByte)**
  - Texte, CAD-Daten
  - Bilddaten, Tonfolgen
  - Videosequenzen, . . .

- **Prinzipielle Möglichkeiten der DB-Integration**

Speicherung als LOB in der DB (meist indirekte Speicherung)

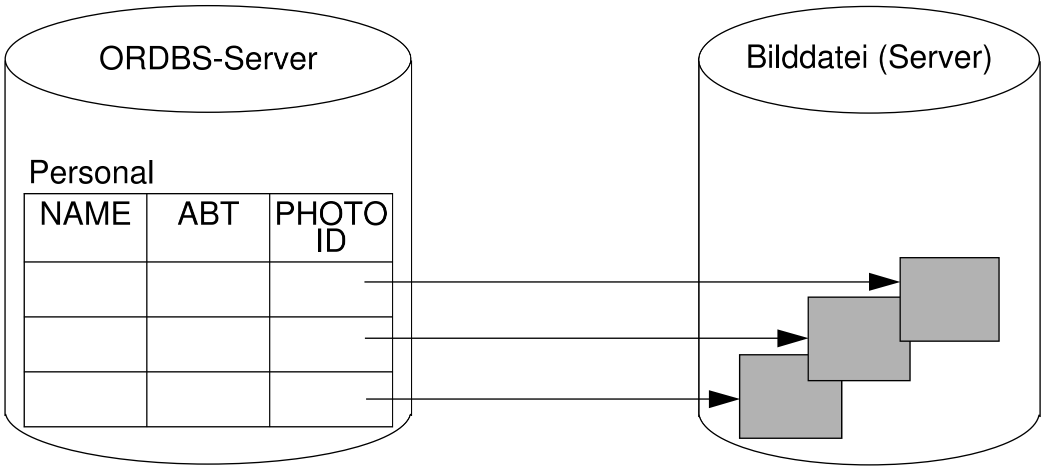


BLOB - *Binary Large Object für Tonfolgen, Bilddaten usw.*

CLOB - *Character Large Object für Textdaten*

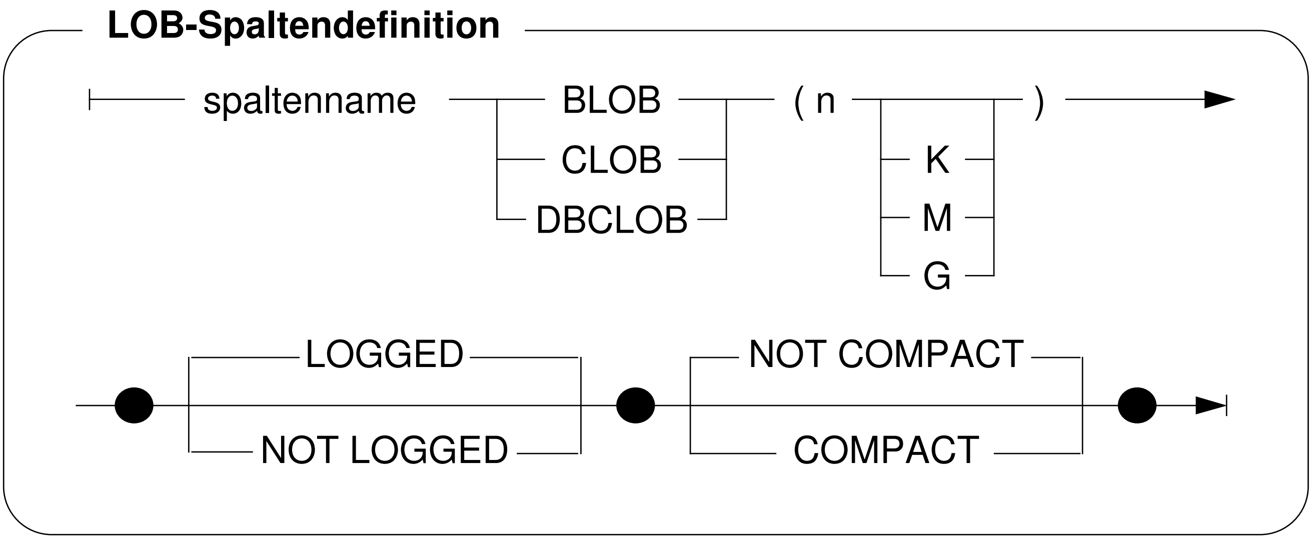
DBCLOB - *Double Byte Character Large Object (DB2) für spez. Graphikdaten usw.*

Speicherung mit DataLink-Konzept in externen Datei-Servern



Große Objekte (2)

- **Erzeugung von LOB-Spalten**



- **Beispiele**

```
CREATE TABLE      Absolvent
   (Lfdnr         Integer,
   Name           Varchar (50),
   . . .
   Photo          BLOB (5 M) NOT LOGGED COMPACT,       -- Bild
   Lebenslauf     CLOB (16 K) NOT LOGGED COMPACT);     -- Text

CREATE TABLE      Entwurf
   (Teilnr            Char (18),
   Änderungsstand     Timestamp,
   Geändert_von       Varchar (50)
   Zeichnung          BLOB (2 M) LOGGED NOT COMPACT);  -- Graphik

ALTER TABLE       Absolvent
   ADD COLUMN     Diplomarbeit CLOB (500 K)
                  LOGGED NOT COMPACT;
```

Große Objekte (3)

- **Spezifikation von LOBs erfordert Sorgfalt**
  - **maximale Länge**
    - Reservierung eines Anwendungspuffers
    - Clusterbildung und Optimierung durch indirekte Speicherung;
      Deskriptor im Tupel ist abhängig von der LOB-Länge
      (72 Bytes bei <1K - 316 Bytes bei 2G)
    - bei kleinen LOBs (< Seitengröße) direkte Speicherung möglich
  - **Kompaktierung**
    - COMPACT reserviert keinen Speicherplatz für späteres Wachstum

      ➥ Was passiert bei einer LOB-Aktualisierung?
    - NOT COMPACT ist Default
  - **Logging**
    - LOGGED: LOB-Spalte wird bei Änderungen wie alle anderen Spalten behandelt (ACID!)

      ➥ Was bedeutet das für die Log-Datei?
    - NOT LOGGED: Änderungen werden nicht in der Log-Datei protokolliert.
      Sog. Schattenseiten (shadowing) gewährleisten Atomarität bis zum Commit

      ➥ Was passiert bei Gerätefehler?

Große Objekte (4)

- **Wie werden große Objekte verarbeitet?**
  - BLOB und CLOB sind keine Typen der Wirtssprache
  - ➥ Spezielle Deklaration von BLOB, CLOB, ... durch SQL TYPE ist erforderlich, da sie die gleichen Wirtssprachentypen benutzen. Außerdem wird sichergestellt, daß die vom DBS erwartete Länge genau eingehalten wird.

