Inhalt

1. Einführung

2. Wozu Datenbankanbindung
2.1 Einbindung von unternehmensinternen Daten
2.2 Einbindung vorhandener Daten in das WWW
2.3 Entwicklung von stark interaktiven Anwendungen

3. Wie wird die Datenbankanbindung realisiert?

3.1 Serverseitige Applikationslogik
3.1.1 Einstufige Verfahren
3.1.1.1 Verarbeitung mit Hilfe von CGI-Bin's
3.1.1.2 Verwendung von Server-API's
3.1.2 Zweistufige Lösung
3.1.3 Funktionsabarbeitung im Datenbank-Server

3.2 Client-seitige Applikationslogik
3.2.1 Positive Aspekte beim Einsatz von Java
3.2.2 Negative Aspekte beim Einsatz von Java
3.2.3 Datenbankanbindung in Java mittels JDBC

3.3 Benutzung von SQL3
3.3.1 Entwicklungsmöglichkeiten durch neue Aspekte von SQL3
3.3.2 Einbindung der Neuerung zur Aktualisierung von Web-Seiten

4. Zusammenfassung

2. Wozu Datenbankanbindung?

- Einbindung von unternehmensinternen Daten mit Hilfe der Technologie des Internets – Aufbau eines sog. Intranets
  - Plattform-Unabhängigkeit
  - Einfache Administration
  - Benutzerfreundlichkeit
  - Einfache Anbindung nach Außen

- Externe Bereitstellung vorhandener Daten im WWW
  - Proprietäres Dateiformat
  - Regelmäßige Konvertierung
  - vs. On-the-Fly (bei der Benutzeranfrage)
  - vs. Triggerbenutzung (SQL3)

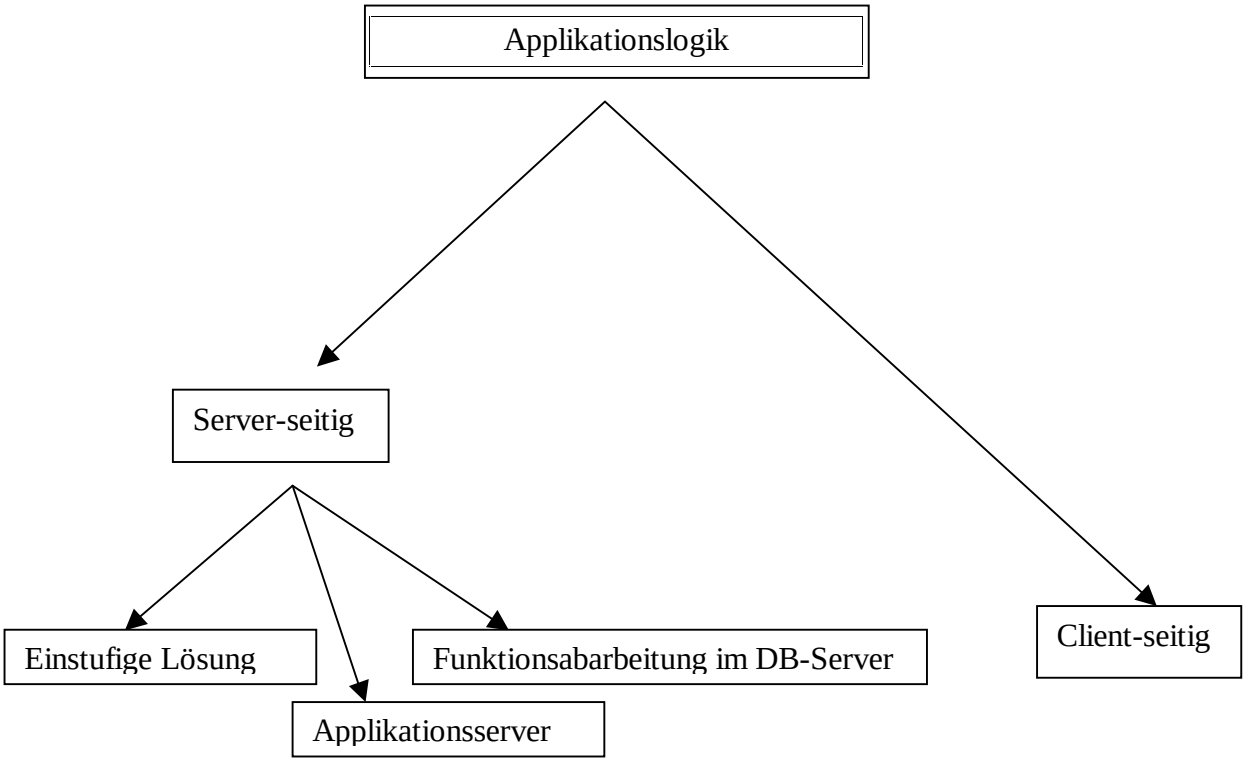
- Erstellung von stark interaktiven Anwendungen, z. B. Online-Einkaufszentren oder –Auskünften
  - Interaktionsmöglichkeiten gehen weit über das Hindurchnavigieren mit Hilfe von Links hinaus
  - Eigenhändige Eingaben mit Hilfe von Formularen möglich
  - Die Sitzungen erstrecken sich über mehrere Schritte, bei denen immer wieder Eingaben vom Benutzer erforderlich sind.

