

# Daten manipulieren

[illegible]

# Verwendung der Unterlagen



- Bitte beachte, dass die vorliegenden Unterlagen als Begleitmaterial für die Schulung erstellt worden sind. Sie sind daher nur eingeschränkt zum Selbststudium geeignet.
- Die meisten der vorgestellten Konzepte, Anweisungen, Syntaxdiagramme u.ä. bieten noch weitere als die hier vorgestellten Möglichkeiten. Die Darstellung ist im Wesentlichen auf den Umfang der Grundlagenschulung eingeschränkt.

# Voraussetzungen

- Grundkenntnisse Datenbanken (z.B. aus Teil 1 der Schulung)
- Grundkenntnisse SQL Abfragen (SELECT-Anweisung – z.B. aus Teil 2 der Schulung)
- Für die Übungsdatenbank: Mac/PC mit aktueller Java-Version

# Übersicht

- Data Manipulation Language (DML)
- Daten eingeben mit **INSERT**
- Übungen zu **INSERT**
- Transaktionen
- Daten ändern mit **UPDATE**
- Daten löschen mit **DELETE**
- Übungen zu DML

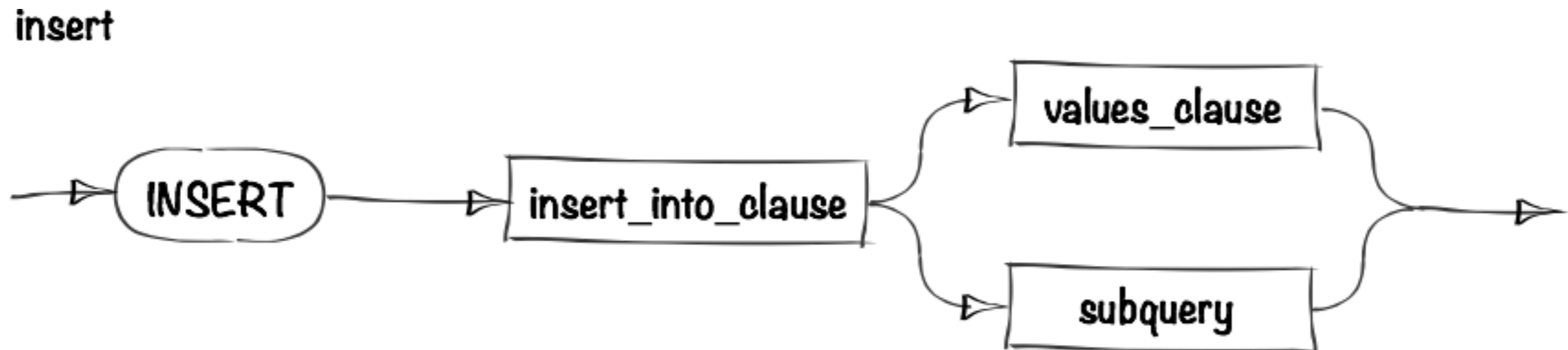
# Data Manipulation Language (DML)

- Eine DML-Anweisung wird ausgeführt, wenn
  - neue Zeilen in eine Tabelle eingefügt werden (**INSERT**)
  - vorhandene Zeilen in einer Tabelle geändert werden (**UPDATE**)
  - vorhandene Zeilen aus einer Tabelle entfernt werden (**DELETE**)

# Daten einfügen mit INSERT

- Mit dem **INSERT** Befehl werden Zeilen in eine Tabelle eingefügt.
- Es können sowohl einzelne Zeilen als auch mehrere Zeilen in einer Anweisung eingefügt werden.
- Die definierten Konsistenzbedingungen werden geprüft (wird in Data Definition Language näher erläutert).

# Daten einfügen mit INSERT – Syntax (1)



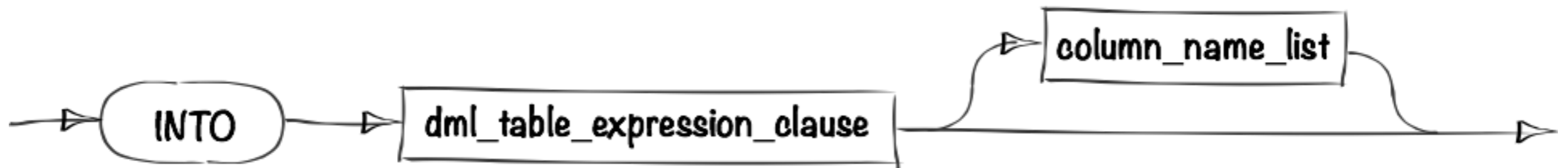
- Der Platzhalter *subquery* entspricht dem bereits eingeführten Syntaxkonstrukt dieses Namens (**SELECT**-Anweisung)