- **Vorbereitungen im AWP erforderlich**
  - SQL TYPE IS CLOB (2 K) c1 (oder BLOB (2 K))
    wird durch C-Precompiler übersetzt in

    ```
    static struct c1_t
        {
        unsigned long length;
        char data [2048];
        } c1;
    ```

  - Erzeugen eines CLOB

    ```
        c1.data = ‘Hello’;
        c1.length = sizeof (‘Hello’)-1;
    ```

    kann durch Einsatz von Makros (z. B. c1 = SQL_CLOB_INIT(‘Hello’);)
    verborgen werden

- **Einfügen, Löschen und Ändern**
  kann wie bei anderen Typen erfolgen, wenn genügend große AW-Puffer vorhanden sind

- **Hole die Daten des Absolventen mit Lfdnr. 17 ins AWP**

Große Objekte (5)

```
void main ( )                                    /*  Beispielprogramm                    */
  {                                              /*  Verarbeitung von Filmkritiken auf   */
                                                 /* Tabelle Filme (Titel, Besetzung, Kritik) */
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
      char dbname[9] = “Filmdb“;                 /*  Name der Datenbank*/
      char msgbuffer[500];                       /*  Puffer für DB2-Fehlermeldungen*/
      char titel[100];                           /*  für Varchar-Daten*/
      SQL TYPE is CLOB (50 K) kritik;            /*  Ausgabe-Clob-Struktur*/
      SQL TYPE is CLOB (50 K) neuekritik;        /*  Eingabe-Clob-Struktur*/
      short indikator1, indikator2;              */  Indikator-Variable */
  EXEC SQL END DECLARE SECTION;
  EXEC SQL WHENEVER SQLERROR GO TO schlechtenachrichten;
  EXEC SQL CONNECT TO :dbname;

  strcpy (neuekritik.data, “Bullet ist ein ziemlich guter Film.“);
  neuekritik.length = strlen (neuekritik.data);
  indikator1 = 0;
  EXEC SQL
      UPDATE Filme
      SET       Kritik = :neuekritik :indikator1
      WHERE   Titel = ‘Bullet’;
  EXEC SQL COMMIT;
  EXEC SQL DECLARE f1 CURSOR FOR
      SELECT Titel, Kritik
      FROM     Filme
      WHERE   Besetzung LIKE ‘%Steve McQueen%’;
  EXEC SQL WHENEVER NOT FOUND GO TO close_f1;
  EXEC SQL OPEN f1;
  WHILE (1)
      {
      EXEC SQL FETCH f1 INTO :titel, :kritik :indikator2;
      /* Angabe eines eigenen Nullterminierers */
      kritik.data[kritik.length] = ‘\0’;
      printf(„\nTitel: %s\n“, titel);
      if   (indikator2 < 0)
          printf (“Keine Kritik vorhanden\n“);
      else
          printf(“%s\n“, kritik.data);
      }
  close_f1:
      EXEC SQL CLOSE f1;
      return;
  schlechtenachrichten:
      printf (“Unerwarteter DB2-Return-Code.\n“);
      sqlaintp (msgbuffer, 500, 70, &sqlca);
      printf (“Message: &s\n“, msgbuffer);
      }   /* End of main */
```

Große Objekte (6)

- **Welche Operationen können auf LOBs angewendet werden?**
  - Vergleichsprädikate: =, <>, <, <=, >, >=, IN, BETWEEN
  - LIKE-Prädikat
  - Eindeutigkeit oder Reihenfolge bei LOB-Werten
    - PRIMARY KEY, UNIQUE, FOREIGN KEY
    - SELECT DISTINCT, . . ., COUNT (DISTINCT)
    - GROUP BY, ORDER BY
  - Einsatz von Aggregatfunktionen wie MIN, MAX
  - Operationen
    - UNION, INTERSECT, EXCEPT
    - Joins von LOB-Attributen
  - Indexstrukturen über LOB-Spalten
- **Wie indexiert man LOBs?**
  - Benutzerdefinierte Funktion ordnet LOBs Werte zu
  - Funktionswert-Indexierung

Große Objekte (7)

- **Verarbeitungsanforderungen bei LOBs**
  - Verkürzen, Verlängern und Kopieren
  - Suche nach vorgegebenem Muster, Längenbestimmung
  - Stückweise Handhabung (Lesen und Schreiben), . . .

    ➥ Einsatz von Funktionen bietet manchmal Ersatzlösungen

- **Funktionen für CLOBs und BLOBs**
  - string1 | | string2 oder CONCAT (string1, string2)
  - SUBSTRING (string FROM start [FOR length])
  - LENGTH (expression)
  - POSITION (search-string IN source-string)
  - OVERLAY (string1 PLACING string2 FROM start [FOR length])
  - TRIM ([[{LEADING | TRAILING | BOTH}] [string1] FROM] string2)
  - . . .

Große Objekte (8)

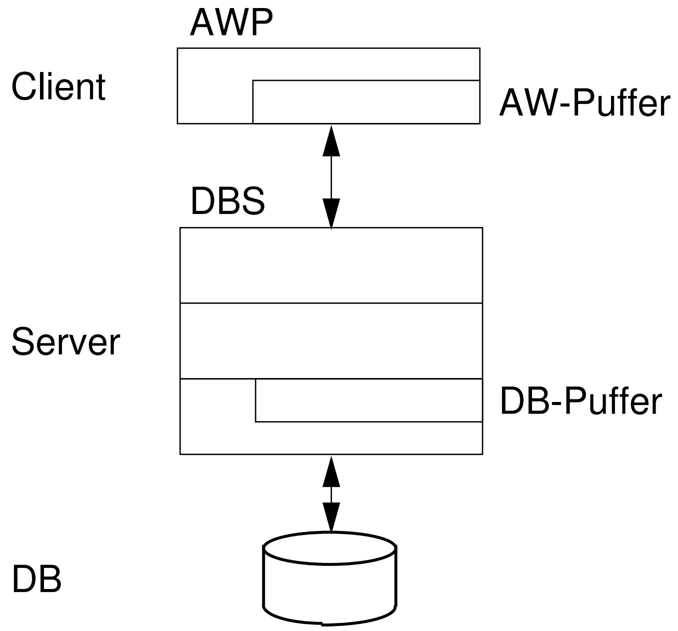
- **Ist die direkte Verarbeitung von LOBs im AWP realistisch?**

```
Bücher
  (Titel         Varchar (200),
   BNR           ISBN,
   Kurzfassung   CLOB (32 K),
   Buchtext      CLOB (20 M),
   Video         BLOB (2 G) )
```

```
EXEC SQL
  SELECT Kurzfassung, Buchtext, Video
  INTO :kilopuffer, :megapuffer, :gigapuffer

  FROM Bücher
  WHERE Titel = ‘American Beauty’
```

- **Client/Server-Architektur**



- Allokation von Puffern?

- Transfer eines ganzen LOB ins AWP?

- Soll Transfer über DBS-Puffer erfolgen?

- „Stückweise“ Verarbeitung von LOBs durch das AWP erforderlich!

  ➥ Lokator-Konzept für den Zugriff auf LOBs

Lokator-Konzept

- **Ziel**
  - Minimierung des Datenverkehrs zwischen Client und Server:
    Es sollen „stückweise“ **so wenig** LOB-Daten **so spät wie möglich** ins AWP übertragen werden
  - **Noch besser:** Bereitstellung von Server-Funktionen
    Durchführung von Operationen auf LOBs durch das DBMS

- **Lokator-Datentyp**
  - Wirtsvariable, mit der ein LOB-Wert referenziert werden kann
    - In C wird long als Datentyp benutzt (4-Byte-Integer)
    - Jedoch Typisierung erforderlich

      ```
      SQL TYPE IS BLOB_LOCATOR
      SQL TYPE IS CLOB_LOCATOR
      ```
  - Identifikator für persistente und flüchtige DB-Daten

