Ullman & al. @ Stanford

http://infolab.stanford.edu/~ullman/fcdb/aut07/index.html

Relational Algebra

Basic Operations

Algebra of Bags

What is an “Algebra”

- Mathematical system consisting of:
  - *Operands* --- variables or values from which new values can be constructed.
  - *Operators* --- symbols denoting procedures that construct new values from given values.

What is Relational Algebra?

- An algebra whose operands are relations or variables that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.
  - The result is an algebra that can be used as a *query language* for relations.

Core Relational Algebra

- Union, intersection, and difference.
  - Usual set operations, but *both operands must have the same relation schema*.
- Selection: picking certain rows.
- Projection: picking certain columns.
- Products and joins: compositions of relations.
- Renaming of relations and attributes.

Selection

- R1 := σ_C(R2)
  - C is a condition (as in “if” statements) that refers to attributes of R2.
  - R1 is all those tuples of R2 that satisfy C.

Example: Selection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

JoeMenu := $\sigma_{bar=\text{“Joe's”}}(Sells)$:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Example: Selection

```
select * from SELLS where BAR = 'Joe's'
```

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

JoeMenu := $\sigma_{bar=\text{“Joe's”}}(Sells)$:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Column and table names
with small and capital letters
and with blanks: double quotation marks

```
select * from "Sells" where "bar" = 'Joe's' ;
```

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

JoeMenu := $\sigma_{bar="Joe's"}(Sells)$:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Column and table names
with small and capital letters
and with blanks: double quotation marks

```
select * from "Sells" where "bar" = 'Joe's' ;
```

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

double quotes for
table and column names

single quotes for
string constants

JoeMenu := $\sigma_{bar=\text{“Joe's”}}(Sells)$:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Projection

- R1 := π_L (R2)
  - L is a list of attributes from the schema of R2.
  - R1 is constructed by looking at each tuple of R2, extracting the attributes on list L, in the order specified, and creating from those components a tuple for R1.
  - Eliminate duplicate tuples, if any.

Example: Projection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Prices := $\pi_{beer,price}$(Sells):

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

Example: Projection

select **distinct** "beer" , "price" from "Sells" ;

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Prices := $\Pi_{beer,price}$(Sells):

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

Extended Projection

- Using the same π_L operator, we allow the list L to contain arbitrary expressions involving attributes:
  1. Arithmetic on attributes, e.g., $A+B->C$.
  2. Duplicate occurrences of the same attribute.

Example: Extended Projection

R =

A	B
1	2
3	4

$\pi_{A+B->C,A,A}$ (R) =

C	A1	A2
3	1	1
7	3	3

Example: Extended Projection

```
select A + B as C , A as A1 , A as A2 from R ;
```

R =

A	B
1	2
3	4

$\pi_{A+B->C,A,A}$ (R) =

C	A1	A2
3	1	1
7	3	3

Example: Extended Projection

R =

A	B
1	2
3	4

$\pi_{A+B->C,A,A}$ (R) =

C	A1	A2
3	1	1
7	3	3

select "beer" + 3 , "price" + 3 from "Sells" ;

Product

- R3 := R1 X R2
  - Pair each tuple t1 of R1 with each tuple t2 of R2.
  - Concatenation t1t2 is a tuple of R3.
  - Schema of R3 is the attributes of R1 and then R2, in order.
  - But beware attribute *A* of the same name in R1 and R2: use R1.*A* and R2.*A*.

Example: R3 := R1 X R2

R1(

A,	B
1	2
3	4

)

R2(

B,	C
5	6
7	8
9	10

)

R3(

A,	R1.B,	R2.B,	C
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

)

Example: R3 := R1 X R2

```
select R1.* , R2.* from R1 , R2 ;
```

R1(

A,	B
1	2
3	4

)

R2(

B,	C
5	6
7	8
9	10

)

R3(

A,	R1.B,	R2.B,	C
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

)

Theta-Join

- R3 := R1 $\bowtie_C$ R2
  - Take the product R1 X R2.
  - Then apply σ_C to the result.
- As for σ, C can be any boolean-valued condition.
  - Historic versions of this operator allowed only A θ B, where θ is =, <, etc.; hence the name “theta-join.”

Example: Theta Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo(

bar,	beer,	price,	name,	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Coors	3.00	Sue's	River Rd.

)

Example: Theta Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo(

bar,	beer,	price,	name,	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.