# Daten einfügen mit INSERT – Syntax (2)

insert



insert\_into\_clause



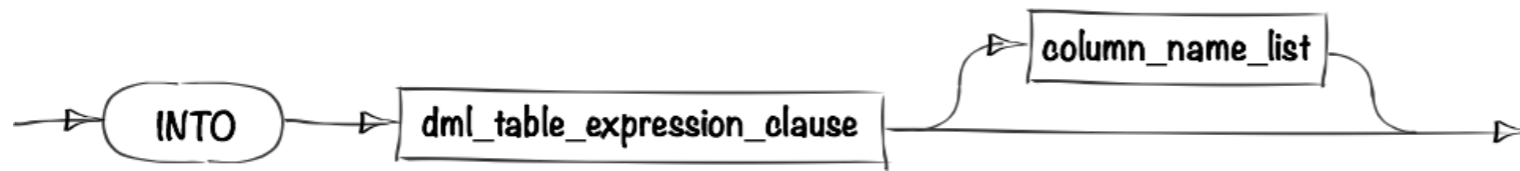
- *dml\_table\_expression\_clause* wird vorerst lediglich in der Form von einfachen Tabellennamen verwendet

# Daten einfügen mit INSERT – Syntax (3)

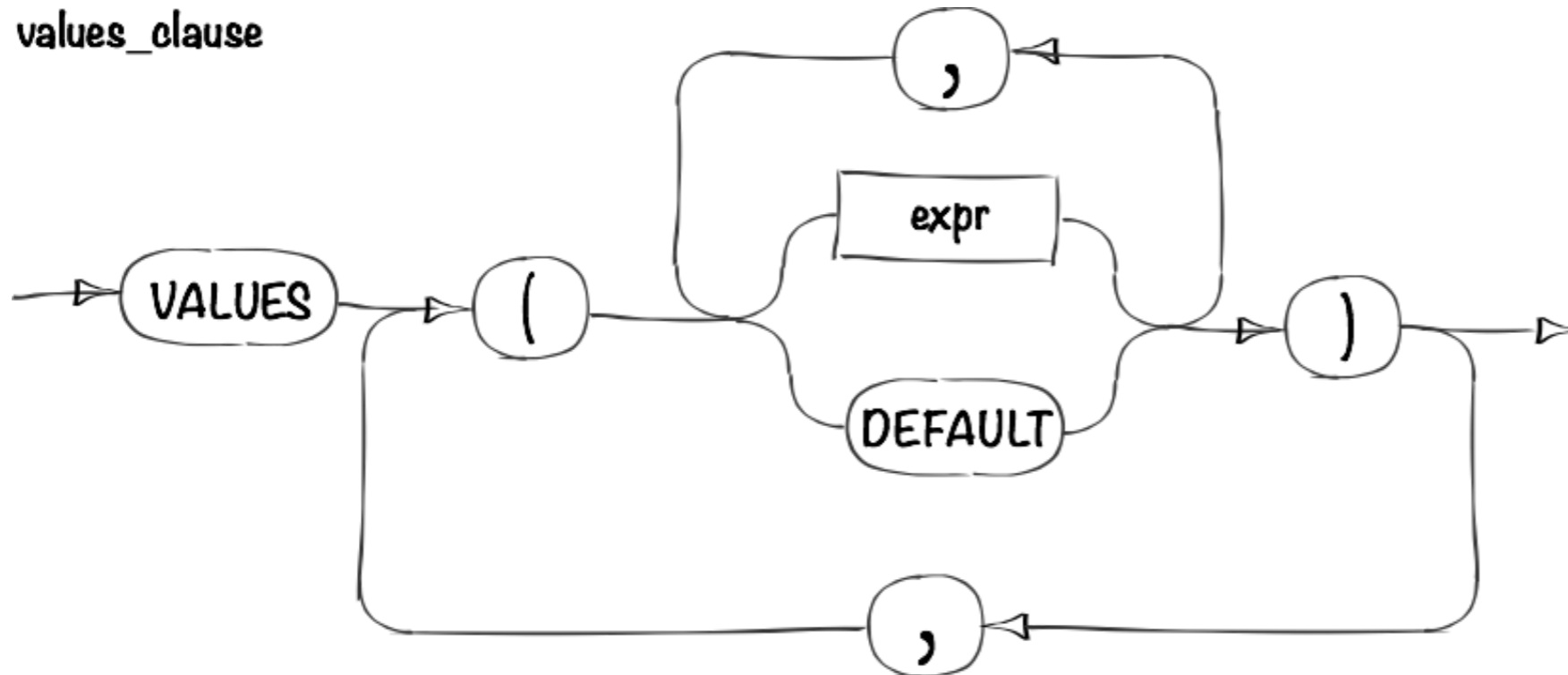
insert



insert\_into\_clause



values\_clause



# INSERT – Beispiele

- Beispiel für die Verwendung mit einzelnen Werten (*values\_clause*)
  - ```
insert into person
      (persnr, name, vorname, geburtsdatum)
values ('42', 'Adams', 'Douglas',
      to_date('11.03.1952', 'DD.MM.YYYY'));
```
- Beispiel für das Einfügen mehrerer Zeilen (*subquery*)
  - ```
insert into ort (ort, name)
      select ort_key, ortsname from gis_export_tab; [1]
```

[1] Die Tabelle GIS\_EXPORT\_TAB wird nicht mit den Beispieldaten angelegt.

# INSERT – Datenformat

- Beim **INSERT** müssen die Datenformate der neu eingefügten Daten in jeder Spalte mit den erforderlichen Formaten übereinstimmen.
- Beim **INSERT** müssen nicht alle Spalten angesprochen werden. Die nicht angegebenen Spalten sind dann **NULL**.

# Datenmanipulation (1)

- Eröffne eine neue Filiale in der Region *Thurgau* in der Schweiz (Länderschlüssel *CH*). Die Filiale hat die Filialnummer *5000* und liegt in der Hauptstadt der Region.
