

SQL

Grundlagen

Data Definition Language

ROLLBACK COMMIT
RELATIONAL DDL RDBMS DELETE VALUES
INDEX VIEW
SELECT INNER JOIN
EXTRACT TRIGGER
DML TRANSACTION
NUMERIC ISOLATION LEVEL
DATA DEFINITION LANGUAGE
DATABASE
DEFAULT CREATE
LEFT OUTER JOIN INSERT
SAVEPOINT
TABLE
DROP
EXPLAIN PLAN
DEADLOCK
RIGHT OUTER JOIN

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 Anweisungen (DML – z.B. aus Teil 2 und 3 der Schulung)
- Für die Übungsdatenbank: Mac/PC mit aktueller Java-Version

Übersicht

- Erstellen, Ändern und Löschen von Tabellen
- Benutzerrechte verwalten
- Primärschlüssel, Fremdschlüssel (Referenzielle Integrität)
- Constraints, Views
- Verschiedene Datenbankobjekte im Überblick

Tabellen anlegen mit CREATE TABLE

- Um eine neue Tabelle anzulegen, wird der Befehl **CREATE TABLE** verwendet.
- Auf eine neue Tabelle hat zuerst nur der Benutzer Zugriffsrechte, der die Tabelle angelegt hat (siehe später).

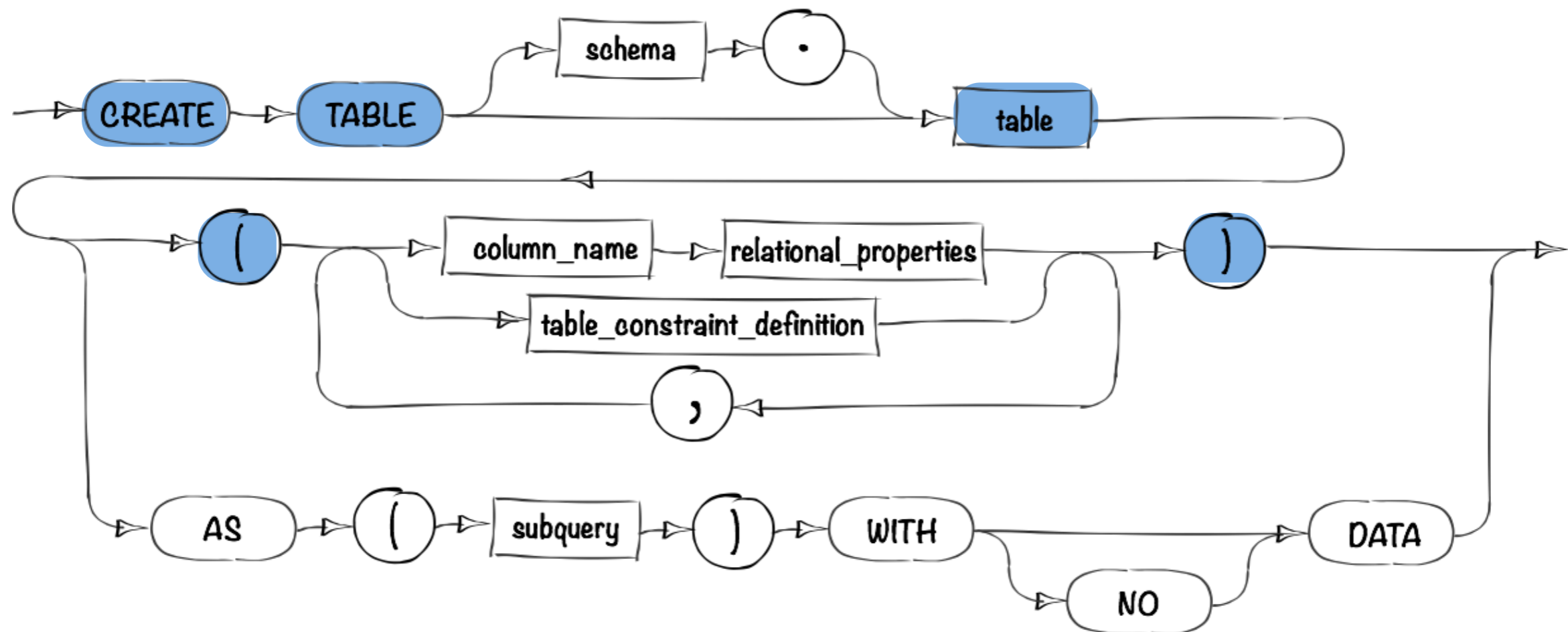
CREATE TABLE – Beispiele

- Anlegen einer Tabelle

- `create table buch`

```
( buch numeric(10) not null,  
  isbn varchar(17),  
  titel varchar(200) default 'Ohne Titel' );
```

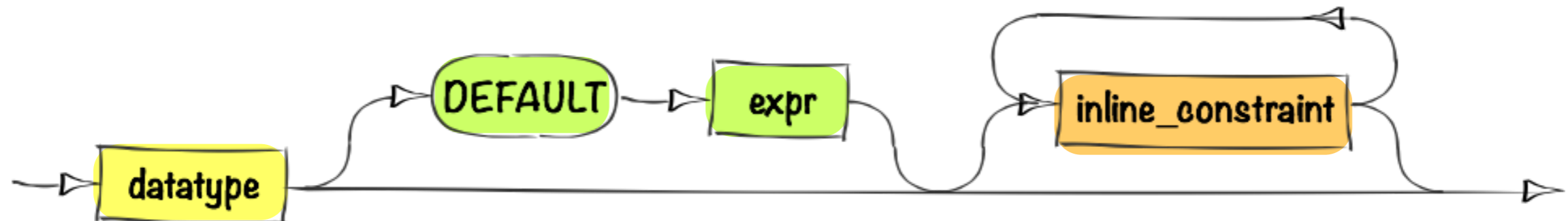
create_table



CREATE TABLE – Beispiele

- Anlegen einer Tabelle
 - `create table buch`
 (`buch numeric(10) not null,`
 `isbn varchar(17),`
 `titel varchar(200) default 'Ohne Titel');`

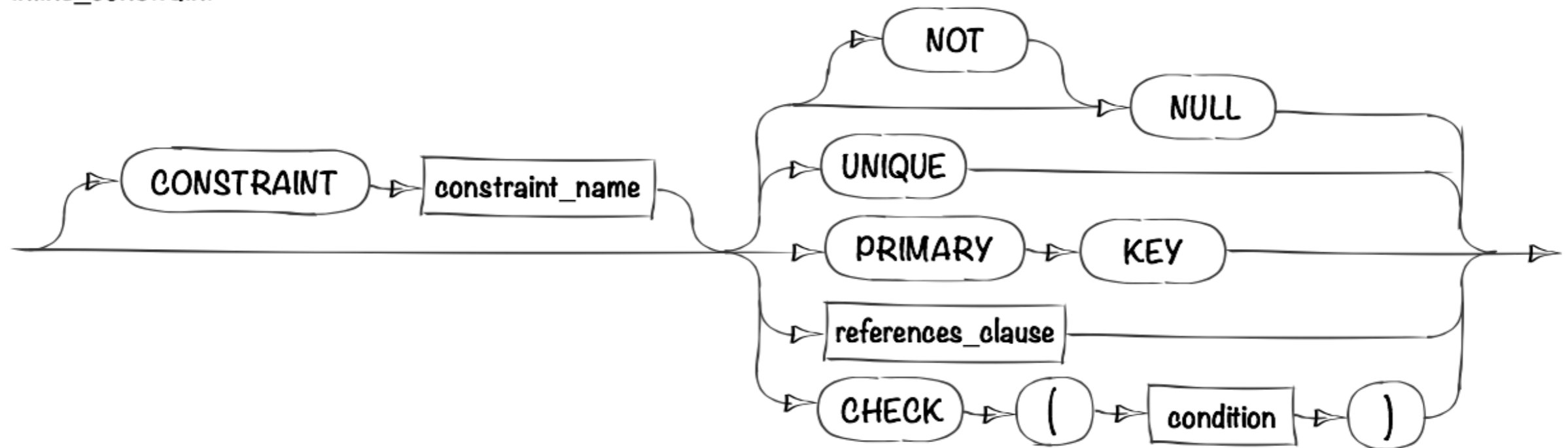
relational_properties



CREATE TABLE – Beispiele

- Anlegen einer Tabelle
 - `create table buch`
 `(buch numeric(10) not null,`
 `isbn varchar(17),`
 `titel varchar(200) default 'Ohne Titel');`