)

```
select "Sells".*, "Bars".* from "Sells", "Bars"
where "Sells"."bar" = "Bars"."name"
```

Natural Join

- A useful join variant (*natural* join) connects two relations by:
  - Equating attributes of the same name, and
  - Projecting out one copy of each pair of equated attributes.
- Denoted R3 := R1 ⋈ R2.

Example: Natural Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells ⋈ Bars

Note: Bars.name has become Bars.bar to make the natural join "work."

BarInfo(

bar,	beer,	price,	addr
Joe's	Bud	2.50	Maple St.
Joe's	Milller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Coors	3.00	River Rd.

)

Example: Natural Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells ⋈ Bars

BarInfo := $\pi_{bar,beer,price,addr}(Sells \bowtie_{Sells.bar = Bars.bar} Bars)$

...tural join" work:

```
select "bar", "beer", "price", "addr" from "Sells", "Bars"
where "Sells"."bar" = "Bars"."bar"
```

Joe's	Miller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Coors	3.00	River Rd.

Renaming

- The ρ operator gives a new schema to a relation.
- R1 := $\rho_{R1(A1,\ldots,An)}$(R2) makes R1 be a relation with attributes A1,...,A*n* and the same tuples as R2.
- Simplified notation: R1(A1,...,A*n*) := R2.

Example: Renaming

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

R(bar, addr) := Bars

R(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

)

Example: Renaming

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

R(bar, addr) := Bars

R(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

)

```
select "name" as "bar" , "addr" from "Bars" ;
```

Building Complex Expressions

- Combine operators with parentheses and precedence rules.
- Three notations, just as in arithmetic:
  1. Sequences of assignment statements.
  2. Expressions with several operators.
  3. Expression trees.

Sequences of Assignments

- Create temporary relation names.
- Renaming can be implied by giving relations a list of attributes.
- Example: R3 := R1 $\bowtie_C$ R2 can be written:

  R4 := R1 X R2

  R3 := σ_C(R4)

Expressions in a Single Assignment

- Example: the theta-join R3 := R1 $\bowtie_C$ R2 can be written: R3 := σ_C (R1 X R2)
- Precedence of relational operators:
  1. [σ, π, ρ] (highest).
  2. [X, ⋈].
  3. ∩.
  4. [∪, —]