- **Anwendung**

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_LOCATOR Video_Loc;
EXEC SQL END DECLARE SECTION;
EXEC SQL
  SELECT    Video
  INTO      :Video_Loc
  FROM      Bücher
  WHERE     Titel = ‘American Beauty’
```

  - Ein Lokator kann überall dort eingesetzt werden, wo ein LOB-Wert verwendet werden kann
    - Wirtsvariable (z. B. in UPDATE-Anweisung)
    - Parameter von Routinen
    - Rückgabewerte von Funktionen
  - Wie lange ist eine Lokator-Referenz gültig?

Lokator-Konzept (2)

- **Lokatoren können LOB-Ausdrücke repräsentieren**
  - Innerhalb des DB-Servers entspricht jeder Lokator einer Art „Rezept“ zum Zusammenbau eines LOB-Wertes aus an unterschiedlichen Stellen gespeicherten Fragmenten
  - Ein LOB-Ausdruck ist ein Ausdruck, der auf eine LOB-Spalte verweist oder einen LOB-Datentyp als Ergebnis hat. Er kann LOB-Funktionen beinhalten.
  - LOB-Ausdrücke können andere Lokatoren referenzieren.

- **Beispiel**

```
SELECT
  SUBSTRING    (Buchtext FROM
     POSITION  (‘Kapitel 1’ IN Buchtext) FOR (
     POSITION  (‘Kapitel 2’ IN Buchtext) –
     POSITION  (‘Kapitel 1’ IN Buchtext)) )
INTO    : Kap1Loc
FROM    Bücher
WHERE   Titel = ‘American Beauty’
```

Lokator-Konzept (3)

- **Mächtigkeit des Lokator-Konzeptes**
  - Ein Lokator repräsentiert immer einen konstanten Wert
  - Operationen auf LOBs werden nach Möglichkeit indirekt mit Hilfe ihrer Verweise („Rezepte“) vorgenommen

    CONCAT (:loc1, :loc2) erzeugt einen neuen Verweis, ohne die physische Konkatenation der LOBs vorzunehmen
  - Ein Anlegen oder Kopieren von LOBs erfolgt nur
    - beim Aktualisieren einer LOB-Spalte
    - bei der Zuweisung eines LOB-Wertes zu einer Wirtsvariablen

- **Einsatz von Lokator-Variablen**
  - LENGTH (:loc1)
  - POSITION (‘Schulabschluss’ IN :loc2)
  - SUBSTRING (:loc3 FROM 1200 FOR 200)
  - EXEC SQL VALUES

    ```
    EXEC SQL VALUES
          SUBSTRING ( :loc1 FROM POSITION ( 'Schulabschluss' IN :loc1 )
          FOR 100 ) INTO :loc2
    ```

- **Lebensdauer von Lokatoren**
  - Explizite Freigabe

    ```
    EXEC SQL
          FREE LOCATOR :loc1, :loc2;
    ```
  - Transaktionsende (non-holdable locators)
  - Sitzungsende

    ```
    EXEC SQL
          HOLD LOCATOR :loc1;
    ```

Lokator-Konzept (4)

- **Beispielprogramm Theaterstück:**
  Korrektur eines Textes in Tabelle Theaterstücke (Titel, Text, ...)

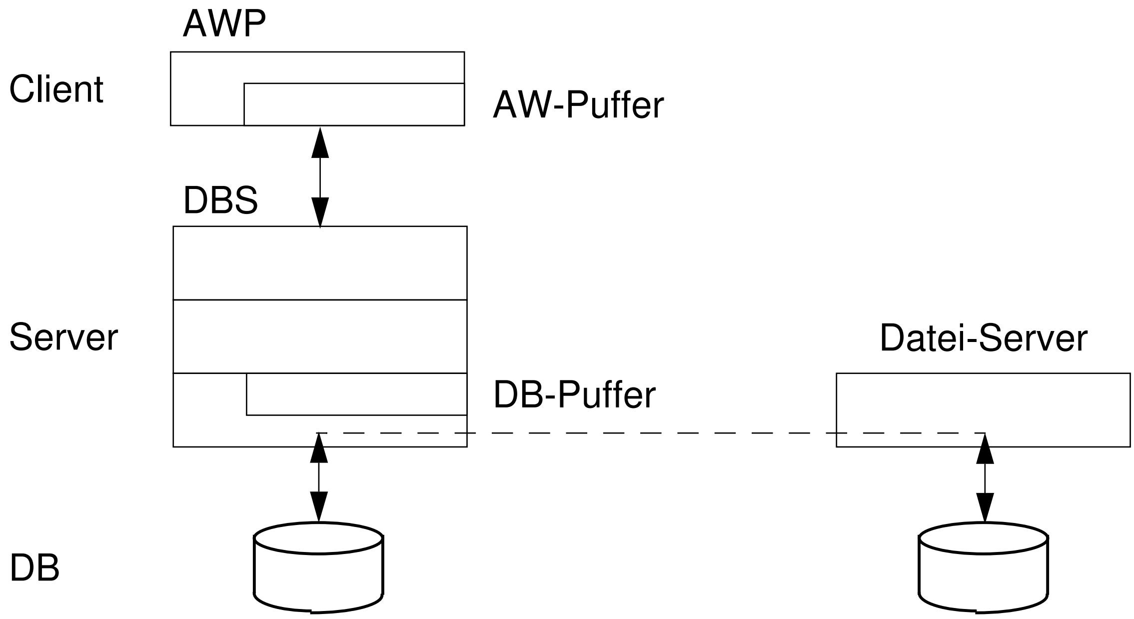
```
void main ( )
  {
  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
       char dbname[9] = „Stückedb“;                /*  Name der Datenbank             */
       char msgbuffer[500];                        /*  Puffer für DB2-Fehlermeldungen */
       SQL TYPE IS CLOB_LOCATOR loc1, loc2;
       long n;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL WHENEVER SQLERROR GO TO schlechtenachrichten;
  EXEC SQL CONNECT TO :dbname;
  EXEC SQL     SELECT   Text INTO :loc1
               FROM     Theaterstücke
               WHERE    Titel = ‘As You Like It’;
  EXEC SQL VALUES POSITION ( ‘colour’ IN :loc1 ) INTO :n;

  while (n > 0)
       {
       EXEC SQL VALUES SUBSTRING ( :loc1 FROM 1 FOR :n-1) ||  ’color’
                           ||  SUBSTRING (:loc1 FROM :n+6) INTO :loc2;
       /*
       ** Gib alten Lokator frei und behalte den neuen.
       */
       EXEC SQL FREE LOCATOR :loc1;
       loc1 = loc2;
       EXEC SQL VALUES POSITION ( ‘colour’ IN :loc1 ) INTO :n;
       }
  /*
  ** Es wurden noch keine Daten bewegt; es wurden lediglich neue Lokatoren erzeugt.
  */
  EXEC SQL  UPDATE Theaterstücke SET Text = :loc1
            WHERE Titel = ‘As You Like It’;
  /*
  ** Jetzt wird der neue Text zusammengesetzt
  ** und der DB-Tabelle Theaterstücke zugewiesen.
  */
  EXEC SQL COMMIT;
  return;
  . . .
```

Dateireferenzen

- **Transfer eines LOB-Wertes ohne Zwischenpufferung**
  - aus einer Datei in die DB
  - aus der DB in eine Datei



- **Dateireferenzdeklaration im SQL-Vereinbarungsteil des AWP**

  SQL TYPE IS CLOB_FILE F1;
  wird vom C-Precompiler in eine Struktur zur Darstellung einer Dateireferenz im Wirtsprogramm übersetzt