2. Realisierungsansätze der Datenbankanbindung

2.1 Übersicht

2.1.1. Dynamische Verfahren

⇒ hohe Serverlast; lange Antwortzeiten
⇒ Seiten immer aktuell
⇒ Reaktion auf Benutzereingaben (Formulare) möglich
⇒ Kap. 2.2 und 2.3



2.1.2 Regelmäßige Konvertierung

⇒ Daten nicht immer aktuell

2.1.3 Benutzung von neuen Methoden (SQL3)

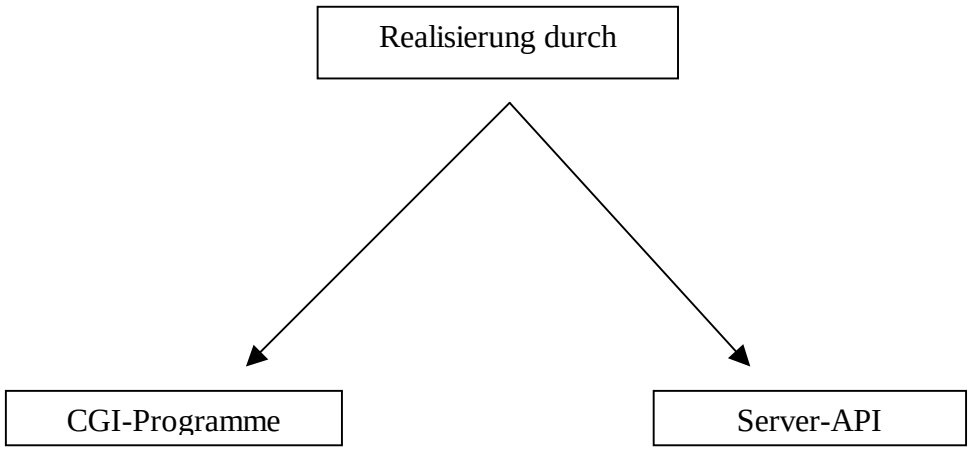
⇒ Kap. 2.4

2.2 Serverseitige Applikationslogik

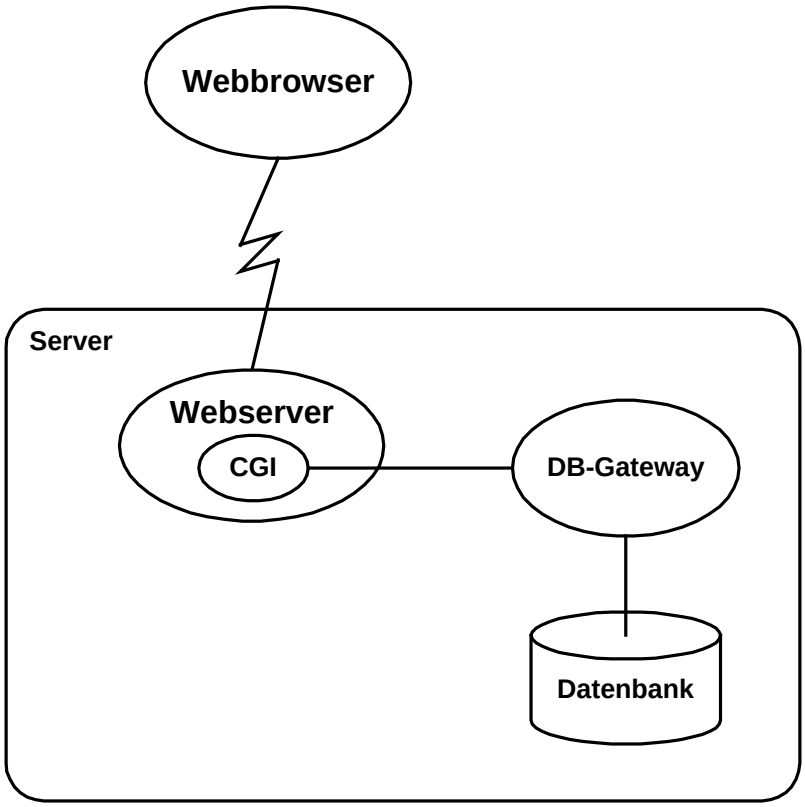
+ "thin Client" (schlanker Browser: keine Übertragung von Programmdaten in den Browser nötig)
+ kürzere Ladezeit
- Zustandslosigkeit von HTTP