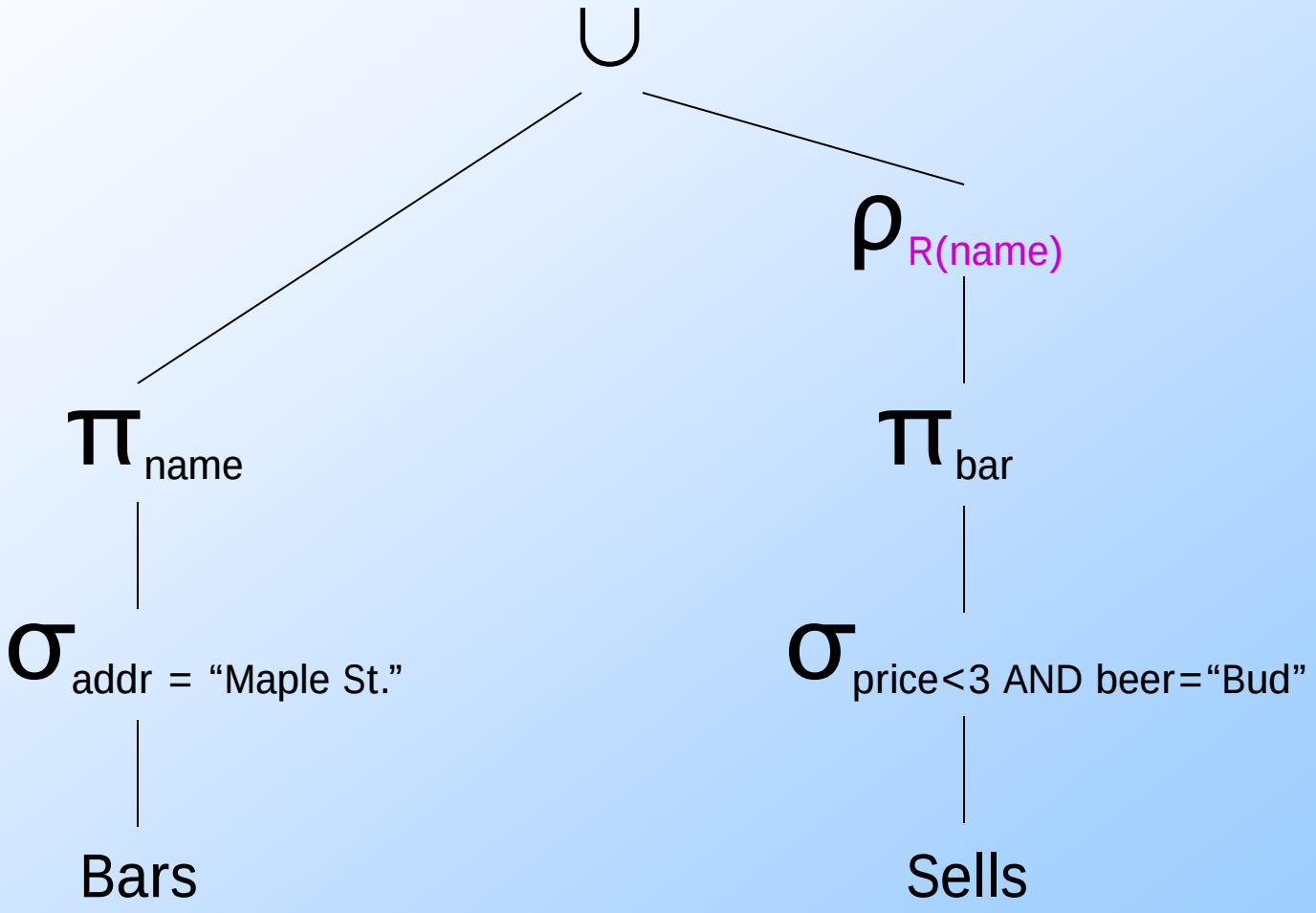
Expression Trees

- Leaves are operands --- either variables standing for relations or particular, constant relations.
- Interior nodes are operators, applied to their child or children.

Example: Tree for a Query

- Using the relations Bars(name, addr) and Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3.

As a Tree:


∪
ρ R(name)
π name
π bar
σ addr = “Maple St.”
σ price<3 AND beer=“Bud”
Bars
Sells

UNION

```
(select * from "Sells" where "beer" = 'Bud')
union
(select * from "Sells" where "price" < 3)
```

```
select * from "Sells" where "beer" = 'Bud' or "price" < 3
```

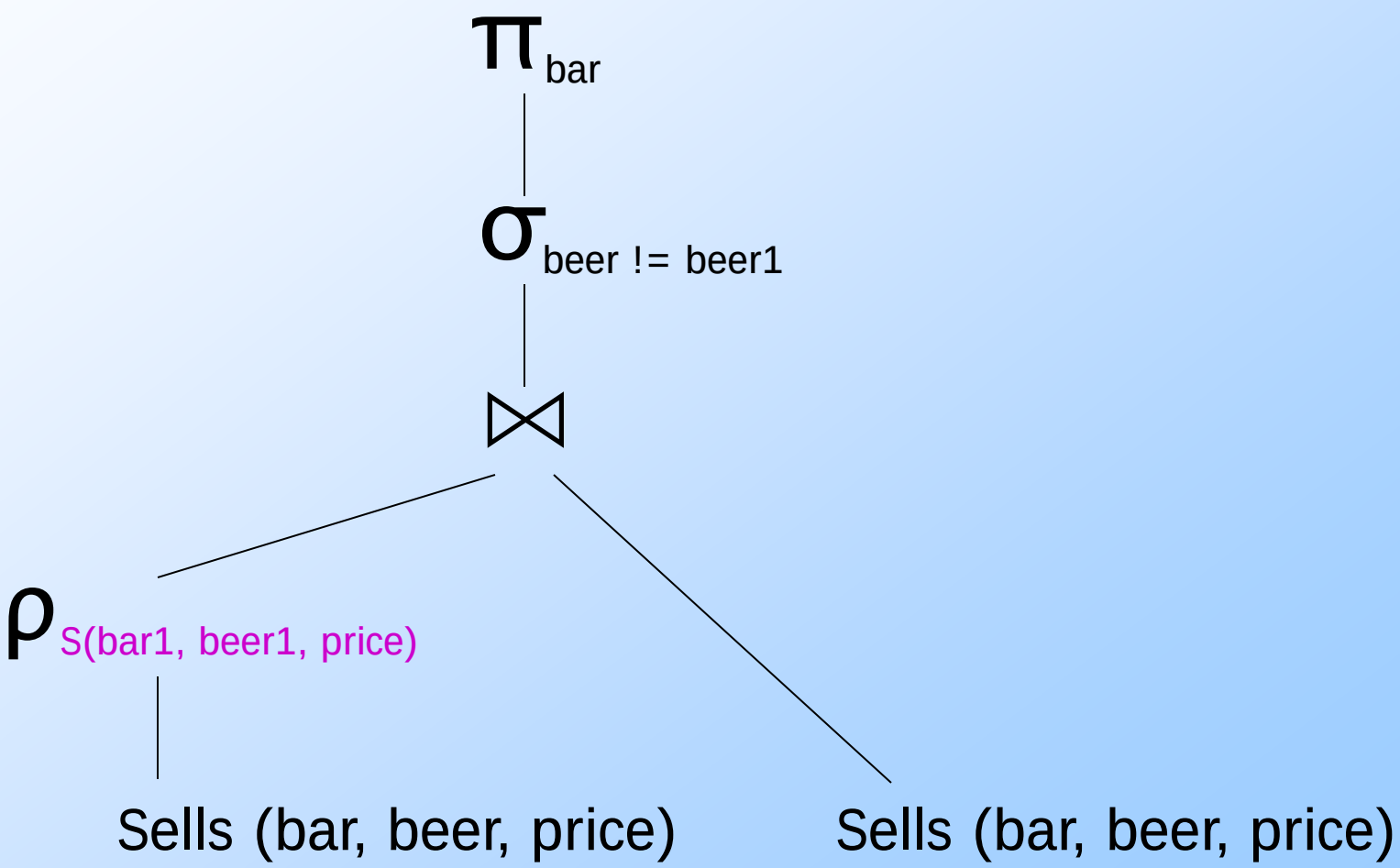
```
(select * from "Sells" where "beer" = 'Bud')
union all
(select * from "Sells" where "price" < 3)
```

```
select "name" as "bar" from "Bars"
union all
select "bar" from "Sells"
```