- Du kannst dir vorab mit einer **SELECT**-Anweisung das Kürzel und die Hauptstadt der Region Thurgau ausgeben lassen.
- Führe dann eine **INSERT**-Anweisung mit entsprechenden Angaben in der **VALUES**-Klausel aus, mit der eine entsprechende Zeile in der Tabelle `filiale` eingefügt wird.
- Ordne der Filiale *5000* in der Tabelle `listung` alle Artikel der Produktgruppen zu, die mit *HRY* beginnen. Die Listung soll ab heute gelten.
- Du kannst zuerst eine **SELECT**-Anweisung schreiben, um die Filialnummer (als Literal anzugeben), die Artikelnummer der entsprechenden Artikel und das Systemdatum (`current_date`) auszugeben.
- Diese **SELECT**-Anweisung kann dann in einer **INSERT**-Anweisung verwendet werden.
- (Fortsetzung nächste Seite...)

# Datenmanipulation (2)

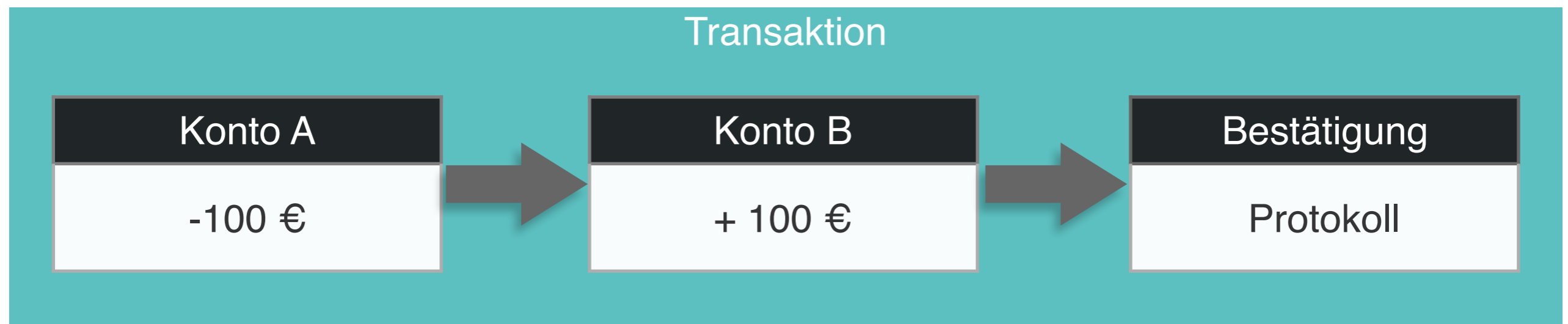
- Die Filialnummer ist als Primärschlüssel definiert. Führe die **INSERT**-Anweisung aus der ersten Teilaufgabe erneut aus, um die Überprüfung von Konsistenzbedingungen bei DML-Anweisungen zu testen.
  - Die Doppelanlage mit gleichem Primärschlüssel muss zu einer Fehlermeldung führen.
- Versuche, eine Filiale ohne Filialnummer anzulegen.
  - Die Anlage ohne Primärschlüssel muss zu einer Fehlermeldung führen.