```
struct
   {
   unsigned long name_length;        /* Länge des Dateinamens         */
   unsigned long data_length;        /* Länge der Daten in der Datei  */
   unsigned long file_options;       /* Art der Dateibenutzung        */
   char name [255]                   /* Dateiname                     */
   } F1;
```

Dateireferenzen (2)

- **Optionen**
  - Art der Dateibenutzung
    - SQL_FILE_READ
    - SQL_FILE_CREATE
    - SQL_FILE_OVERWRITE, ...
  - Dateiname
    - absoluter Pfadname /u/homes/haerder/awp/myphoto
    - relativer Pfadname awp/myphoto
      wird an den aktuellen Pfad des Client-Prozesses angehängt

- **Laden eines Photos aus der Tabelle Absolvent in eine Datei**

```
EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS BLOB_FILE           PhotoFile;
    short ind;
EXEC SQL END DECLARE SECTION;

strcpy (PhotoFile.name, “Bilder/Schnappschuss“);
PhotoFile.name_length = strlen (PhotoFile.name)
PhotoFile.file_options = SQL_FILE_OVERWRITE;

EXEC SQL
        SELECT    Photo
        INTO      :PhotoFile :ind
        FROM      Absolvent
        WHERE     Lfdnr = 17
```

  - Es sind eine Reihe von Fehlermeldungen (:ind) zu beachten
  - PhotoFile.data_length enthält anschließend die Länge der Datei
  - Austausch von potentiell sehr großen Datenmengen:
    Es ist kein Puffer für den Transfer zu allokieren!

Speicherung großer Objekte[1]

- **Darstellung großer Speicherobjekte**
  - besteht potentiell aus vielen Seiten oder Segmenten
  - ist eine uninterpretierte Bytefolge
  - Adresse (OID, *object identifier*) zeigt auf Objektkopf (*header*)
  - OID ist Stellvertreter im Satz, zu dem das **lange Feld** gehört
  - geforderte Verarbeitungsflexibilität bestimmt Zugriffs- und Speicherungsstruktur

- **Verarbeitungsprobleme**
  - Ist Objektgröße vorab bekannt?
  - Gibt es während der Lebenszeit des Objektes viele Änderungen?
  - Ist schneller sequentieller Zugriff erforderlich?
  - . . .

- **Abbildung auf Externspeicher**
  - seitenbasiert
    - Einheit der Speicherzuordnung: eine Seite
    - „verstreute“ Sammlung von Seiten
  - segmentbasiert (mehrere Seiten)
    - Segmente fester Größe (Exodus)
    - Segmente mit einem festen Wachstumsmuster (Starburst)
    - Segmente variabler Größe (EOS)
  - Zugriffsstruktur zum Objekt
    - Kettung der Segmente/Seiten
    - Liste von Einträgen (Deskriptoren)
    - B*-Baum

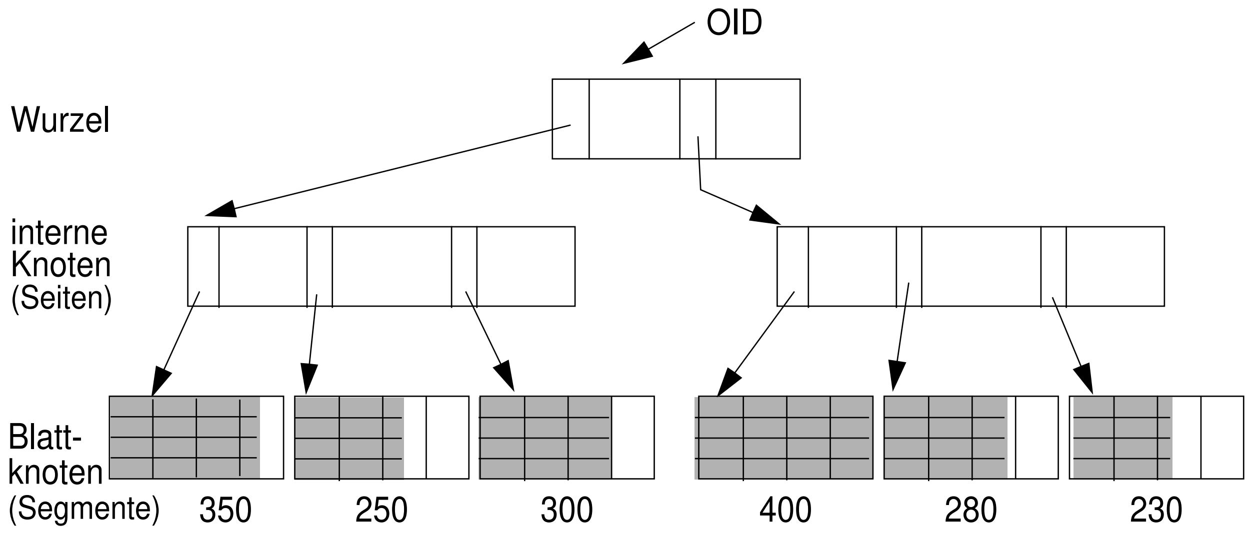
1. Biliris, A.: *The Performance of Three Database Storage Structures for Managing Large Objects*, Proc. ACM SIGMOD’92 Conf., San Diego, Calif., 1992, pp. 276-285

Lange Felder in Exodus

- **Speicherung langer Felder**
  - Daten werden in (kleinen) Segmenten fester Größe abgelegt
