# DBUEB – Zusammenfassung

Wanja Hofer, wanja.hofer@informatik.stud.uni-erlangen.de

SS 2005

# 0 Disclaimer

Welcome to the disclaimer![1]

Diese Zusammenfassung enthält Fakten, die ***ich*** für wichtig erachte und lässt teilweise Fakten weg, die ***ich*** für trivial halte. Bei einigen Teilen habe ich mich um eine „objektive“ Zusammenfassung bemüht.

Auf jeden Fall gebe ich keine Garantie für Vollständigkeit oder Unvollständigkeit, für Hinweise auf inhaltliche und Tippfehler bin ich dankbar.

# 1 Konzeptionelle Modellierung

## 1.1 Datenbanksysteme

DBS aus DB und DBVS (Operationen und Benutzerinteraktion).

**Vorteile gegenüber Dateisystem:**

- Zentralisierung Daten (Zugriff, Konsistenz, Aktualität)
- Zentralisierung Systemdienste
- Datenunabhängigkeit (Geräte, Formate, Pfade)
- Anfragesprache und Anwendungsneutralität (deklarativ ohne Bevorzugung)
- Mehrbenutzerbetrieb (Transaktionen)
- Recovery
- Transaktionskonzept
- Handhabbarkeit (Abstraktion von Details)

**Drei-Schema-Architektur:**

- Konzeptionelles Schema: Miniwelt anwendungsübergreifend redundanzfrei dargestellt
- Externe Schemata: durch Transformation (Sichten) für Anforderungen der Einzelanwendungen
- Interne Schemata: technische Realisierung (physisch und Hilfsstrukturen)
- Datenneutralität: extern ⇒ konzeptionell: integrierte Sicht vs. anwendungsspezifisch
- Datenunabhängigkeit: Verwendung der DB rein logisch, transparenter Wechsel des Speichers

[1] The Offspring, „Ixnay On The Hombre“

**DB-Entwurf:**

1. Analyse der Dinge und Beziehungen
2. Konzeptuelles Design mit High-Level-Modell
3. Logisches Design: Abbildung auf logisches Schema (relational, hierarchisch, Netzwerk-)
4. Physisches Design: interne Speicherstrukturen und spezifische Optimierungen

**Entity-Relationship-Diagramme (ERD):** Entity-Mengen als Rechtecke, Attribute als Ovale, Primärschlüssel unterstrichen. Relationship-Mengen als Routen, Rollennamen an Kanten. Abbildungstyp (1:1, m:n), Partizipation (partiell vs. total (muss an Relationship beteiligt sein)) an Kanten. Rekursive Relationships mit unterschiedlichen Rollen, ternäre Relationships mit mehr Infogehalt als drei binäre.
Entity = alles, was ohne Existenz anderer durch Primärschlüssel eindeutig benannt werden kann.

**Erweiterte ERD:** Klassifikation (Spezialisierung und Generalisierung) mit Dreiecken mit neuen Attributen und speziellerer Verwendung in Relationships.

**Relationenmodell:** gleichartige Objekte in derselben Relation, Unterscheidung der Ausprägungen in (zusammengesetztem) Primärschlüssel. Referenzielle Integrität mit Fremdschlüssel, damit nur Tupel referenziert werden, die vorhanden sind.

# 2 SQL – Basics und DDL, SQL*Loader

## 2.1 SQL – Basics und DDL

- Mengenorientierung, deskriptive Anfragen (was $\neq$ wie), Auswertungen
- DDL (Definition von Schemaobjekten), DML (Anfrage und Manipulation), DCL (Rechte und Rollen)

**Storage-Clause** für Extents: `STORAGE (INITIAL 100K NEXT 5K PCTINCREASE 50)`

- Tablespaces: Speicherung aller Datenstrukturen
- Segmente: Daten-, Index-, Rollback-, temporäre
- Extents: einem DB-Objekt zugeordnet, Speichergranulat zur Minimierung der Fragmentierung

**Schemata** zur inhaltlichen Zusammenfassung von Elementen, bei Oracle nur zur Kapselung mehrerer DDL-Befehle, sonst nur Speicherbereiche der Benutzer.

**implizite Datentypkonvertierung** bei `INSERT, UPDATE`, SQL-Funktionen und Vergleichsoperator

**Integritätsbedingungen** schränken Werte einer einzelnen Relation ein, Zurückweisung, falls nicht konforme Tupel vorhanden. Sprechende Namen für Fehlermeldungen und Löschen. Column-Constraints (`PRIMARY KEY, REFERENCES`), Table-Constraints (zusammengesetzte Primärschlüssel).
`CONSTRAINT bla_con CHECK (att1 > 10)`

**Ändern** mit `ADD`, `MODIFY` (Typ, Default-Wert, `NOT NULL`)

**Löschen** mit `CASCADE CONSTRAINTS` werden alle externen Integritätsbedingungen, die auf die Relation verweisen, auch gelöscht.

### 2.2 SQL*Loader

Einlesen aus externen Dateien, parametrisierbar ($\neq$ Import/Export von/nach anderen Oracle-DBs). Erzeugt werden Logfile und evtl. Badfile.