# Transaktionen

- Eine Transaktion ist eine Abfolge von DML-Anweisungen, die zusammen eine logische Arbeitseinheit bilden.
- Das Datenbanksystem bietet Möglichkeiten, solche logische Arbeitseinheiten zu formen.
- Unterschiedliche Isolation Level erlauben es, den zwischenzeitlich potentiell „inkonsistenten“ Stand nach außen unsichtbar zu halten.
- Im Fehlerfall wird der vorherige konsistente Stand wiederhergestellt.

# Transaktion – Ein Beispiel

- Eine Banküberweisung transferiert 100 € von einem Konto A auf ein Konto B. Diese Transaktion könnte z.B. aus drei verschiedenen Operationen bestehen:
  - Belastung von Konto A
  - Gutschrift auf Konto B
  - Anlage einer Buchungsbestätigung



# Transaktion – Garantien

- Die Datenbank garantiert, dass
  - entweder alle Operationen ausgeführt werden (damit im Beispiel die Konten ausgeglichen bleiben und eine korrekte Dokumentation erfolgt),
  - oder keine der Operation ihre Änderungen festschreibt, sofern auch nur eine der Anweisungen nicht korrekt ausgeführt werden kann
- Der potentiell inkonsistente Datenstand während einzelner Schritte der Transaktion ist für andere Anwender und Anwendungen je nach Isolation Level nicht sichtbar.

# Datenbanktransaktion

- Der Abschluss einer Transaktion wird mit der Anweisung **COMMIT** explizit bestätigt und damit die Änderungen an der Datenbank freigegeben.
- Um alle Änderungen seit dem letzten **COMMIT** zu verwerfen, dient die Anweisung **ROLLBACK**. Mit ihr werden alle Änderungen einer Transaktion widerrufen.
- Alle DML-Befehle unterstehen dem Transaktionskonzept. Der erste DML-Befehle (nach der letzten Transaktion) öffnet automatisch eine neue Transaktion.

# Datenbanktransaktionen – Spezifika

- Bei der Umsetzung von Transaktionen unterscheiden sich Datenbanksysteme.
- Details wie z.B. expliziter Beginn einer Transaktion, Sperrmechanismen für einzelne Zeilen, ganze Tabellen, Definition von Zwischenständen (Savepoints) können unterschiedlich ausgeprägt sein.

# Transaktionen (1)

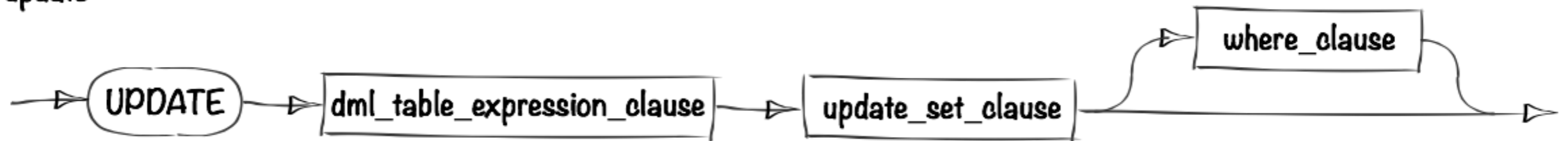
- Schalte die Auto-Commit Einstellung im SQL-Client aus! (unter „Einstellungen“)
- Lege eine Filiale mit der Filialnummer *6000* an.
- Lies die Tabelle `filiale`. Die Filiale mit der Filialnummer *6000* erscheint.
- Führe die Anweisung `ROLLBACK` aus (alternativ: Klicke das Rollback-Symbol in der Symbolleiste links neben der SQL-Eingabe).
- Lies erneut die Tabelle `filiale`. Die Änderung wurde verworfen.
- (Fortsetzung nächste Seite...)