  - Wahl an Verarbeitungscharakteristika angepaßter Segmentgrößen
  - Einfügen von Bytefolgen einfach und überall möglich
  - schlechteres Verhalten bei sequentiellem Zugriff

- **B*-Baum als Zugriffsstruktur**
  - Blätter sind Segmente fester Größe (hier 4 Seiten zu 100 Bytes)
  - interne Knoten und Wurzel sind Index für Bytepositionen
  - interne Knoten und Wurzel speichern für jeden Kind-Knoten Einträge der Form (Seiten-#, Zähler)
    - Zähler enthält die maximale Bytenummer des jeweiligen Teilbaums (links stehende Seiteneinträge zählen zum Teilbaum).
    - Objektlänge: Zähler im weitesten rechts stehenden Eintrag der Wurzel

- **Repräsentation sehr langer dynamischer Objekte**
  - bis zu 1GB mit drei Baumebenen (selbst bei kleinen Segmenten)
  - Speicherplatznutzung typischerweise ~ 80%

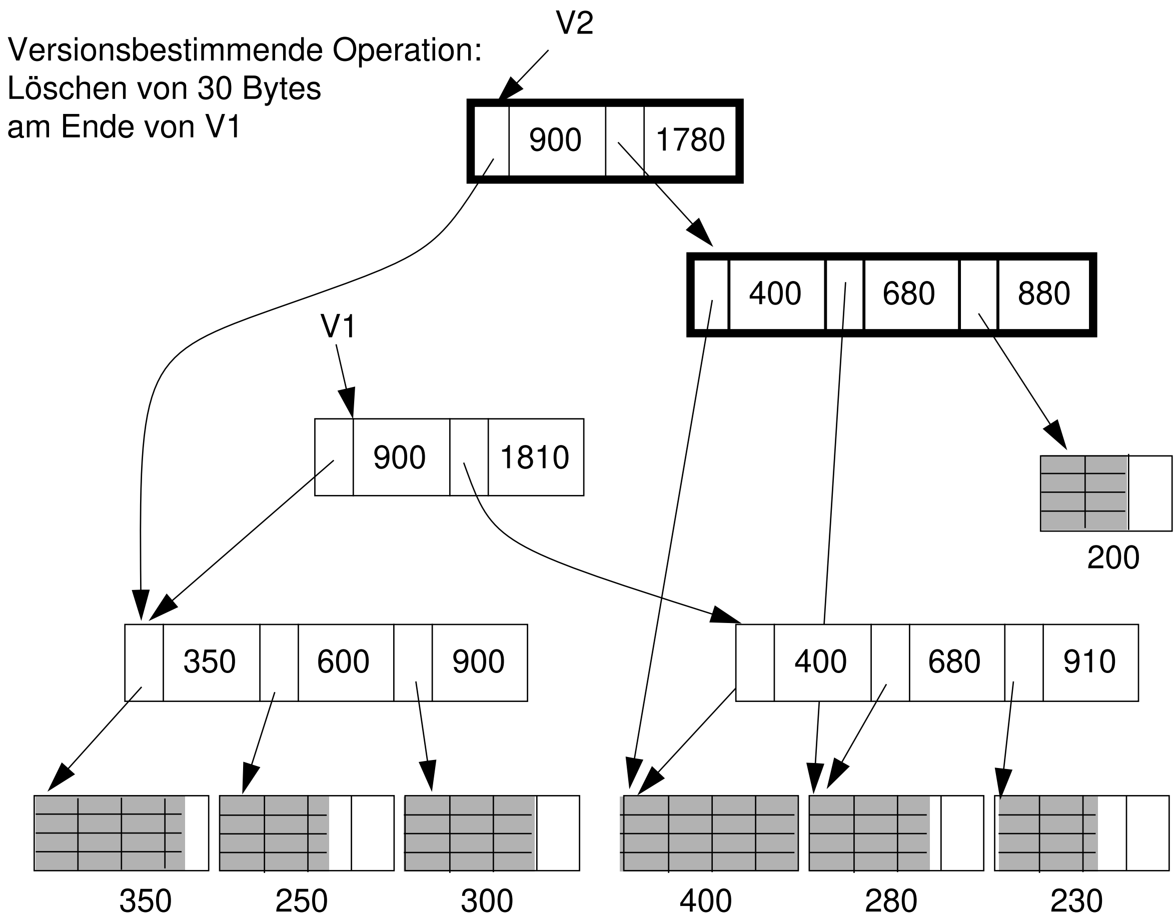


- Byte 100 in der letzten Seite?

Exodus (2)[1]

- **Spezielle Operationen**
  - Suche nach einem Byteintervall
  - Einfügen/Löschen einer Bytefolge an/von einer vorgegebenen Position
  - Anhängen einer Bytefolge ans Ende des langen Feldes

- **Unterstützung versionierter Speicherobjekte**
  - Markierung der Objekt-Header mit Versionsnummer
  - Kopieren und Ändern nur der Seiten, die sich in der neuen Version unterscheiden (in Änderungsoperationen, bei denen Versionierung eingeschaltet ist)

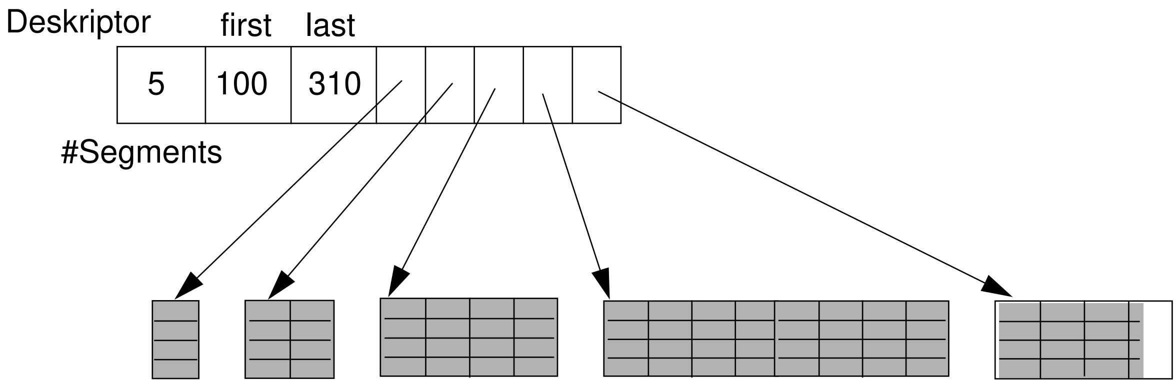


1. M.J. Carey, D.J. DeWitt, J.E. Richardson, E.J. Shekita: *Object and File Management in the EXODUS Extensible Database System*. Proc. 12th VLDB Conf., 1986, pp. 91-100

Lange Felder in Starburst

- **Erweiterte Anforderungen**
  - Effiziente Speicherallokation und -freigabe für Felder von 100 MB - 2 GB
  - hohe E/A-Leistung:
    Schreib- und Lese-Operationen sollen E/A-Raten nahe der Übertragungsgeschwindigkeit der Magnetplatte erreichen

- **Prinzipielle Repräsentation**
  - Deskriptor mit Liste der Segmentbeschreibungen
  - Langes Feld besteht aus einem oder mehreren Segmenten.
  - Segmente, auch als Buddy-Segmente bezeichnet, werden nach dem Buddy-Verfahren in großen vordefinierten Bereichen fester Länge auf Externspeicher angelegt.

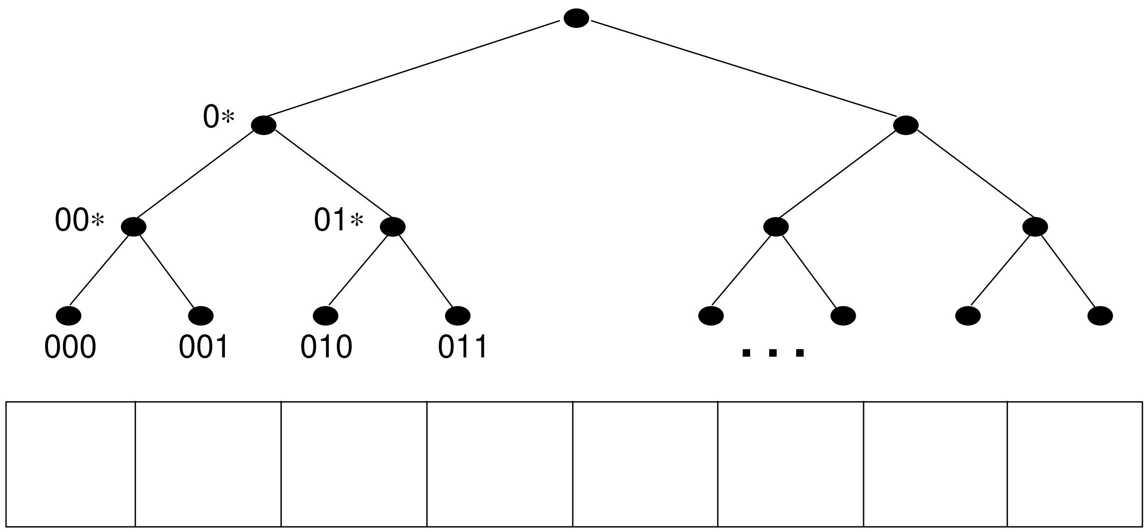


- **Segmentallokation bei vorab bekannter Objektgröße**
  - Objektgröße G (in Seiten)
  - $G \leq$ MaxSeg: es wird ein Segment angelegt
  - $G >$ MaxSeg: es wird eine Folge maximaler Segmente angelegt
  - letztes Segment wird auf verbleibende Objektgröße gekürzt

Lange Felder in Starburst [1](2)

- **Segmentallokation bei unbekannter Objektgröße**
  - Wachstumsmuster der Segmentgrößen wie im Beispiel: 1, 2, 4, ..., 2^n Seiten werden jeweils zu einem Buddy-Segment zusammengefaßt