**Steuerdatei:**

```
LOAD DATA
INFILE 'inputfile' // * fuer Einlesen nach BEGINDATA
REPLACE INTO TABLE bla_tab // APPEND zum Anhaengen
TRAILING NULLCOLS
(
    att1 POSITION (5:10) INTEGER EXTERNAL,
    att2 POSITION (*+3)  CHAR TERMINATED BY '|' // 3 Zeichen ueberspringen
)
```

**Homogene Struktur:**

```
FIELDS TERMINATED BY ','
(att1, att2)
```

# 3 SQL – Abfrage

- Data Dictionary (Catalog) mit Views über Objekte in der DB
- Tupelvariable für langen Relationennamen
- `LIKE` mit % und _
- Potenzierung mit `**`
- Relationenalgebra mit Selektion $\sigma$, Projektion $\pi$, kartesischem Produkt $\times$, Verbund $\bowtie$, Mengen- und Aggregatfunktionen
- nichtkorrelierte Unterabfragen werden nur einmal berechnet
- Reihenfolge: `FROM, WHERE, GROUP BY und AGG, HAVING, SELECT, ORDER BY`
- Datum: `TO_CHAR (time, 'HH24:MI')`, $*1440$ = Minuten

# 4 Transaktionen, DML, Tricky SQL

Transaktionen für Überführung zwischen konsistenten Zuständen (ACID) per `COMMIT`, Savepoints für partielle Rollbacks.

```
INSERT INTO bla_tab VALUES (1, 2);
UPDATE bla_tab SET (att1, att2) = ...;
DELETE FROM bla_tab WHERE ...;
```

**Tricky:** wenn Nutzung einer Ordnung, da Relationenmodell mengen- und nicht sequenzorientiert.

- evtl. zweimal mit selber Relation joinen (`ON a.gehalt <= b.gehalt`)
- geschicktes Gruppieren

# 5 Views, Sequenzen, Synonyme und PL/SQL

## 5.1 Views, Sequenzen und Synonyme

Dynamisches Fenster, zugeschnitten auf Bedarf und Rechte, mit `FORCE` über noch nicht existierende Relationen. Einfache (≠ komplexe) Sichten: 1 Basistabelle, keine Funktionen, keine Gruppierung, alle DML erlaubt. Einschränkung von:

- `DELETE`: keine Gruppierung, Aggregate, `DISTINCT`
- `UPDATE`: + Ausdrucksberechnung
- `INSERT`: + Attribut, das nicht in Sicht ist, `NOT NULL`

`WITH CHECK OPTION CONSTRAINT bla`: Auch nach Definition nur korrekte Einfügung von Tupeln in Sicht.

**Sequenzen** für Surrogat-Primärschlüssel mit Performanzvorteil (Cache). Synonyme für fremde Schematabellen.

```
CREATE SEQUENCE bla_seq;
INSERT INTO bla_tab VALUES (bla_seq.NEXTVAL);
// aktueller Wert: bla_seq.CURVAL

CREATE SYNONYM my_bla FOR scott.bla;
```

## 5.2 PL/SQL

Oracle-eigene OO Erweiterung von SQL.
Cursor: `CURSOR c (par NUMBER) IS SELECT att1 FROM bla_tab WHERE att2 = par;`
Arrays: `TYPE bla_arr IS TABLE OF NUMBER;`, erstes Element: `bla_arr (1) := 4711;`
Structs: `TYPE bla_rec IS RECORD ( a1 NUMBER, ...);`
Cursor-Schleife:

```
FOR x IN c LOOP
        DBMS_OUTPUT.PUT_LINE ('Var: ' || x.att1);
END LOOP;
```

Exceptions: Deklarieren, Raisen, Auffangen
Prozeduren / Funktionen für Wartbarkeit, Wiederverwendung und Erweiterbarkeit:
`PROCEDURE bla_proc (par IN NUMBER, ...) IS ...;`
Stored Procedures direkt in DB, vorkompiliert: `CREATE PROCEDURE bla_stored ...;`

# 6 Trigger und Packages

## 6.1 Trigger

```
CREATE TRIGGER bla_bdr
BEFORE DELETE ON bla_tab FOR EACH ROW
WHEN (new.value > 50)
DECLARE
        ...
BEGIN
        if (:new.value - :old.value > 10) then raise_application_error (-20501, 'Nein!');
        END IF;
END;
```

- Event (`INSERT, DELETE, UPDATE`), Condition, Action (SQL und PL/SQL)
- Row Trigger (`FOR EACH ROW`, bei Abhängigkeit von Parametern) – Statement Trigger
- Before Trigger (Verhinderung (kein Rücksetzen nötig), Speicherung von Attributwerten) – After Trigger (weitere Aktionen)

### 6.2 Packages

Vorteile:

- Leistung (komplett im Speicher)
- Verfügbarkeit (über Benutzer und Sitzungen hinweg)
- Information Hiding
- Transport
- Modularität (abhängige Funktionen zusammen)

```
CREATE PACKAGE bla_pack AS
        PROCEDURE ...;
        FUNCTION ... RETURN ...;
END bla_pack;

CREATE PACKAGE BODY bla_pack AS
        PROCEDURE ... IS
                ...
        BEGIN
                ...
        END ...;
END bla_pack;
```