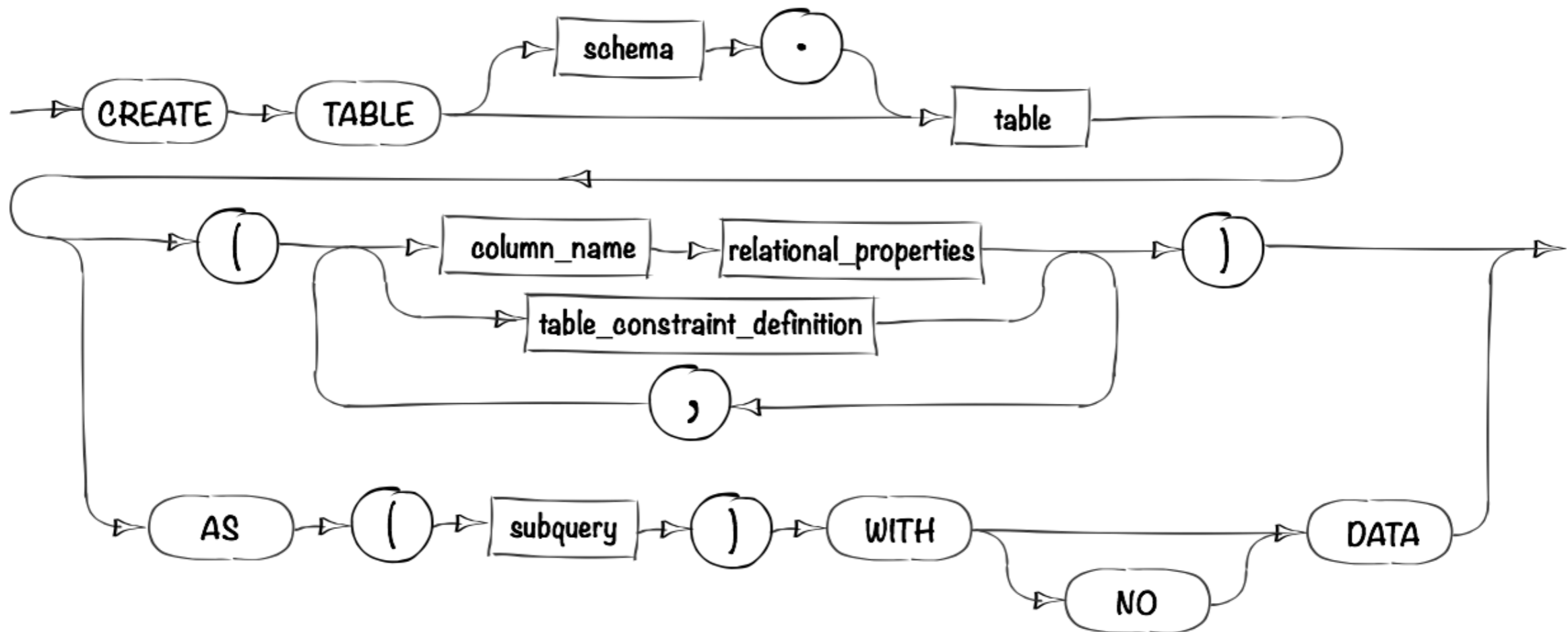
inline_constraint



CREATE TABLE – Beispiele

- Anlegen einer Tabelle
 - `create table buch as`
 `(select book_number, isbn, title from book_import)`
 `with data; -- HSQL spezifisch`

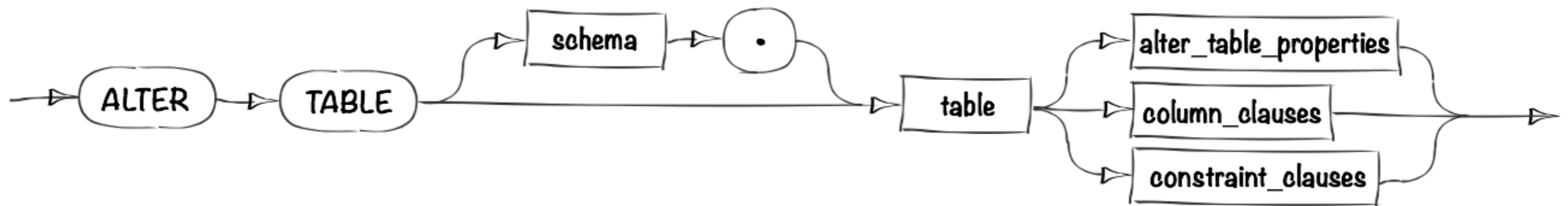
create_table



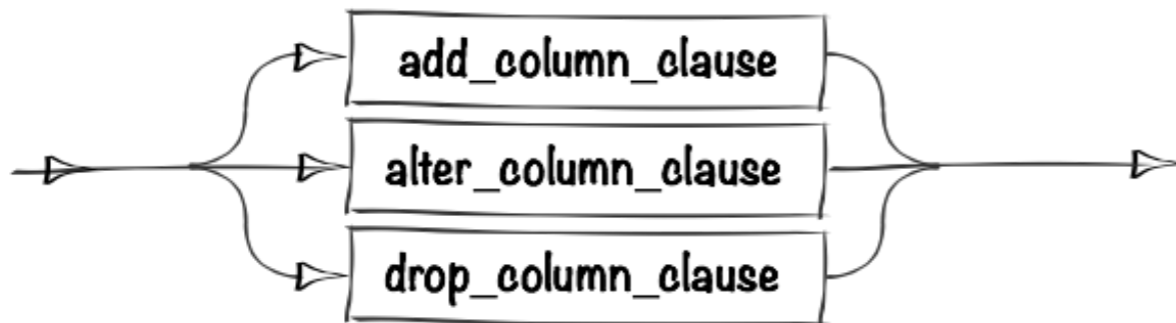
ALTER TABLE

- Die Struktur einer Tabelle kann nachträglich mit **ALTER TABLE** verändert werden

alter_table



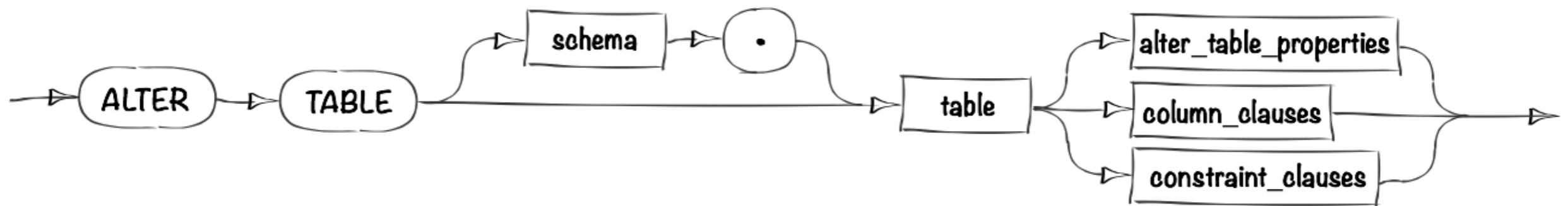
column_clauses



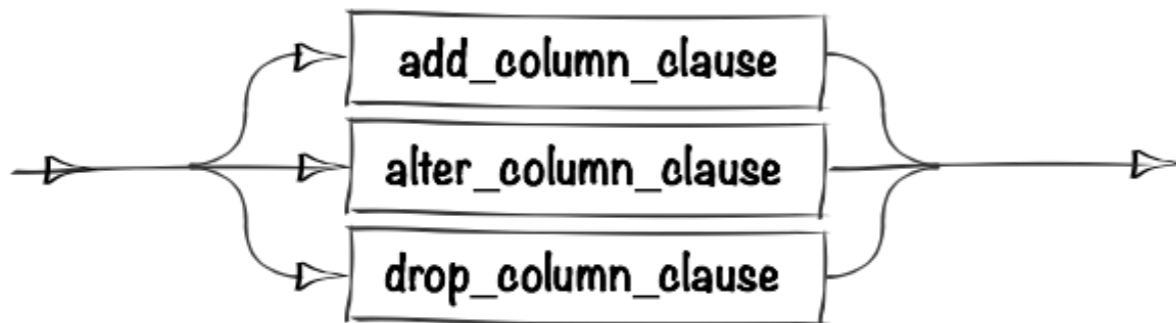
ALTER TABLE

- Die Struktur einer Tabelle kann nachträglich mit **ALTER TABLE** verändert werden