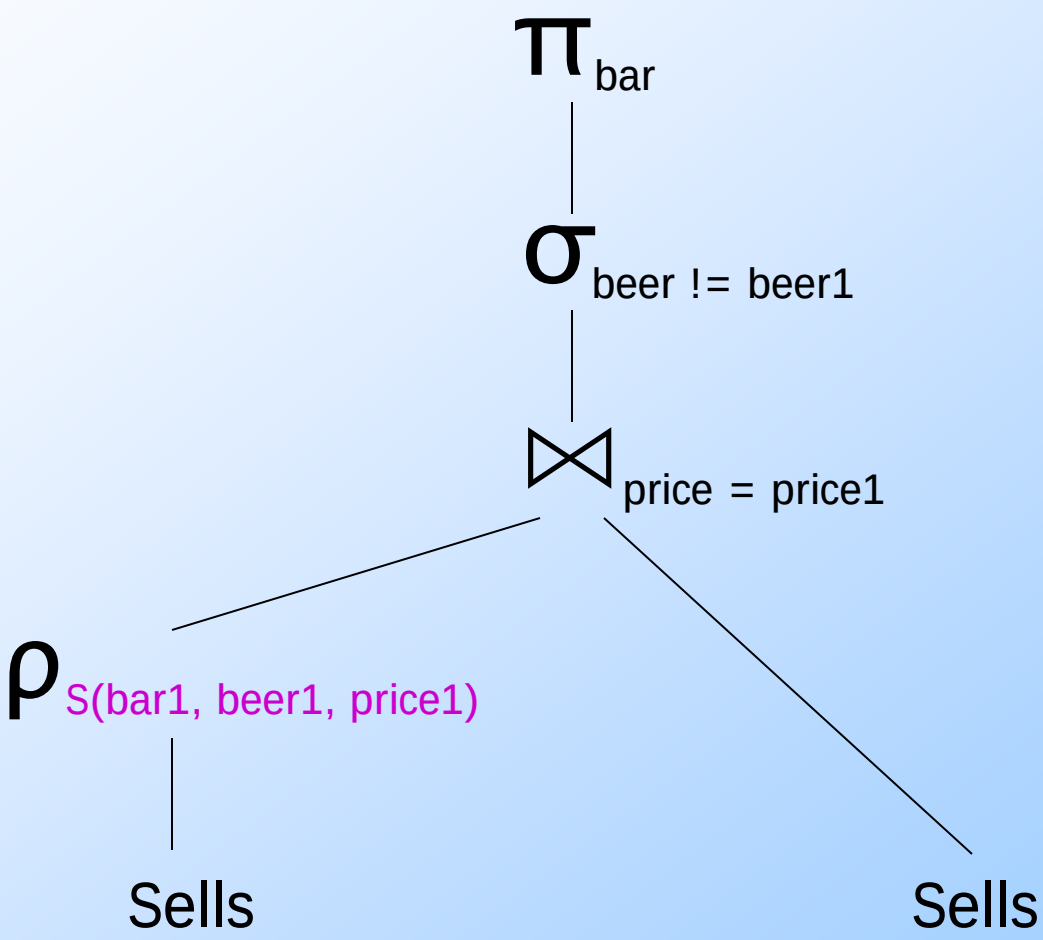
Example: Self-Join

- Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.
- Strategy: by renaming, define a copy of Sells, called S(bar, beer1, price). The natural join of Sells and S consists of quadruples (bar, beer, beer1, price) such that the bar sells both beers at this price.