2.2.1 Einstufige Lösung

- Einfacher Aufbau
- Direkte (logische) DB-Verbindung
- Hoher Ressourcenverbrauch
- Verwendbar bei niedrigem Besucher-aufkommen und/oder einfachen Anwendungen
- Beispiele: Gästebuch, Paketverfolgung



2.2.1.1 CGI-Bin's



+ Gateway separater Prozeß, teilt keine Ressourcen mit WS
+ einfache Implementierung und Erweiterung des Gateways
+ Kompatibel mit jedem CGI-konformen Web-Server

- Hoher Aufwand durch:
  - Starten und Beenden der Prozesse
  - relativ viele Systemressourcen
  - für jede Benutzeranfrage eine Kopie des Gateway-Prozesses
  - wiederholtes Aufbauen und Schließen einer DB-verbindung mit jeder Benutzeranfrage

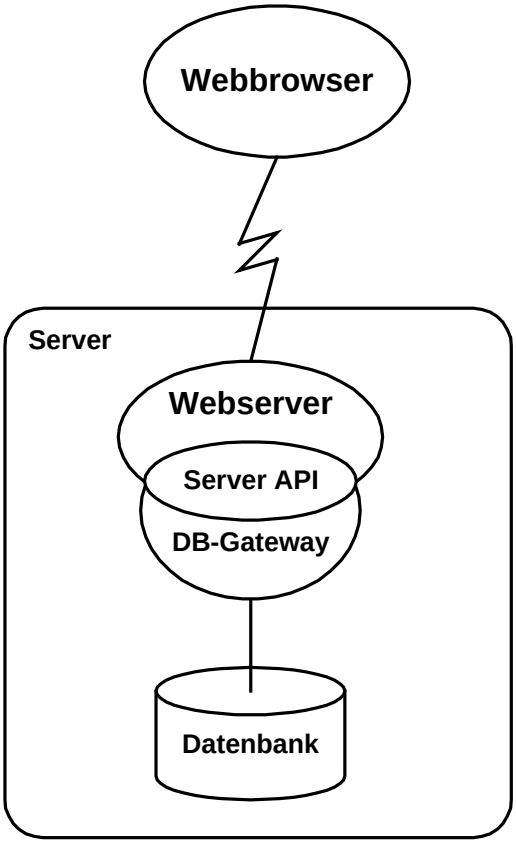
2.2.1.2 Server-API

proprietäre Programmierschnittstellen (APIs)