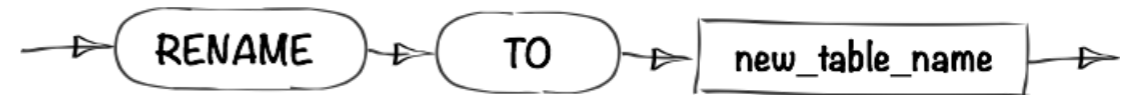
alter_table



column_clauses



alter_table_properties

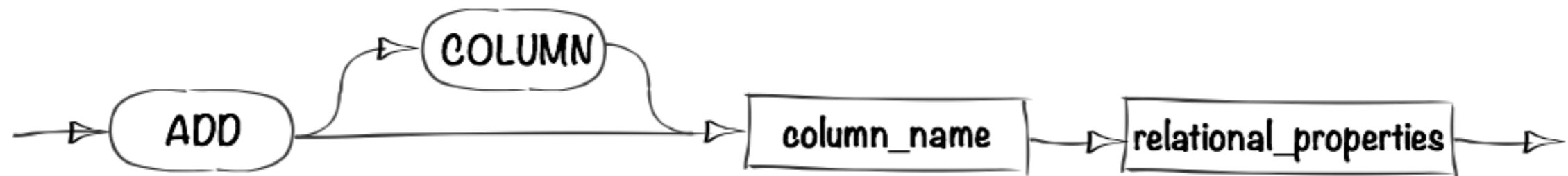


bestehende SQL-Anweisungen müssen angepasst werden – nicht im SQL Standard

ALTER TABLE – Beispiele

- Eine weitere Spalte hinzufügen
 - `alter table buch add autor integer;`

add_column_clause



ALTER TABLE – Beispiele (2)

- Eine bestehende Spalte ändern
 - `alter table buch alter titel
set data type varchar(222);`
 - `alter table buch alter titel varchar(222);`
 - `alter table buch alter titel set not null; -- oder set null`
 - `alter table buch alter titel set default 'Ohne Titel...';`
 - `alter table buch alter titel drop default;`

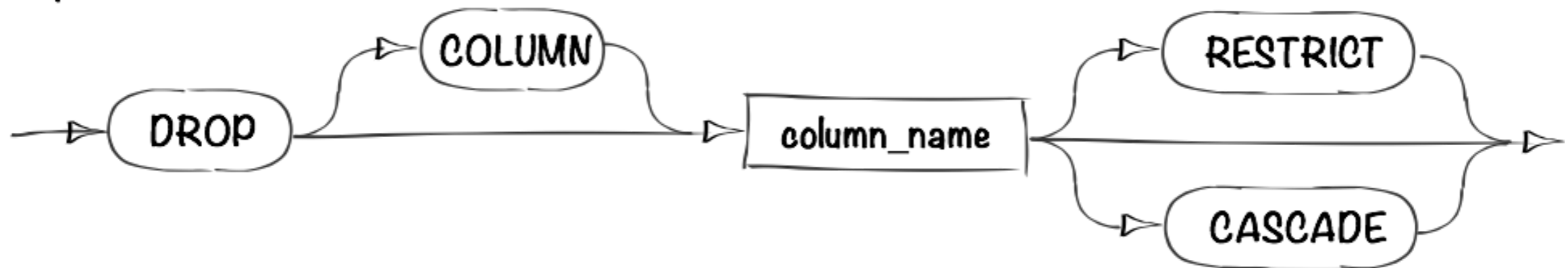
alter_column_clause



ALTER TABLE – Beispiele (3)

- Eine bestehende Spalte entfernen
 - `alter table buch drop autor;`

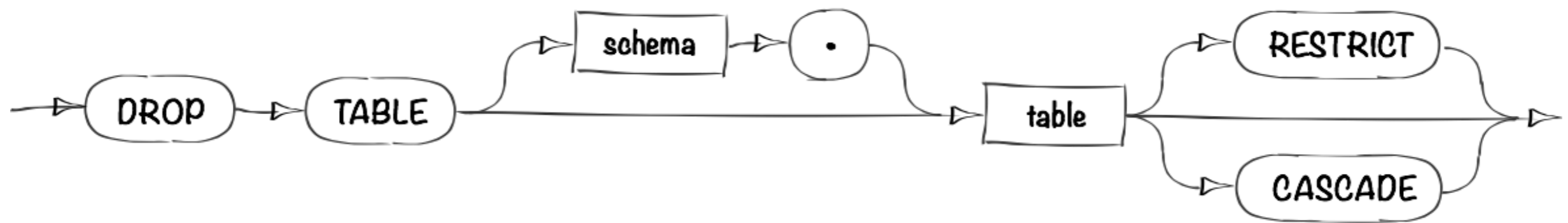
drop_column_clause



Tabellen löschen – DROP TABLE

- Bestehende Tabellen können mit **DROP TABLE** komplett gelöscht werden.
- Mit **DROP TABLE** wird nicht nur der Inhalt, sondern die komplette Tabellenstruktur aus der Datenbank gelöscht.
- Beispiel:
 - `drop table buch;`

drop_table



Datenbankobjekte

- Lege eine Tabelle **kunde** mit folgenden Spalten an:
 - **kundennummer**, Typ `varchar(10)`, als Constraint `not null`
 - **vorname**, Typ `varchar(20)`
 - **nachname**, Typ `varchar(20)`
 - **anlagedatum**, Typ `date`
- Füge einen Datensatz (für eine beliebige Person) in die Tabelle **kunde** ein.
- Erweitere die Tabelle **kunde** um eine Spalte
 - **geburtsdatum**, Typ `date`