### 6.3 Diverses in PL/SQL

```
DECLARE
        t1 bla_tab.att1%TYPE;
        t2 bla_tab%ROWTYPE;
BEGIN
        SELECT att1 INTO t1 FROM bla_tab WHERE ...;
        SELECT * INTO t2 FROM bla_tab WHERE ...;
```

Aggregatfunktionen ignorieren NULL-Werte, Konvertierung in andere Zeichenketten / Zahlen:
`SELECT AVG (NVL (att1_tab, 0)) FROM bla_tab;`

## 7 OO

Vorteile:

- direkte Abbildung
- datennahe Funktionen
- Effizienz: 1 Round-Trip für zusammengehörige Objekte über Referenzen
- Modellierung Teil-Ganze-Beziehung

```
CREATE TYPE bla AS OBJECT (
        blubb NUMBER;
        [OVERRIDING] MEMBER PROCEDURE dostuff,
        [OVERRIDING] MEMBER FUNCTION returnstuff RETURN NUMBER
) NOT INSTANTIABLE, NOT FINAL;

CREATE TYPE BODY bla AS
        MEMBER PROCEDURE dostuff IS
        ...
        SELF.blubb;
        ...
        END dostuff;
END;

CREATE TYPE blachild UNDER bla ();
```

Speicherung von Methoden (Member-, statische und Konstruktur-) in DB, falls in PL/SQL oder Java.
Objekttabellen: `CREATE TABLE bla_tab OF bla (blubb PRIMARY KEY);`, Interpretation als ein- oder mehrspaltige Tabelle.
Extrahieren: `SELECT VALUE (b) FROM bla_tab b WHERE ...;`

`REF`-Datentyp als logischer Zeiger, m:n-Beziehungen ohne Fremdschlüssel
Extrahieren: `SELECT REF (b) FROM bla_tab b WHERE ...;`
Beim Insert: `INSERT INTO bla_tab b VALUES (bla (4711)) RETURNING REF (b) INTO b_ref;`
Dereferenzieren: `SELECT DEREF (b_ref) INTO b_obj FROM dual;` (in SQL implizit mit dot-Notation)
Typ-Bestimmung: `SELECT * FROM bla_tab b WHERE VALUE (b) IS OF (blachild);`

Nested Tables: ungeordnete Menge:
`CREATE TYPE nest AS TABLE OF bla;`
, bei Attribut-Verwendung explizite Speicherung in separater Tabelle:
`CREATE TABLE tab (mynest nest) NESTED TABLE mynest STORE AS mynest_tab;`
Varray: geordnete Menge:
`CREATE TYPE varr AS VARRAY (10) OF bla;`
, flache Anzeige:
`SELECT v.* FROM tab t, TABLE (t.varr) v;`

# 8 JDBC / SQLJ

## 8.1 JDBC

- plattformunabhängige Datenbank-API (Ziel und Quelle)
- Two-Tier: direkte Client-Server-Kommunikation
- Three-Tier: GUI, Anwendungslogik, DB – Middle Tier: Zugriffskontrolle, Übersetzung High-Level- in Low-Level-API
- Treiber:
  1. JDBC-ODBC-Bridge: keine herstellerspezifischen Treiber, aber ODBC-Treiber nötig
  2. Datenbankproprietäres API: JDBC-Call ⇒ OCI-Call, Code bei Client nötig
  3. JDBC-Netzwerkprotokoll: JDBC-Call ⇒ DB-unabhängige Netzwerkaufrufe an Middleware-Server
  4. Datenbankproprietäres Netzwerkprotokoll: JDBC ⇒ SQL*Net-Call

## 8.2 JDBC-API

```
Connection conn = DriverManager.getConnection ("jdbc:oracle:thin:@host:port:DB",
"user", "pw");

conn.getMetaData ();

Statement stmt = conn.createStatement ();
stmt.executeQuery ("select * from dual");
stmt.executeUpdate ("insert into ...");

// Uebersetzung bei der Objekterzeugung
PreparedStatement stmt = conn.prepareStatement ("update bla where x = ?;);
stmt.setInt (1, 4711);

// Aufruf von Stored Procedures
CallableStatement

ResultSet r = stmt.executeQuery ();
while (r.next ()) {
        getInt (1); // oder getInt ("Nummer");
}

...
} catch (SQLException e) {}
```

## 8.3 SQLJ

- statisch, weniger flexibel, Vorübersetzer
- declaration statements (Iteratoren)
  - named iterator
  - positioned iterator
- execution statements in {}
  - assignment clauses mit Ausgaben
  - statement clauses ohne Ausgaben

Vorteile:

- höhere logische Ebene (Klartext $\neq$ Methodenaufrufe)
- einfachere Syntax
- syntaktische und semantische Analyse zur Übersetzungszeit ($\Rightarrow$ schneller, weniger Fehler)