# Transaktionen (2)

- Führe erneut die **INSERT**-Anweisung zum Anlegen der Filiale *6000* aus.
- Öffne eine neue Sitzung (Menü *Fenster – Neue Sitzung* öffnen) und melde dich an der gleichen Datenbank wie im ersten Fenster an.
- Lies in der neuen Sitzung die Tabelle **filiale**. Die Filiale *6000* ist in der neuen Sitzung noch nicht sichtbar, in der Originalsitzung ist sie sichtbar.
- Führe in der Originalsitzung, in der die **INSERT**-Anweisung ausgeführt wurde, die Anweisung **COMMIT** aus (alternativ: Klicke auf das **COMMIT**-Symbol in der Symbolleiste links neben der SQL-Eingabe).
- Nach dem **COMMIT** ist die neue Filiale in beiden Sitzungen sichtbar.

# Daten ändern mit UPDATE

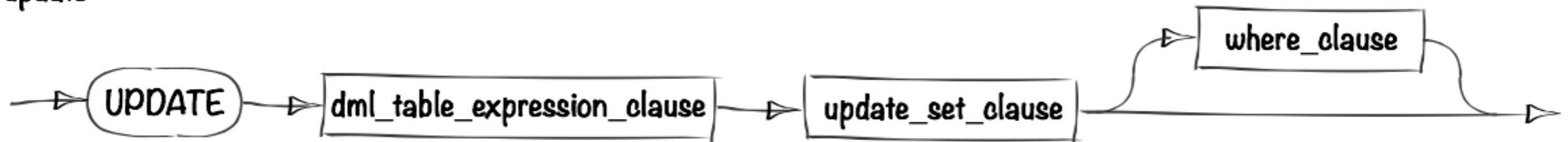
update



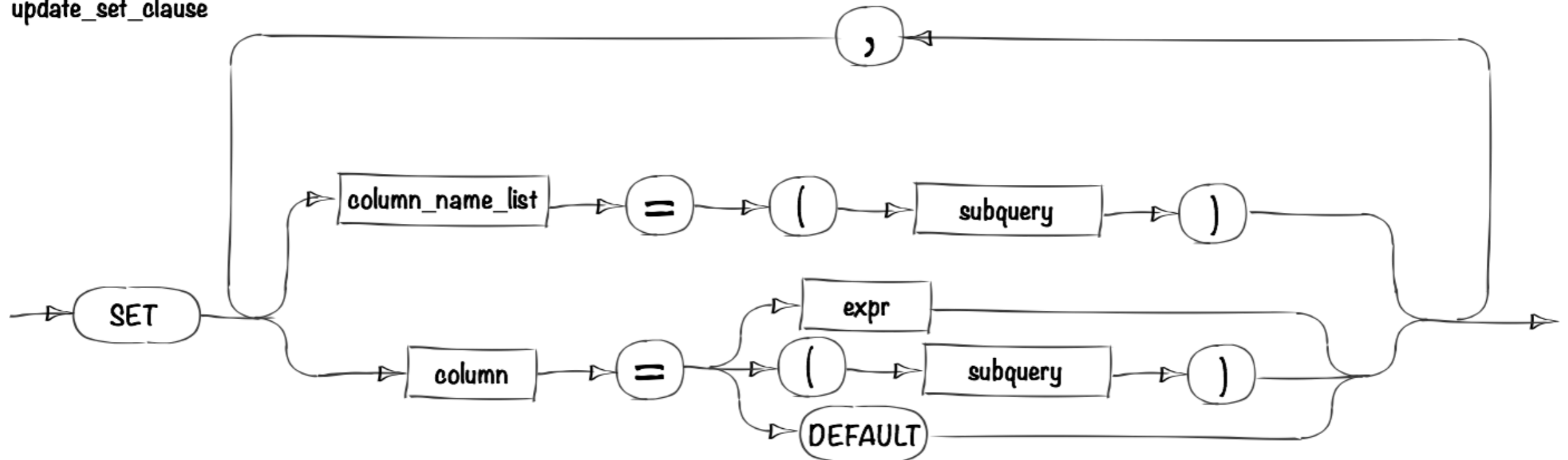
- Die Platzhalter *dml\_table\_expression\_clause* und *where\_clause* entsprechen den bereits eingeführten Syntaxkonstrukten (**SELECT** & **INSERT**)