Benutzer, Schema und Rechte

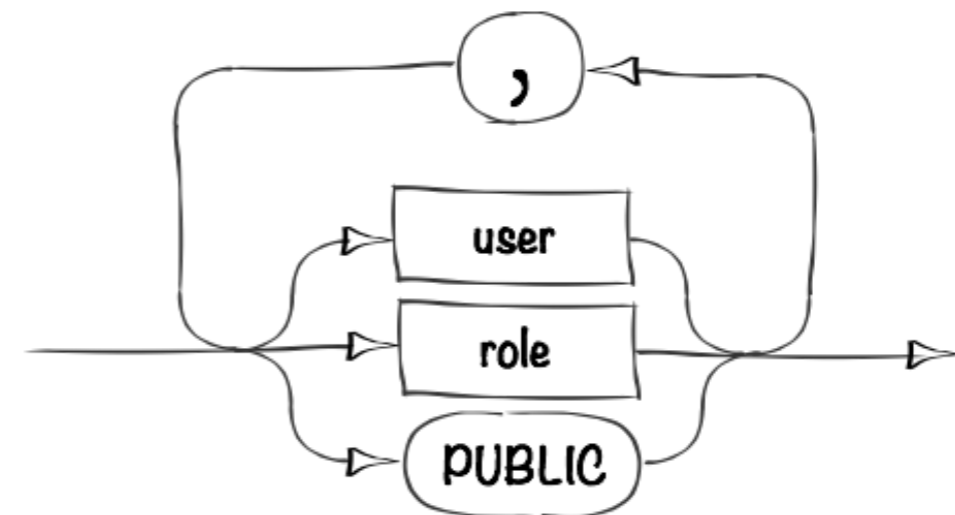
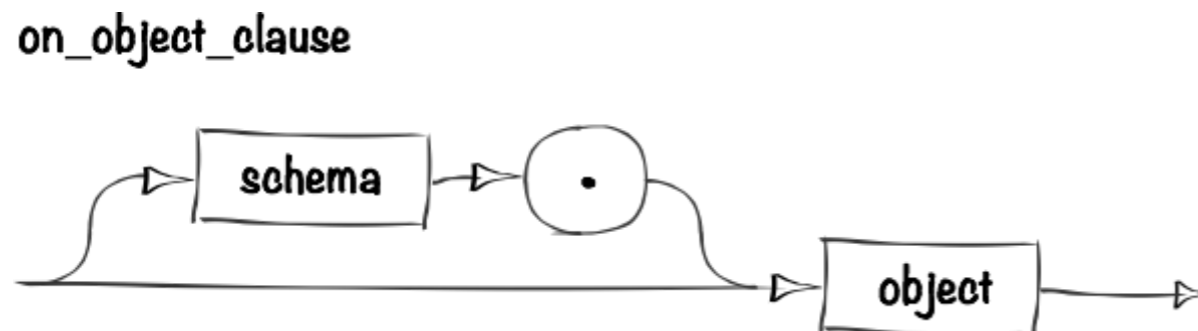
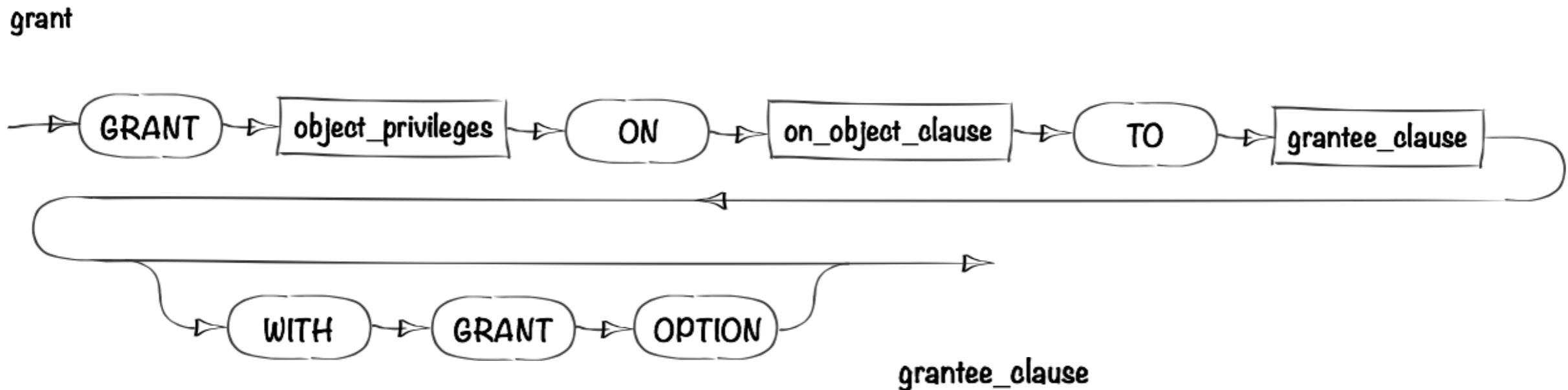
- Damit ein Benutzer eine Tabelle anlegen kann, benötigt er entsprechende Privilegien / Rechte.
- Tabellen, die in einem anderen Schema liegen, müssen mit einem Präfix angesprochen werden, z.B.
`select * from john_doe.ort;`
- Die Unterscheidung nach Benutzer, Schema und darauf aufbauender Rechteverwaltung ist in den unterschiedlichen Datenbanksystemen nicht einheitlich gelöst.

Rechte verwalten

- Soll ein Datenbankobjekt (z.B. eine Tabelle) anderen Benutzern zur Verfügung stehen, müssen Zugriffsrechte auf das Objekt an die Benutzer vergeben werden
- Wichtige Objektprivilegien sind **SELECT**, **UPDATE**, **INSERT** und **DELETE** (die Privilegien heißen also i.d.R. so wie die zugehörige SQL-Anweisung)

Rechte vergeben

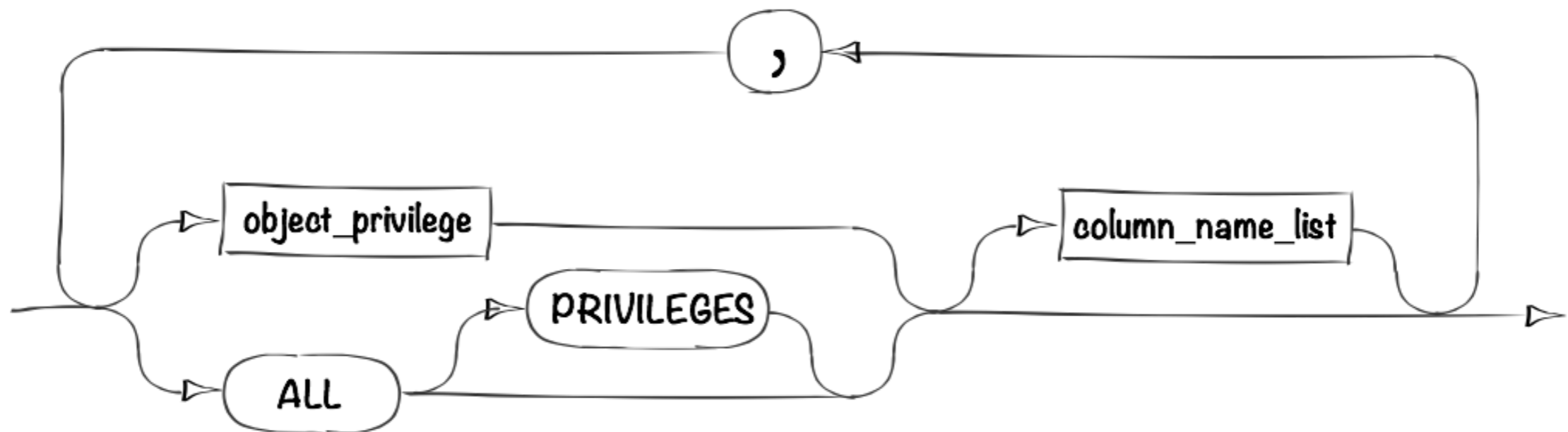
- Objektprivilegien werden mit dem **GRANT**-Befehl vergeben



Rechte vergeben (2)

- Mehrere Privilegien können mit einer Anweisung angegeben werden und sich entweder auf das Datenbankobjekt oder auf einzelne Spalten einer Tabelle oder eines Views beziehen

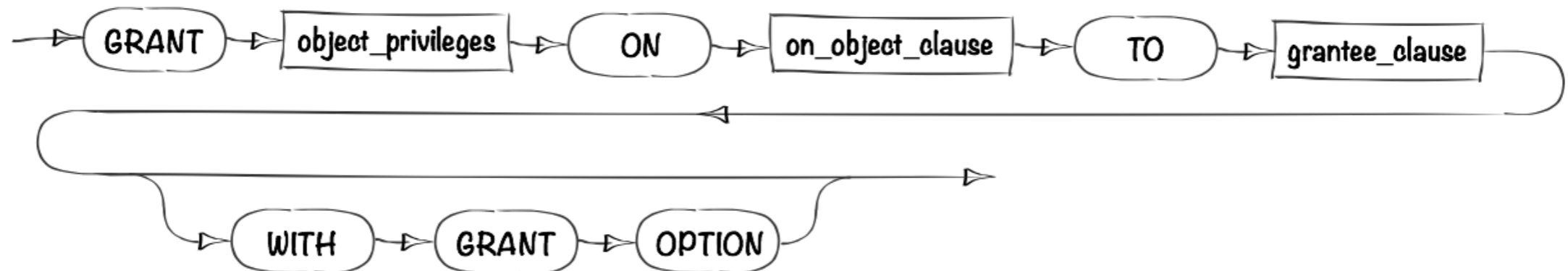
`object_privileges`



Rechte vergeben (3)