  - MaxSeg = 2048 für n = 11
  - Falls MaxSeg erreicht wird, werden weitere Segmente der Größe MaxSeg angelegt
  - Letztes Segment wird auf die verbleibende Objektgröße gekürzt

- **Allokation von Buddy-Segmenten** in sequentiellem Buddy-Bereich gemäß binärem Buddy-Verfahren



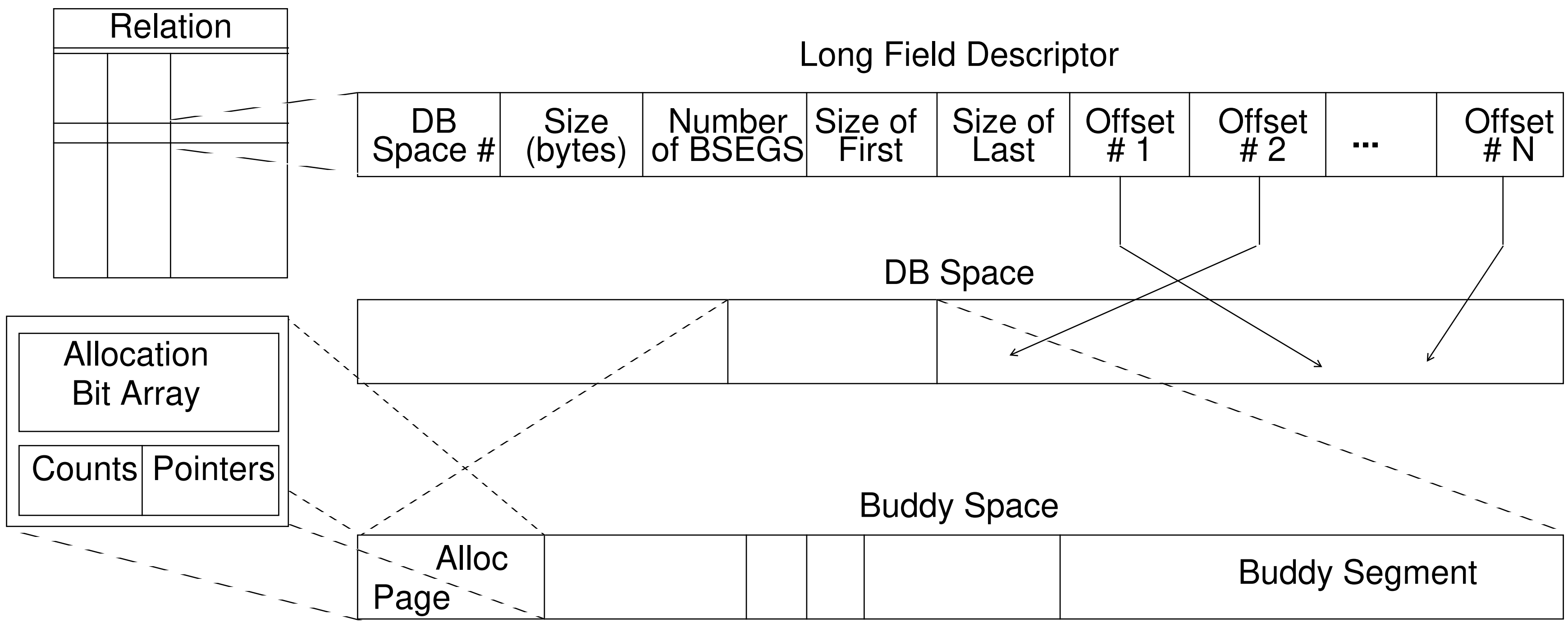
  - Zusammenfassung zweier Buddies der Größe $2^n \Rightarrow 2^{n+1}$ ($n \geq 0$)

- **Verarbeitungseigenschaften**
  - effiziente Unterstützung von sequentiellem und wahlfreiem Lesen
  - einfaches Anhängen und Entfernen von Bytefolgen am Objektende
  - schwieriges Einfügen und Löschen von Bytefolgen im Objektinnern

---

1. T.J. Lehman, B.G. Lindsay: *The Starburst Long Field Manager.* Proc. 15th VLDB Conf., 1989, pp. 375-383

Starburst: Speicherorganisation zur Realisierung Langer Felder

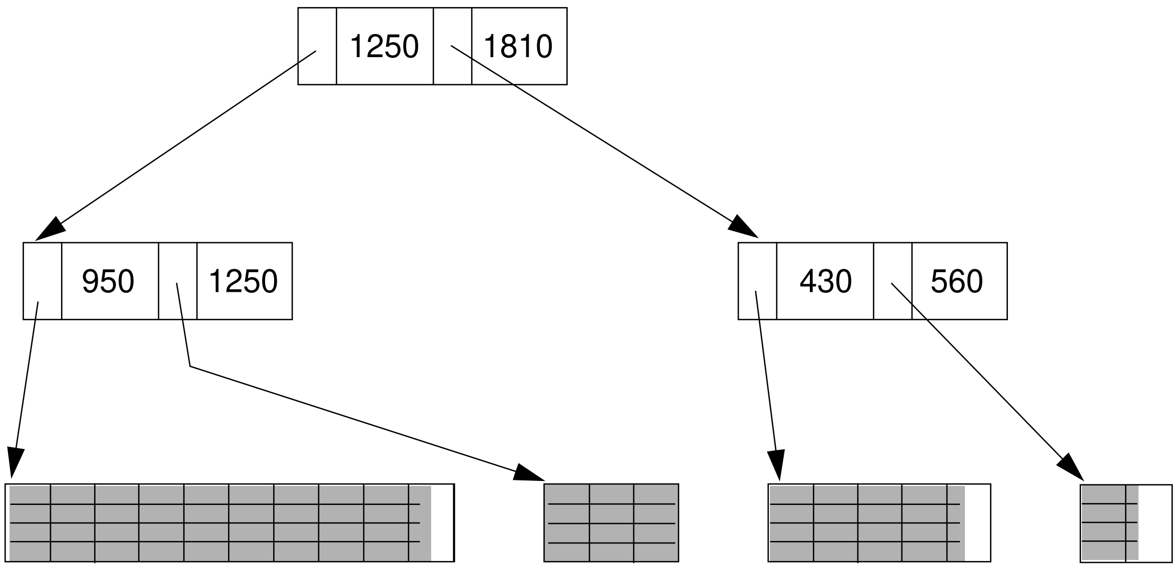


- **Aufbau eines Langen Feldes**
  - Deskriptor des Langen Feldes (< 316 Bytes) ist in Relation gespeichert
  - Long Field ist aufgebaut aus einem oder mehreren ***Buddy-Segmenten***, die in großen vordefinierten ***Buddy-Bereichen*** fester Länge auf Platte angelegt werden
  - Buddy-Segmente enthalten nur Daten und keine Kontrollinformation
  - Segment besteht aus 1, 2, 4, 8, ... oder 2048 Seiten (➥ max. Segmentgröße 2 MB bei 1 KB-Seiten)
  - Buddy-Bereiche sind allokiert in (noch größeren) DB-Dateien (DB Spaces). Sie setzen sich zusammen aus Kontrollseite (Allocation Page) und Datenbereich

Speicherallokation mit variablen Segmenten

- **Verallgemeinerung des Exodus- und Starburst-Ansatzes in Eos**
  - Objekt ist gespeichert in einer Folge von Segmenten variabler Größe
  - Segment besteht aus Seiten, die physisch zusammenhängend auf Externspeichern angeordnet sind
  - nur die letzte Seite eines Segmentes kann freien Platz aufweisen

- **Prinzipielle Repräsentation**



➥ Die Größen der einzelnen Segmente können sehr stark variieren

- **Verarbeitungseigenschaften**
  - die guten operationalen Eigenschaften der beiden zugrundeliegenden Ansätze können erzielt werden
  - Reorganisation möglich, falls benachbarte Segmente sehr klein (Seite) werden

DB-Anbindung externer Daten[1]

- **Motivation**
  - Die meisten Daten in einem Unternehmen sind in Dateien gespeichert.
  - Sie werden es auf lange Zeit bleiben und im Umfang zunehmen.
  - Da viele Anwendungen mit Dateien arbeiten, ist es erforderlich, auch diese Datenzugriffe zu unterstützen.
    (gleichförmiger Zugriff zu DBs und anderen Datenquellen, siehe OLE DB)

- **Eigenschaften**
  - Dateisysteme bieten nicht genügend Metadaten für Suchfunktionen und Integritätserhaltung.