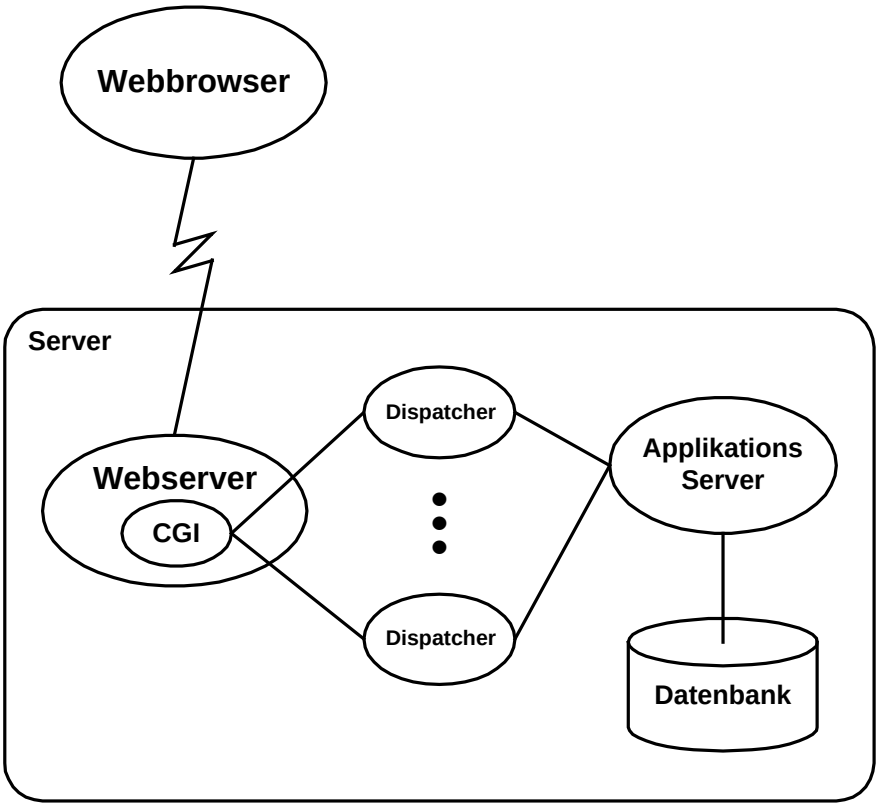
API	Hersteller	Web-Server
NSAPI	Netscape	Fast Track Server, Enterprise Server
ISAPI	Microsoft	Internet Information Server
WSAPI	O'Reilley	WebSite Pro
Apache API	Apache Group	Apache

Ressourcenteilung mit Serverprozeß

+ können in jede Phase der Serveroperation (URL-Interpretation ... Logging) eingreifen.
+ Starten und Beenden des Prozesses entfällt
+ Verbindung zur Datenbank kann aufrechterhalten werden
- proprietär, dadurch abhängig von bestimmten Web-Server-Herstellern und Produkten