- Beispiel: Leserecht auf die Tabelle **person** an einen Benutzer mit der Benutzerkennung *john_doe* vergeben
- `grant select on person to john_doe;`

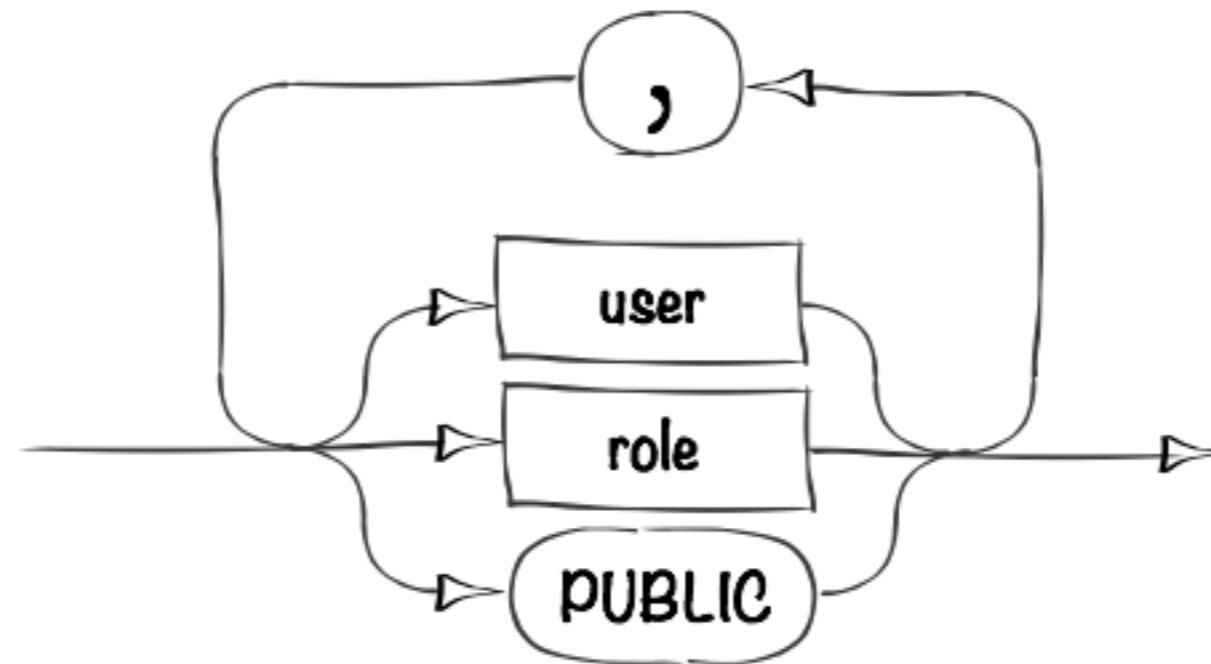
grant



Rechte vergeben (4)

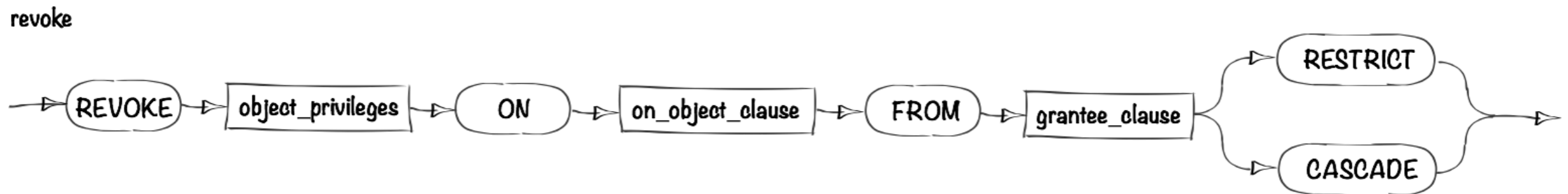
- Rechte können an einzelne Benutzer, an Benutzergruppen oder an alle vergeben werden

grantee_clause



Rechte entziehen

- Objektprivilegien werden mit dem **REVOKE**-Befehl entzogen
- Beispiel: Dem Benutzer *john_doe* die Sichtrechte auf die Tabelle **person** entziehen:
 - `revoke select on person from john_doe restrict;`



Datenbankobjekte und Rechte

- Lies die Tabelle `ort` des Benutzers *john_doe*. (Hinweis: Der bislang verwendete Benutzer hat Administrationsrechte, er darf die Tabellen aller anderen Benutzer lesen)
- Öffne eine neue Sitzung (Menü *Fenster – Neue Sitzung* öffnen) und melde dich dort als Benutzer *JOHN_DOE* (bitte Großschreibung beachten) mit dem Kennwort *geheim* an.
- Versuche, als Benutzer *john_doe*, die Tabelle `filiale` zu lesen. (Hinweis: In der Konstellation beim Aufbau der Schulungsdatenbank hat das Schema, in dem die bisher verwendeten Objekte liegen, die Bezeichnung `public`)
- Vergib – im Fenster mit der Anmeldung als Eigentümer der Tabellen – Leserechte auf der Tabelle `filiale` an den Benutzer *john_doe*.
- Versuche nun erneut, als Benutzer *john_doe* die Tabelle `filiale` zu lesen.
- Entziehe dem Benutzer *john_doe* wieder die Rechte auf der Tabelle `filiale`.

Primärschlüssel

- Eine Tabelle kann ein oder mehrere Attribute oder Kombinationen von Attributen enthalten, die einen Datensatz eindeutig bestimmen
- Eines dieser Attribute (oder Kombination) wird zum Primärschlüssel erklärt
- Aus Gründen der Effizienz kann es sinnvoll sein, trotz eines geeigneten Attributes einen Pseudoschlüssel (z.B. in Form einer laufenden Nummer) zu erzeugen.

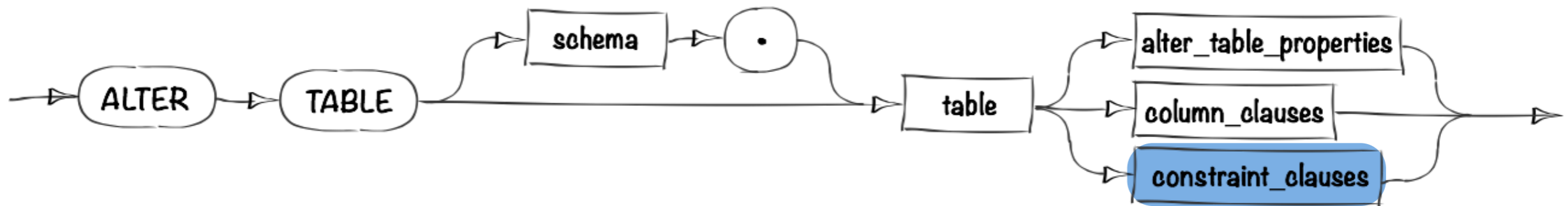
Primärschlüssel (2)

- Merkmale eines Primärschlüssels
 - Jede Tabelle kann nur einen Primärschlüssel haben
 - Die Werte müssen **NOT NULL** sein (wichtig bei nachträglicher Einführung)
 - Der Wert des Primärschlüsselattributs muss sich in jedem Datensatz unterscheiden
- Vorteile eines Primärschlüssels:
 - Kontrolle (keine doppelten Einträge)
 - Mehrere Tabellen können einfacher miteinander verknüpft werden