# Daten ändern mit UPDATE (2)

update



update\_set\_clause



- Die Platzhalter *subquery* & *where\_clause* entsprechen den bereits eingeführten Syntaxkonstrukten

# UPDATE – Beispiele

- Beispiel für ein **UPDATE** mit festen Werten
  - ```
update person
  set      name = 'Molière', vorname = null
  where   persnr = '1622';
```
- Beispiel für **UPDATE** mit subquery
  - ```
update ort
  set name =
    (select ortsname from gis_export_tab
     where ort_key = 'IT00001')
  where ort = 'IT00001';[1]
```

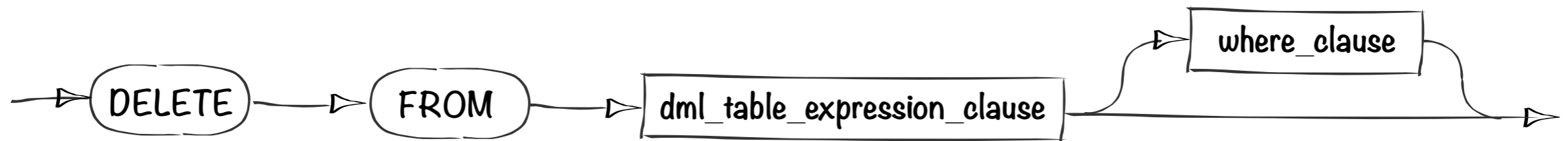
[1] Die Tabelle GIS\_EXPORT\_TAB wird nicht mit den Beispieldaten angelegt.

# UPDATE – Datenformat

- Beim **UPDATE** müssen die Datenformate der geänderten Daten in jeder Spalte mit den erforderlichen Formaten übereinstimmen.
- Beim **UPDATE** müssen nicht alle Spalten angesprochen werden. Die nicht angegebenen Spalten behalten ihren bisherigen Wert.

# Daten löschen mit DELETE

delete



- Die Platzhalter *dml\_table\_expression\_clause* und *where\_clause* entsprechen den bereits eingeführten Syntaxkonstrukten

# DELETE – Beispiele

- Beispiel für ein **DELETE**
  - `delete from ort  
where ort = 'IT00001';`
- Ohne **WHERE**-Klausel werden alle Zeilen aus der Tabelle gelöscht
  - `delete from gis_export_tab; [1]`

[1] Die Tabelle GIS\_EXPORT\_TAB wird nicht mit den Beispieldaten angelegt.

# Datenmanipulation

- Ändere die in einer vorausgegangenen Übung angelegte Filiale mit der Filialnummer *5000* so ab, dass die Adresse  
„*Rathausplatz*  
*8500 Frauenfeld*“  
ist. Achte darauf, nur diese einzelne Zeile zu ändern!
- Mache die Änderung mit **COMMIT** dauerhaft wirksam.
- Lösche aus der Tabelle **filiale** die in einer vorausgegangenen Übung angelegte Filiale mit der Filialnummer *6000*.
- Mache die Änderung mit **COMMIT** dauerhaft wirksam.
- Solltest Du bei den Übungen versehentlich zu viele Daten manipuliert oder gelöscht haben, kannst du den Inhalt der Schulungstabellen mit dem Tutorialsymbol in der Symbolleiste des SQL-Clients wieder auf den Ausgangszustand zurücksetzen.



# Locking und Deadlocks

- Um Fehler durch gleichzeitige Änderungen mehrerer Benutzer an einer Ressource zu vermeiden, werden Ressourcen bei Änderungen gesperrt.
- Ein Deadlock entsteht, wenn eine Session (A) eine Ressource benötigt, die durch eine andere Session (B) geblockt ist und Session (B) gleichzeitig eine Ressource benötigt, die durch Session (A) geblockt ist.
- Die Sessions warten darauf, dass die jeweils andere die benötigten Ressourcen freigibt.
- Es können auch mehr als zwei Sessions betroffen sein.