2.2.2 Applikationsserver

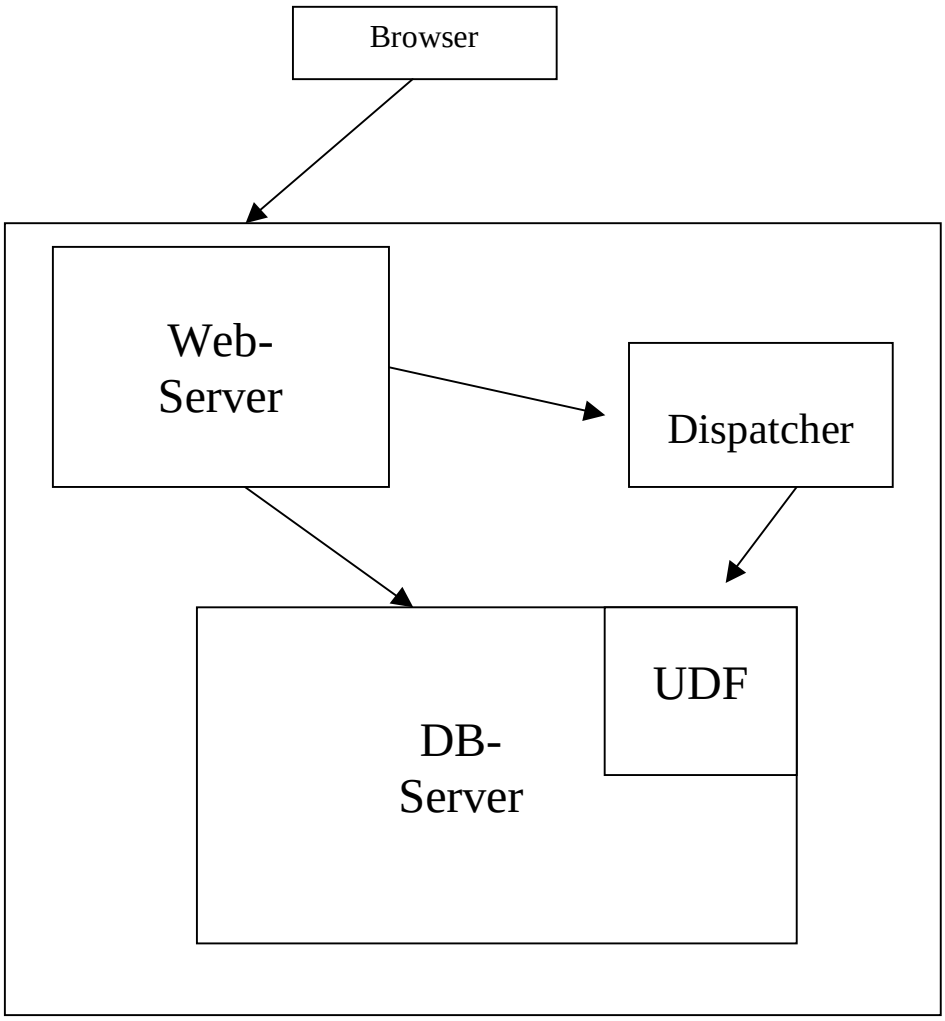


Dispatcher nimmt Benutzeranfragen entgegen und übergibt diese an den Applikationsserver, der im Hintergrund läuft. Dieser enthält die Applikationslogik und steht mit DB-Server in Verbindung. Antworten werden über D. an den Benutzer zurück geschickt.

+ CGI-Konformität
+ eigener Adreßraum
+ kurze Antwortzeit (kl. Programmgröße d. Dispatcher)
+ Aufrechterhaltung der DB-Verbindung
+ flexible Lastverteilung

- komplexe Programmierung
- evtl. hoher Kommunikations-Overhead

2.2.3 Funktionsabarbeitung im DB-Server



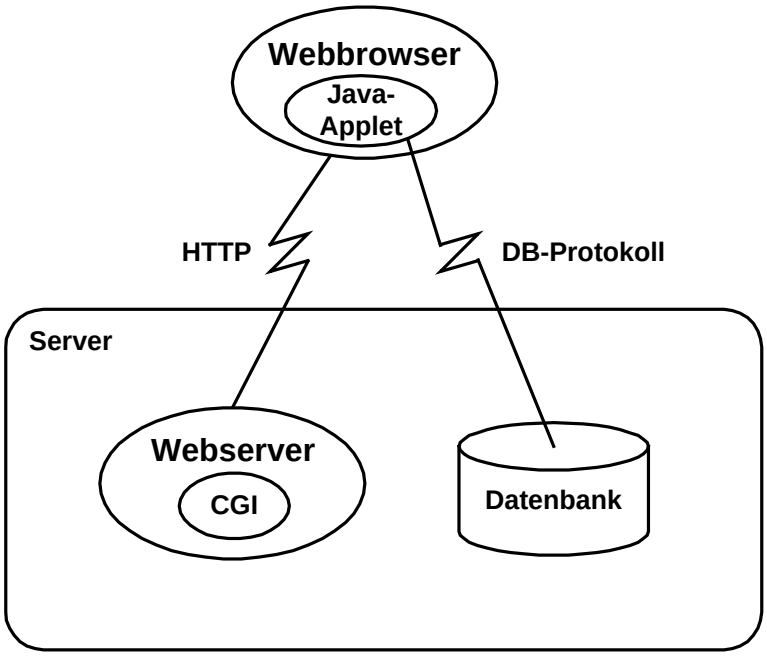
- Ausnutzung der Erweiterungsfähigkeit moderner ORDBMS
- Benutzerdefinierte Funktionen werden direkt im DB-Server ausgeführt

\+ geringere Kommunikation (Ergebnis ist lediglich ein String, der direkt zum Benutzer übertragen wird)
\+ kein seperater Prozeß f. Applikationslogik
\- Mehrbelastung des DB-Servers
\- geringere Flexibilität
\- schlechtere Lastverteilung
\- aufwendigere Programmentwicklung

2.3 Client-seitige Applikationslogik

- Java-Applets / Plug-In's
- Erweiterung der Browser-Funktionalität
- Applikationslogik im Client
- Plug-In's:
  - Browser wird nur zum Starten des Zusatzprogramms gebraucht
  - - : nicht plattformunabhängig
  - ⇒ meist Einsatz von Java