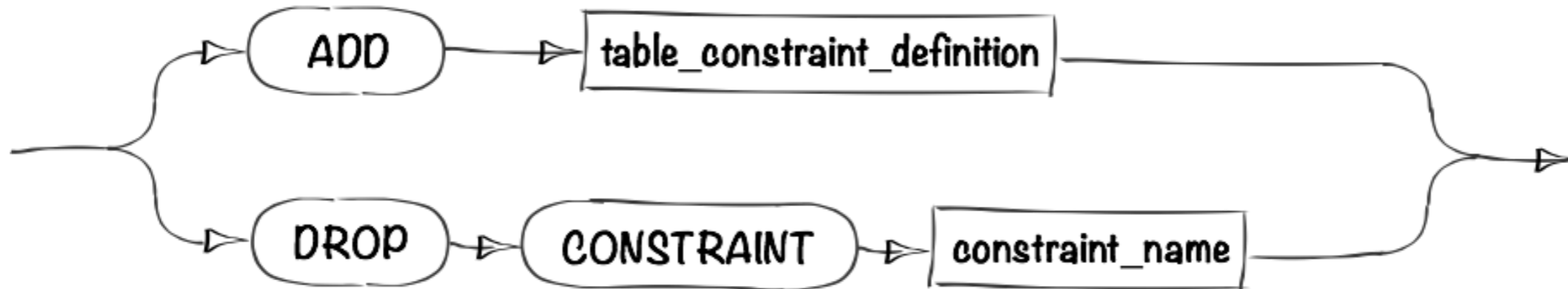
Primärschlüssel anlegen

- Primärschlüssel können direkt bei der Tabellenanlage als inline constraint oder nachträglich mit **ALTER TABLE** festgelegt werden.

alter_table

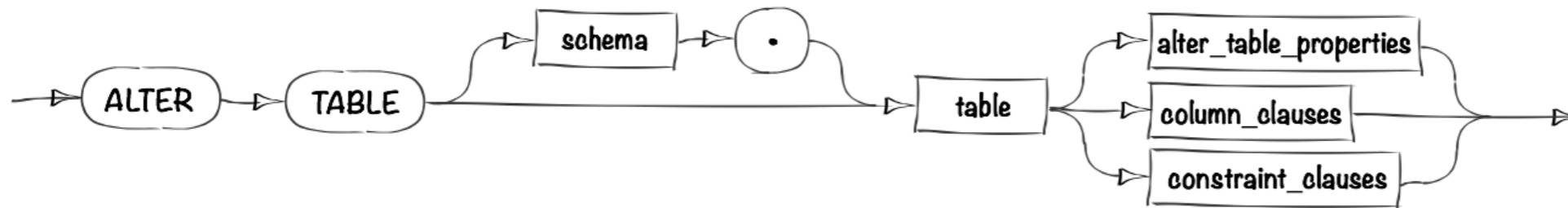


constraint_clauses

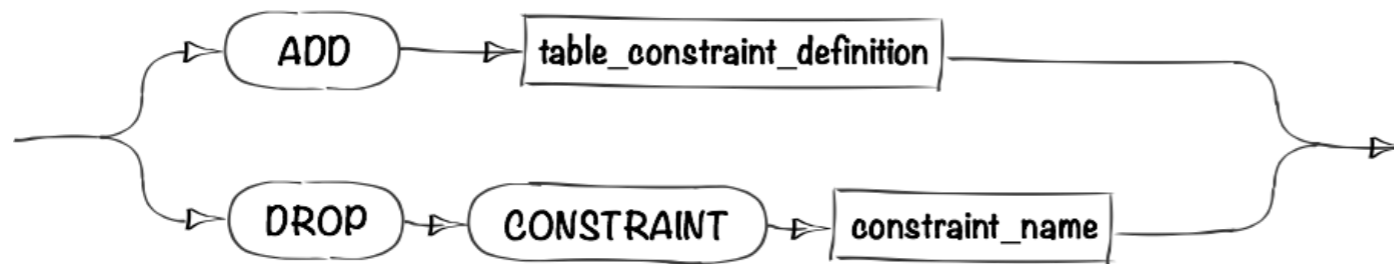


Constraints – Syntax

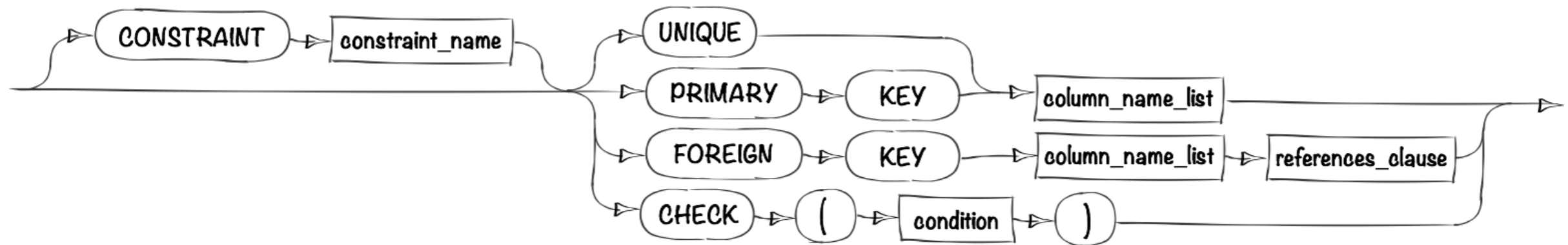
alter_table



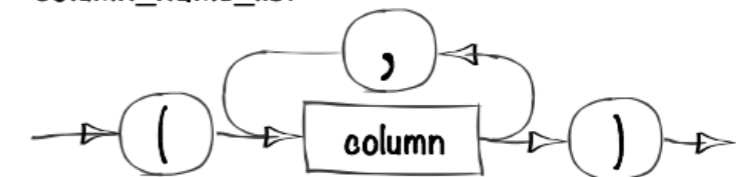
constraint_clauses



table_constraint_definition



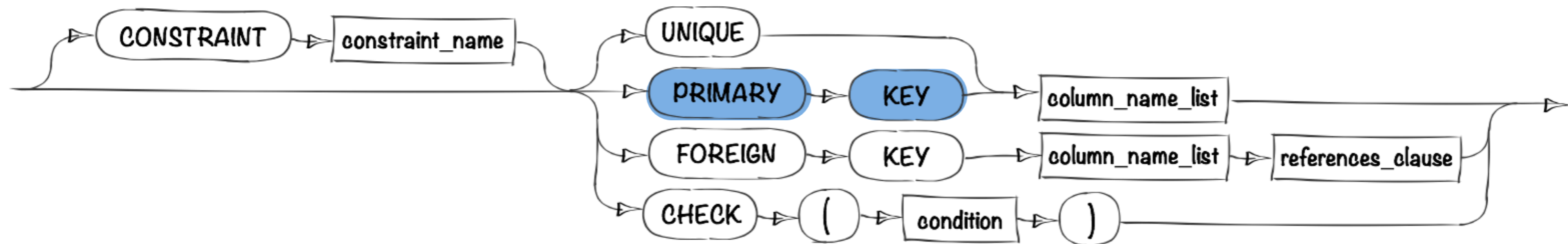
column_name_list



Primärschlüssel anlegen – Beispiel

- Beispiel: Primärschlüssel anlegen
 - `alter table buch`
`add constraint pk_buch`
`primary key (buch);`

table_constraint_definition



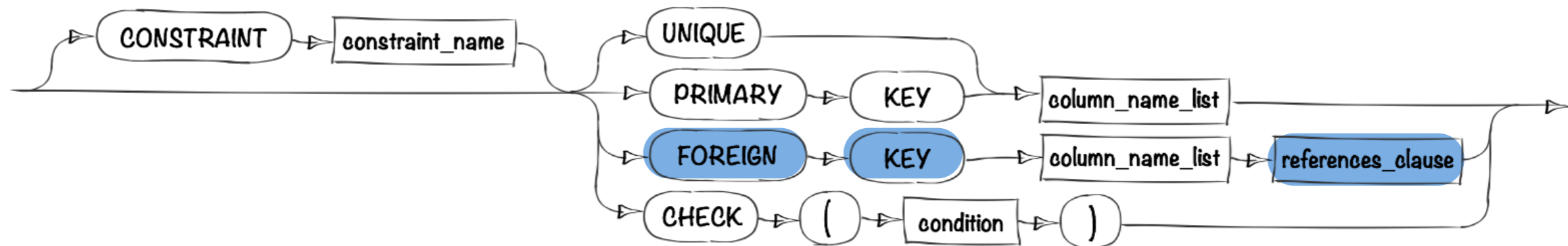
Fremdschlüssel

- Ein Attribut einer Tabelle (oder Attributkombination), das in einer anderen Tabelle Primärschlüssel (oder Schlüsselkandidat) ist, heißt Fremdschlüssel
- In einer Tabelle sind mehrere Fremdschlüssel möglich
- Mit Fremdschlüsseln wird das Prinzip der referenziellen Integrität realisiert
- In einer Spalte, die als Fremdschlüssel angelegt ist, sind nur Werte zulässig, die in der referenzierten Tabelle vorhanden sind