  - DBMS unterstützen u. a. ein großes Spektrum an Funktionen, sind momentan aber nicht für die Speicherung einer großen Anzahl von BLOBs (Multimedia-Typen) optimiert.
  - BLOBs benötigen hierarchische Speicherverwaltung von leistungsfähigen Dateisystemen (z. B. Tertiärspeicher), die eine kosteneffektive Verwaltung der Daten für variierende Zugriffsmuster (häufig oder selten) gewährleisten.
  - Verknüpfung von Dateisystemen und DBMSs soll Vorteile beider Ansätze nutzen!

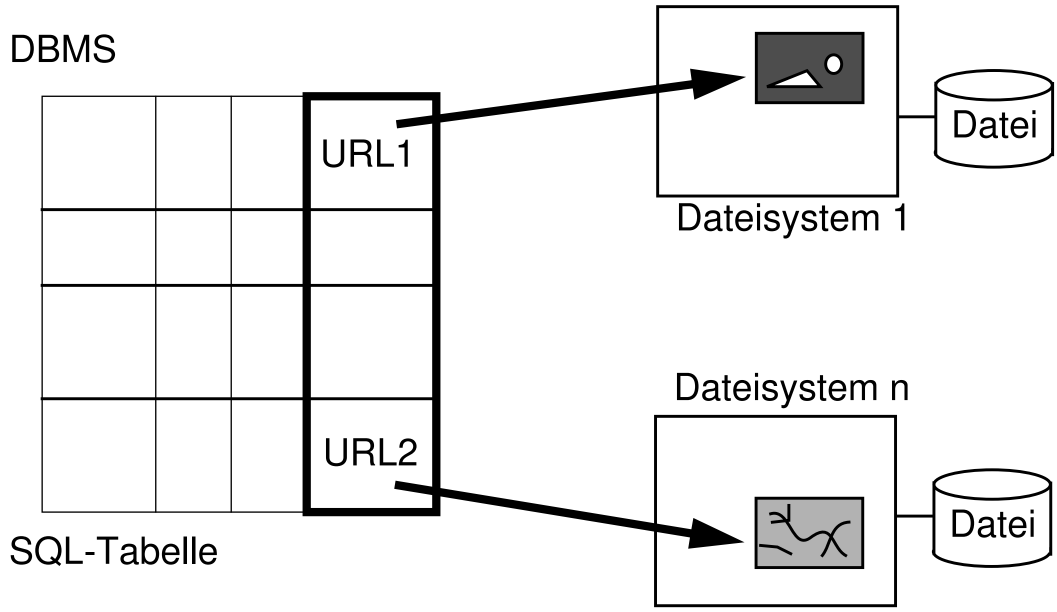
- **Anwendungsbeispiele**
  - CAD-Systeme: Synchronisation von Millionen von Bauteilen
    (Zeichnungen und Baupläne in einem proprietären Format)
  - Multimedia-Objekte: Verwaltung von Bibliotheken für Bilder, Programme, Dokumente oder Videos
  - HTML- und XML- Dateien: DB-Unterstützung für die Funktionalität von Web-Server

---

1. ISO & ANSI: Database Languages - SQL -Part 9: Management of External Data, Working Draft, September 2000

DB-Anbindung externer Daten (2)

- **Speichermodell für die DB-Anbindung**

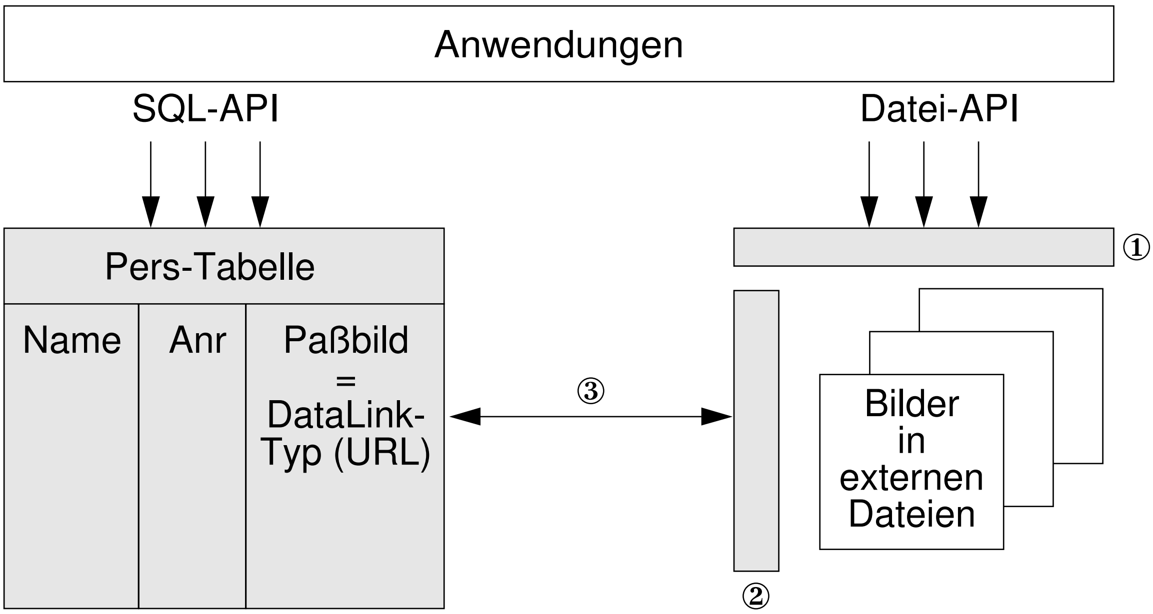


- **Welche Probleme sind zu lösen?**
  - Referentielle Integrität
  - Zugriffskontrolle
  - Koordiniertes Backup und Recovery
  - Transaktionskonsistenz
  - Suche über
    - herkömmliche Datentypen
    - Inhalte externer Daten
  - Leistungsaspekte bei DB- und Datei-Anwendungen

➥ Beteiligte Dateisysteme benötigen zusätzliche Kontrollkomponente, die mit dem DBMS über spezielle Protokolle kooperiert

DB-Anbindung externer Daten (3)

- **DataLinks-Konzept zur Verwaltung externer Daten**



① **DataLinks Filesystem Filter (DLFF)**

- erzwingt referentielle Integrität beim Umbenennen und Löschen von Dateien
- erzwingt DB-zentrierte Zugriffskontrolle beim Öffnen einer Datei
- Datei-API bleibt unverändert – keine Änderungen in den Anwendungen
- DLFF liegt nicht im Lese-/Schreib-Pfad für externe Dateien (Performance!)

② **DataLinks File Manager (DLFM)**

- führt Link-/UnLink-Operationen transaktionsgeschützt durch
- gewährleistet referenzielle Integrität
- unterstützt koordiniertes Backup/Recovery

③ **DBMS verwaltet/koordiniert Operationen auf externen Dateien**

- über URL’s referenziert
- durch DLFM-API (DataLinks File Manager)

DB-Anbindung externer Daten (4)

- **Verarbeitungsmodell aus der Sicht der Anwendung**
  - SQL-Zugriff auf Metadaten-Repository für externe Daten
  - Suche ist auch über den Inhalt externer Daten möglich
    ➥ Funktionswertindexierung
  - Liste von Referenzen der gesuchten Objekte
  - Anwendung referenziert externe Daten direkt über Datei-API.

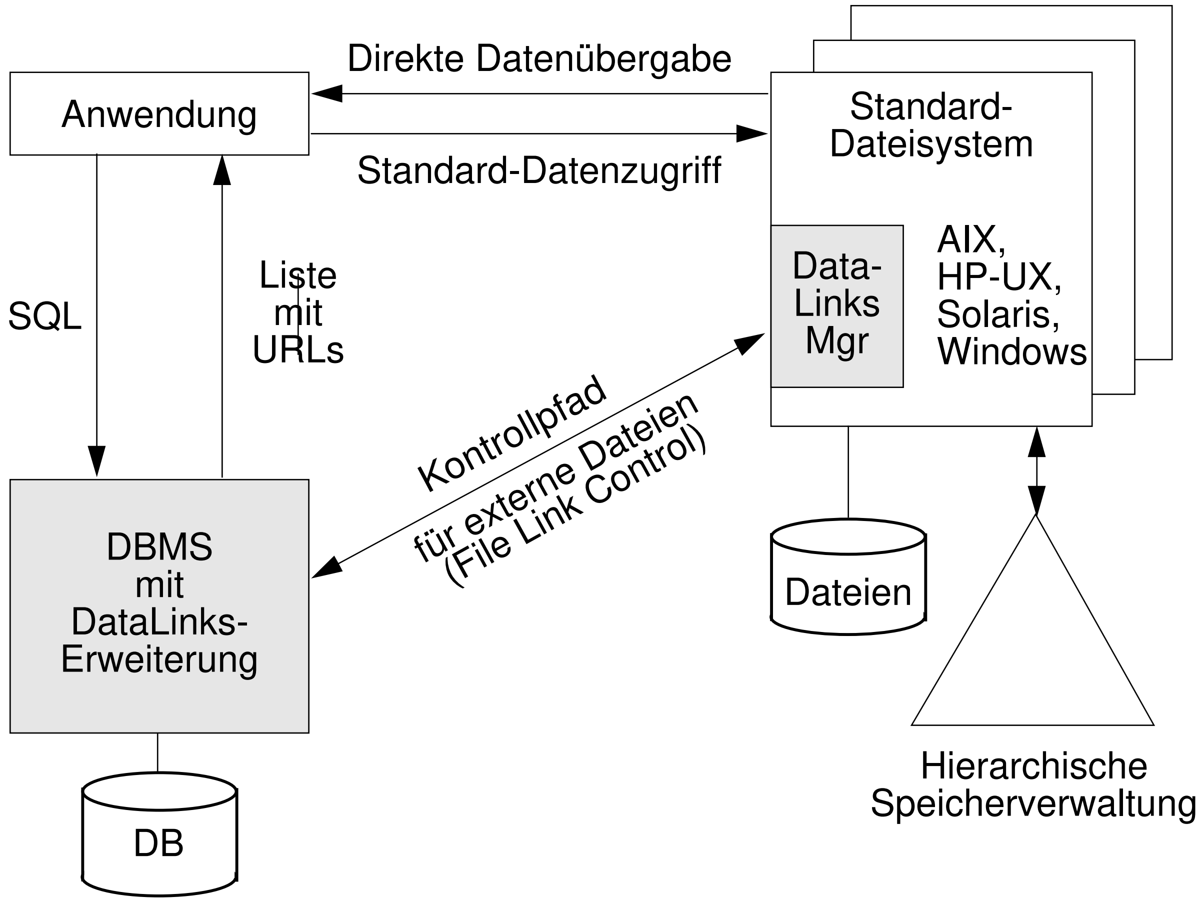
- **DataLink-Datentyp nach SQL:99 - Beispiel**