2.3.1 Einsatz von Java



2.3.1.1 Positive und negative Aspekte beim Einsatz von Java

+ keine Programminstallation auf Client-Seite
+ zentrale Administration
+ Umweg über Web-Server wird vermieden
+ geringere Kommunikation
+ flexiblere GUI-Gestaltung
+ Zustände hier realisierbar
+ "lange" Transaktionen möglich
- lange initiale Ladezeit (Programmdaten müssen zum Browser übertragen werden)
- eingeschränkte Rechte der Applets ("Sandbox")
- o. g. Probleme können durch signierte Applets umgangen werden
- Sicherheit evtl. durch Analyse der Applets gefährdet

2.3.1.2 JDBC

- " Java Database Connectivity"
- Klassenbibliotheken, die Schnittstellen zur Datenbankanbindung definieren
- Seit Version 1.1 in Standard-Java enthalten.
- wird ständig erweitert

+ unabhängig von DB-Systemen (hoher Programmierlevel)
+ hohe Mächtigkeit

2.3.1.2 JDBC (Fortsetzung)

JDBC-Treiber-Typ	*Beschreibung*
Typ 1 (JDBC-ODBC-Bridge)	• Dient zur Anbindung über vorhandene ODBC-Treiber • Benötigt nativen Binärcode
Typ 2 (Native-API-partly-Java)	• Wandelt JDBC-Aufrufe in solche für proprietäre Client-API's • Benötigt ebenfalls nativen Binärcode
Typ 3 (Net-Protocol-All-Java)	• Übersetzt JBDC-Aufrufe in DBVS-unabhängiges Netzwerkprotokoll für Middleware • Middleware übersetzt NW-Aufrufe und verbindet Client mit dem DBVS
Typ 4 (Native-Protocol-All-Java)	• Übersetzt JBDC-Aufrufe in DBVS-spezifisches Protokoll • Direkte Verbindung zwischen Client und DBVS

- Typ 1 und 2 benötigen Binärcode ⇒ kommen für Realisierung von Web-DB-Anwendungen nicht in Frage
- Typ 3 benutzt Verbindungsserver zur Kommunikation
  - flexibelste Lösung
  - erhöhte Sicherheit (keine direkte Kommunikation mit DB-Server)
- Typ 4: direkte Verbindung zum DB-Server ⇒ kürzere Antwortzeiten
- Aber: geringe Sicherheit! / schlechte Lastverteilung bei unsignierten Applets

2.4 Benutzung von SQL3, um Web-Seiten aktuell zu halten

- Benutzung neuer Möglichkeiten moderner ORDBS (und SQL3)
- Inhalte der Seiten immer aktuell
- Geringe Last
- Allerdings: keine dynamische Antwort auf Benutzenanfragen (Formulare)
- Methode gut verwendbar für Indexseiten, Nachrichtenticker...

- Trigger stoßen bei Änderungsoperationen auf DB-Tabellen UDF an:

```
CREATE TRIGGER docGenerate AFTER INSERT ON news
REFERRENCING NEW AS n (EXECUTE PROCEDURE
generateDoc(n.id, 'table=news'));
```

- Hängt Dokument von mehr als einer Tabelle ab, so muß für je eine solche Tabelle ein Trigger definiert sein
- UDF generiert aus Tabellen und im DB-System gespeicherten Formatvorlagen HTML-Dokumente und schreibt diese als Datei in das Dateisystem

3. Zusammenfassung

- Wozu Datenbankanbindung?
  - Einbindung von unternehmensinternen Daten
  - Externe Bereitstellung vorhandener Daten im WWW
  - Erstellung von stark interaktiven Anwendungen

- Realisierungsansätze
  - Regelmäßige Konvertierung
    - + Wenig Ressourcenverbrauch
    - - Web-Seiten nicht immer aktuell
    - - keine Reaktion auf Benutzereingaben

  - Dynamische Erzeugung
    - + Reaktion auf Benutzereingaben möglich
    - + Aktueller Inhalt der Webseiten
    - - längere Antwortzeit
    - Server-seitige Applikationslogik
      - Einstufige Verfahren
        - CGI-BIN's
        - Server-API
      - Applikationsserver
      - Funktionsabarbeitung im DB-Server
    - Client-seitige Applikationslogik
      - Java-Applets
      - $\Rightarrow$ JDBC

  - Benutzung von SQL3
    - + Aktueller Inhalt
    - + kürzere Antwortzeiten
    - - keine Reaktion auf Benutzereingaben
    - Einsetzbar für Newsticker, Indexseiten...