Fremdschlüssel anlegen

- Fremdschlüssel werden mit **ALTER TABLE** angelegt (oder direkt bei der Anlage einer Tabelle)
- Hierbei kommt die *references_clause* der **ALTER TABLE**-Anweisung zum Zug

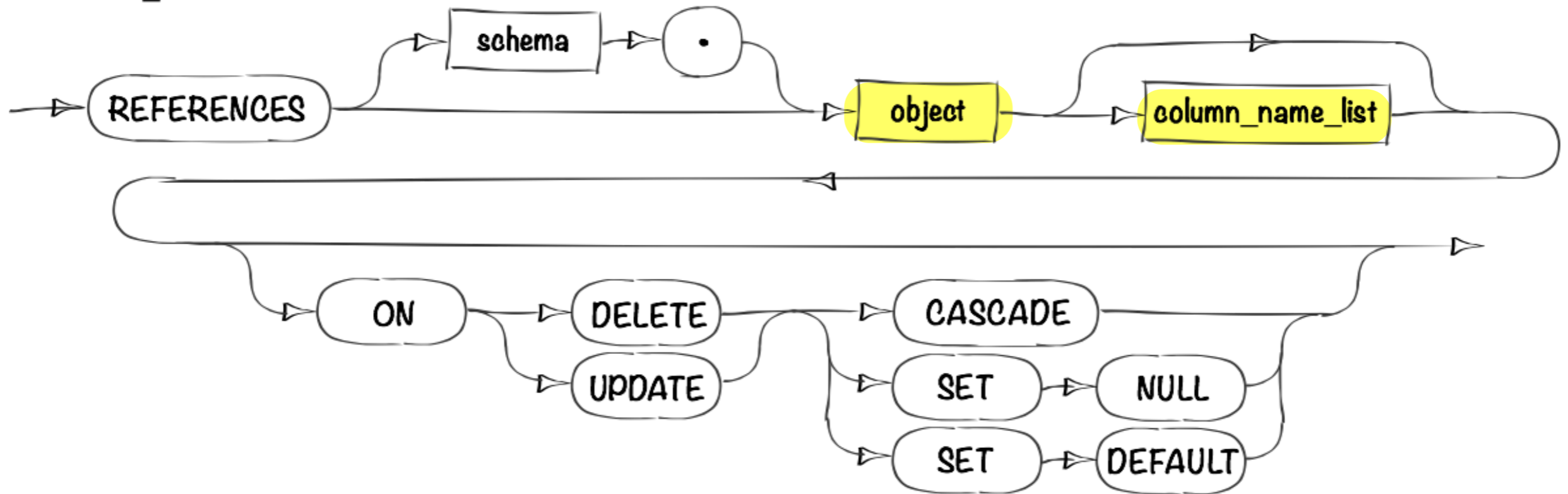
table_constraint_definition



Fremdschlüssel anlegen – Beispiel

- Beispiel: Referenzielle Integrität zwischen Büchern und ihrem Autor
 - `alter table buch`
`add constraint fk_autor`
`foreign key (autor)`
`references person (persnr) ;`

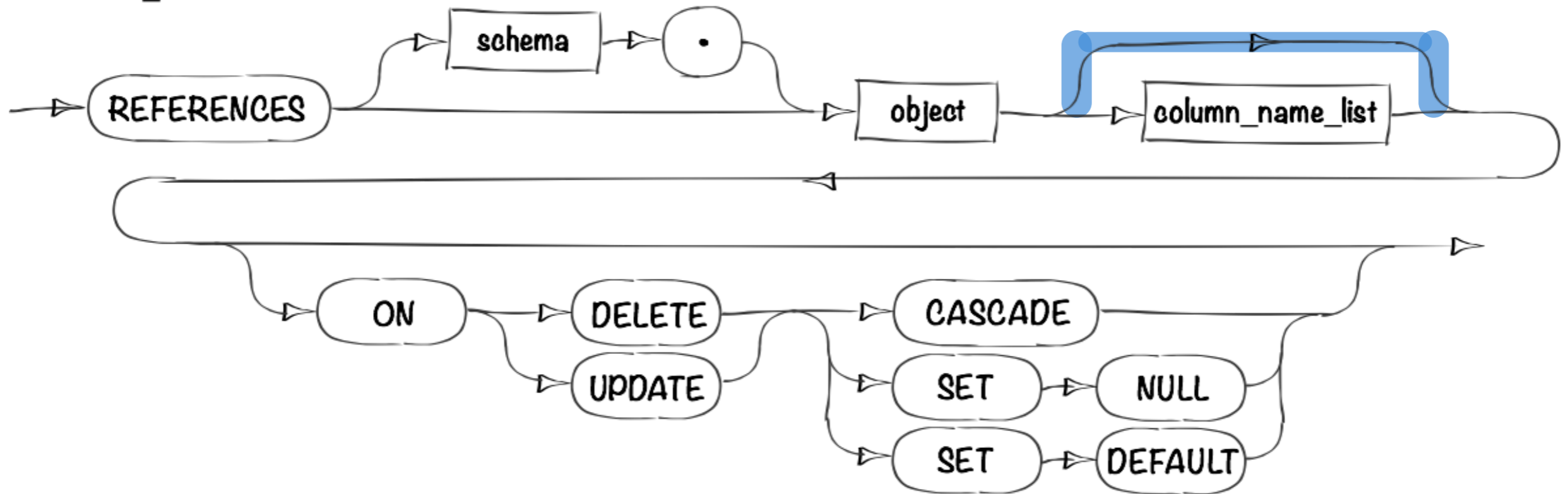
references_clause



Fremdschlüssel anlegen – Beispiel 2

- Ohne Spaltenangabe in der *references_clause* ist das Ziel der Primärschlüssel der verknüpften Tabelle.
- ```
alter table buch
add constraint fk_autor
foreign key (autor) references person;
```

references\_clause



# Primärschlüssel und Fremdschlüssel

- Lege auf der Tabelle **kunde** einen Primärschlüssel mit Namen **pk\_kunde** auf dem Feld **kundennummer** an.
- Lege eine neue Tabelle **workshopteilnehmer** mit den unten angegebenen Spalten an. Die Kombination beider Spalten soll als Primärschlüssel mit dem Namen **pk\_workshopteilnehmer** definiert werden. Der Primärschlüssel kann entweder bei der Tabellenanlage direkt angegeben oder nachträglich hinzugefügt werden.
  - **workshop**, Typ **integer**
  - **kundennummer**, Typ **varchar(10)**
- Füge in die Tabelle **kunde** einen Datensatz ein:
  - Kundennummer = *007*, Vorname = *James*, Nachname = *Bond*
- Versuche, den selben Datensatz ein zweites Mal einzufügen. Was passiert?
- (Fortsetzung nächste Seite...)