# Deadlock

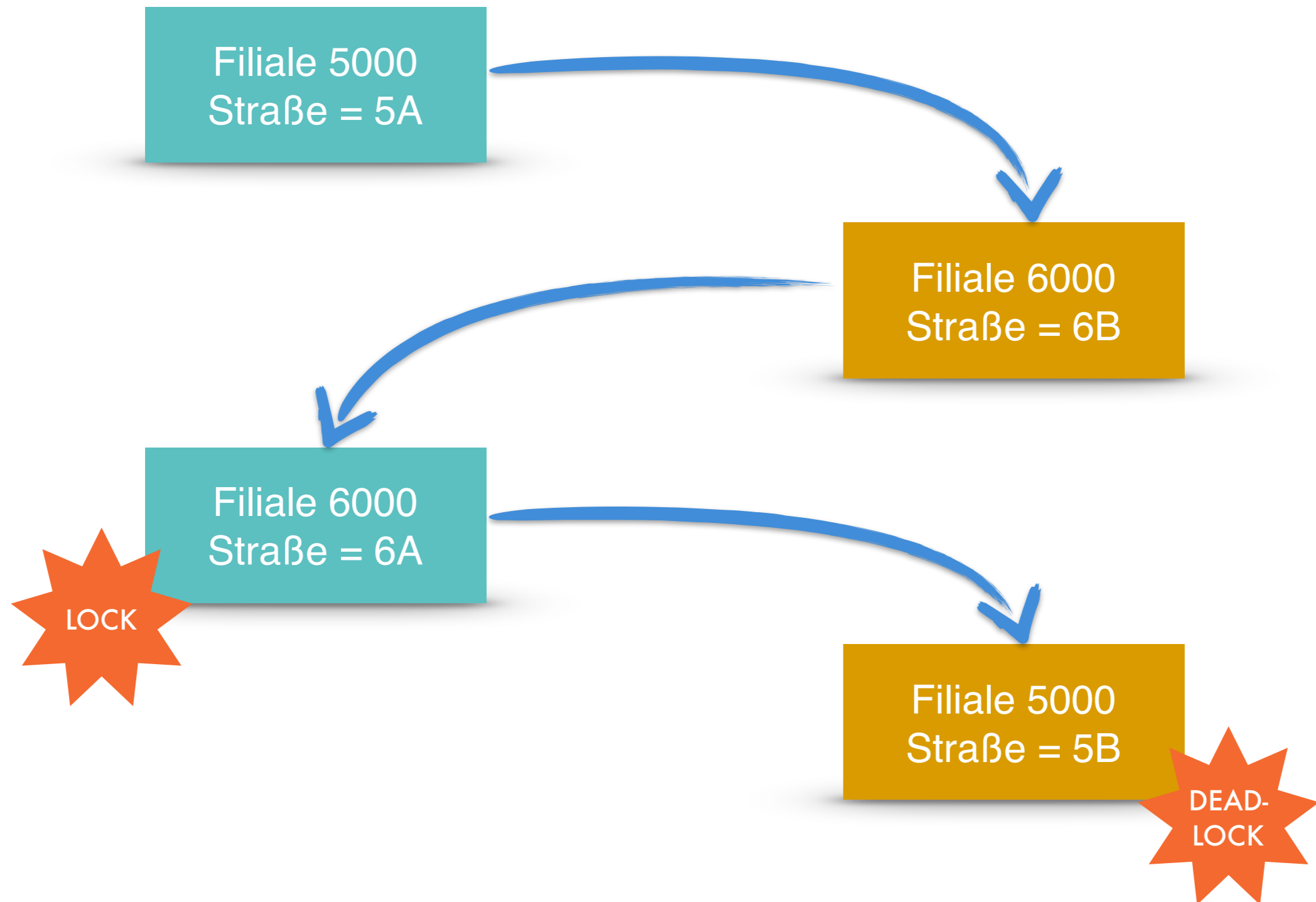
Zeitachse



# Locking & Deadlock

- Achte darauf, die Einstellung Auto-Commit für diese Übung zu deaktivieren!
- Erzeuge in der Tabelle `filiale` erneut eine Filiale mit der Filialnummer `6000`. Schließe die Anlage mit `COMMIT` ab.
- Öffne über das Menü *Fenster – Neue Sitzung* ein zweites Fenster (starte *nicht* ein zweites Mal die Anwendung!) und melde dich dort an der gleichen Datenbank an. In diesem Fenster ebenfalls Auto-Commit ausschalten!
- Ändere nun abwechselnd und über Kreuz in den beiden Sitzungen die Straßennamen der Filialen `5000` und `6000` – jeweils ohne `COMMIT`. Der Ablauf ist in der Darstellung auf der Folgeseite skizziert.
- (Fortsetzung nächste Seite...)

# Deadlock



- Wie reagiert das Datenbanksystem?