```
CREATE TABLE Pers (
  Name VARCHAR (30);
  Anr INTEGER,
  Paßbild DATALINK (200)
         LINKTYPE URL
         FILE LINK CONTROL
                    INTEGRITY all
                    READ PERMISSION DB
                    WRITE PERMISSION blocked
                    RECOVERY yes
                    ON UNLINK restore
  );
```

  - DBMS-Kontrolle läßt sich abgestuft aktivieren.
  - URL: http://servername/pathname/filename/
  - **Integrity:** URLs als Referenzen werden konsistent gehalten.
  - **Read Permission:** bleibt entweder beim Dateisystem oder wird ans DBMS delegiert. Autorisierung wird dann als Token in die URL eingebettet.
  - **Write Permission:** bleibt beim Dateisystem oder wird blockiert
  - **Recovery:** Nur bei Option WRITE PERMISSION blocked ist koordiniertes Backup und Recovery möglich.
  - **On Unlink:** Datei kann gelöscht oder zur Verwaltung ans Dateisystem zurückgegeben werden.

DB-Anbindung externer Daten (5)

- **DataLinks-Architektur**



- **Typische Anwendung**
  - Integration von unstrukturierten und semistrukturierten Daten mit Anwendungen auf DB-Basis
  - Reichweite: große Anzahl von Dateien in Rechnernetzwerken
  - Bei Funktionswertindexierung: Über URLs referenzierte Dateien bleiben weitgehend unverändert.
  - Benutzer extrahiert Features von Bildern oder Videos, speichert sie in der DB zwischen, um Auswertungen zusammen mit Prädikaten auf anderen DB-Daten zu machen.
  - *Query By Image Content* (QBIC) unterstützt Extraktion und Suche auf solchen Features.

Zusammenfassung

- **Spezifikation großer Objekte hat großen Einfluß auf die DB-Verarbeitung**
  - Speicherungsoptionen, Logging
  - Einsatz benutzerdefinierter und systemspezifischer Funktionen

- **Lokator-Konzept**
  - Identifikation von LOBs oder Positionen in LOBs
  - Minimierung des Datenverkehrs zwischen Client und Server
  - Bereitstellung von Server-Funktionen bei der LOB-Verarbeitung

- **Spezielle Verarbeitungstechniken und gute Leistungseigenschaften erforderlich**
  - Transport zur Anwendung (Minimierung von Kopiervorgängen)
  - Anfrageoptimierung, Auswertung von LOB-Funktionen
  - Synchronisation
  - Logging und Recovery
  - . . .

- **Speicherung großer Objekte wird zunehmend wichtiger**
  - B*-Baum-Technik:
    flexible Darstellung, moderate Zugriffsgeschwindigkeit
  - große variabel lange Segmente (Listen): hohe E/A-Leistung
  - Auswahl verschiedener, auf Verarbeitungscharakteristika zugeschnittener Techniken

- **DB-Anbindung für externe Dateien**
  - Die meisten Daten bleiben außerhalb der DB; jedoch ist DB-Unterstützung bei der Verwaltung, der Konsistenzerhaltung, und der inhaltsbasierten Suche wünschenswert.
  - Abgestufte Kontrollmöglichkeiten durch das DBMS
  - **DataLinks-Konzep**t bietet Referentielle Integrität, Zugriffskontrolle, Koordiniertes Backup und Recovery sowie Transaktionskonsistenz