# Primärschlüssel und Fremdschlüssel

- Stelle sicher, dass Kunden nur bestehenden Workshops zugeordnet werden können und umgekehrt auch nur tatsächlich vorhandene Kunden einem Workshop zugeordnet werden können.  
Erstelle dazu jeweils einen Fremdschlüssel, ausgehend von den Spalten **kundennummer** und **workshop** der Tabelle **workshopteilnehmer**, mit einem Verweis auf die entsprechenden Primärschlüssel der zu verknüpfenden Tabellen. (Hinweis: Die Tabelle **workshop** wird mit den Schulungsdaten angelegt)
- Füge in die Tabelle **workshopteilnehmer** einen Datensatz ein, der als Kundennummer den Wert *nixda* und als Workshop einen existierenden Workshop (z.B. Workshop 1) enthält. Was passiert bei diesem Versuch?

# Check Constraints

- Check Constraints geben Bedingungen für den Inhalt eines Datensatzes an
- In einem Check Constraint können keine Bedingungen formuliert werden, die andere Datensätze (der gleichen oder gar anderer Tabellen) benötigen
- Beim Anlegen eines Check Constraints wird jeder Datensatz geprüft, ob der Check-Constraint erfüllt ist. Ist dies nicht der Fall, wird der Constraint nicht angelegt
- Der häufigste Check Constraint ist das Verbot von NULL-Werten durch den NOT NULL Constraint

# Check Constraints (2)

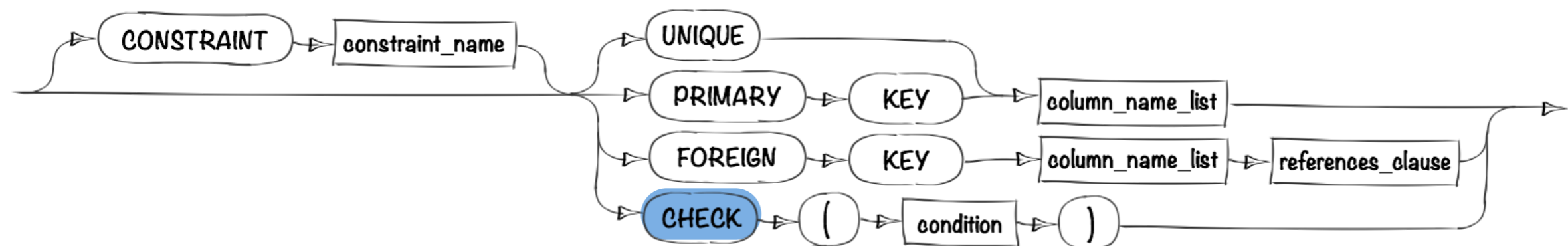
- Beispiel: Check Constraint anlegen (kurze Variante)

- `alter table ort`  
`add check (alt_max <= 8848) ;`

- Beispiel: Check Constraint anlegen (lange Variante)

- `alter table ort`  
`add constraint alt_max_check check (alt_max <= 8848) ;`

table\_constraint\_definition



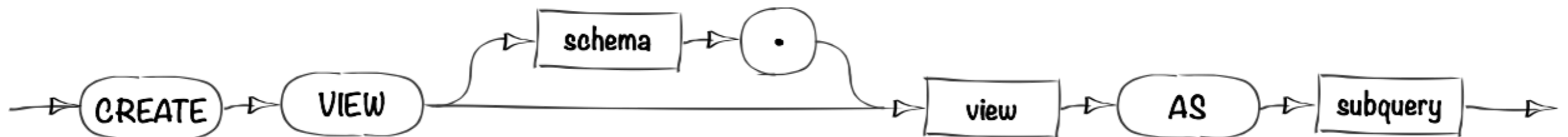
# View

- Eine View stellt eine virtuelle Tabelle dar, die erst bei der Ausführung gebildet wird. Es handelt sich um eine Sicht auf bestehende Tabellen
- Innerhalb eines **SELECT**-Befehls sind Views genau wie Tabellen zu behandeln
  - z.B. `select * from test_view;`
- Für Views sind die selben Objektprivilegien wie für Tabellen zu vergeben

# View anlegen

- Beispiel:
  - `create view timeline`  
`as select name, geburtsdatum from person;`

create\_view



# View – Anwendungsbeispiele

- Anwendungen für Views
  - Häufig gebrauchte **SELECT** Abfragen können permanent in der Datenbank gespeichert werden
  - Unterschiedliche Sichten auf Tabellen können zur Verfügung gestellt werden
  - Zugriffsrechte innerhalb einer Tabelle können auf Spalten- und Zeilenebene eingerichtet werden

# Constraints & Views

- Füge in der **workshop**-Tabelle eine Check-Constraint ein, die sicher stellt, dass es in der Tabelle nur Einträge mit einer Mindestanzahl von Teilnehmern größer als 0 gibt.
- Lege eine View mit dem Namen **basistexte\_de** an, die aus den Spalten **artikelnummer** und **artikeltext** der **artikeltext**-Tabelle besteht. Die Sicht soll eingeschränkt sein auf die Zeilen der Textart *BASIC* und die Locale *DE*.
- Lies den Inhalt der View **basistexte\_de**.
- Lösche die View **basistexte\_de** mit **drop**.

# Index

- Ein Index wird vom DBMS verwendet, um Abfragen in großen Tabellen zu beschleunigen
- Bei Suchabfragen werden – wenn möglich – nicht alle Datensätze durchsucht, sondern gezielt über den Index eingeschränkt
- Der Indexname wird normalerweise im **SELECT**-Befehl nicht explizit angegeben, sondern die Datenbank ermittelt selbst den zu verwendenden Index
- Ein zu verwendender Index kann (in manchen Datenbanksystemen) explizit vorgegeben werden

# Automatische Zähler

- Um eindeutige Primärschlüssel zu garantieren, sind oft fortlaufende Nummern ein geeignetes Mittel
- Fortlaufende Nummern können mittels z.B. `max()` Abfragen nur unzureichend erzeugt werden (Performance, Mehrbenutzerbetrieb,...)
- Die meisten Datenbanksysteme bieten Verfahren an, mit denen sequenzielle Zahlen erzeugt werden können. Hier gibt es herstellerspezifische Unterschiede.

# Automatische Zähler (2)

- Beispiel: Eine Sequence anlegen (HSQLDB)
  - `create sequence my_seq  
increment by 1  
start with 42;`
- Beispiel: Eine Sequence verwenden
  - `insert into epoche (epoche, bezeichnung)  
values (next value for my_seq, 'Zukunft');`

# Automatische Zähler (3)

- Beispiel: Spalte, deren Inhalt beim **INSERT** automatisch als fortlaufende Nummer generiert wird (HSQLDB)

```
• create table moreandmore (
 id integer not null
 generated by default as identity
 (start with 1, increment by 1),
 name varchar(80)
);
-- automatische Nummernvergabe
insert into moreandmore (name) values ('Ein Name'); -- ID 1
-- manuelle Nummernvergabe
insert into moreandmore (id, name)
values (42, 'Ein Name'); -- ID 42
-- automatisch - zählt ab der höchsten ID weiter
insert into moreandmore (name) values ('Ein Name'); -- ID 43
```

# Trigger

- Ein Trigger ist ein in der Datenbank gespeichertes Programm, welches bei bestimmten Ereignissen ausgelöst wird.
- Die gängigsten Trigger sind an Tabellen installierte Trigger, die bei den DML Befehlen **INSERT**, **UPDATE** und **DELETE** aktiviert werden.

# Trigger (2)

- Eine **UPDATE** oder **DELETE** Anweisung kann mehrere Sätze ändern oder löschen. Generell wird daher zwischen Triggern unterschieden, die für jeden Datensatz ausgelöst werden (Row-Trigger), und solchen, die nur einmal für die gesamte Anweisung ausgelöst werden (Statement-Trigger).
- Der Zeitpunkt für das Auslösen eines Statement-Triggers kann festgelegt werden und entweder vor oder nach der Ausführung der eigentlichen Anweisung liegen.

# Trigger (3)

- Beispiele für Trigger
  - Einfügen fester Tabellenwerte wie Benutzerkennung oder Zeitstempel bei der Neuanlage von Tabellenzeilen
  - Protokolltabellen zum Speichern von Veränderungen