The Tree


π bar
σ beer != beer1
⋈
ρ S(bar1, beer1, price)
Sells (bar, beer, price)
Sells (bar, beer, price)

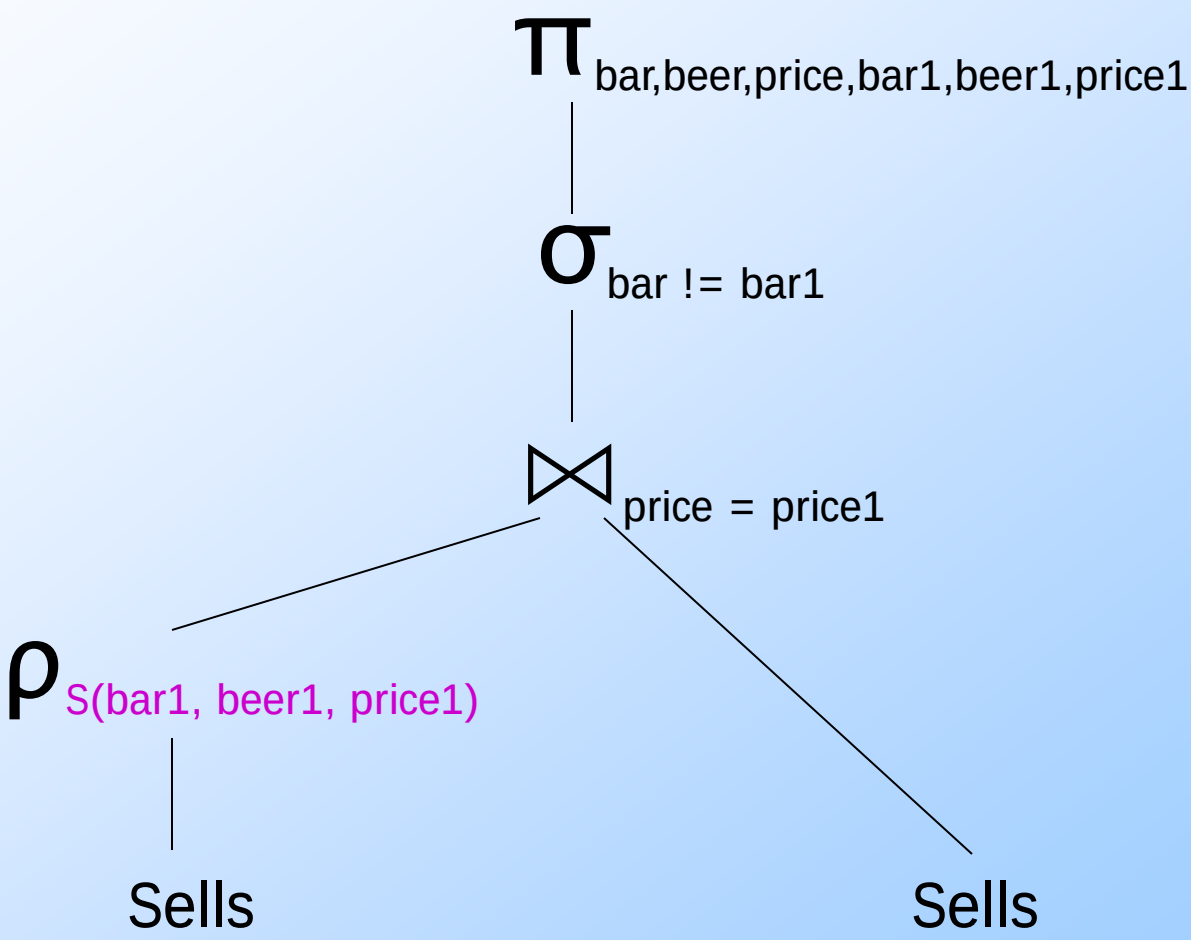
The Tree


π bar
σ beer != beer1
⋈ price = price1
ρ S(bar1, beer1, price1)
Sells
Sells

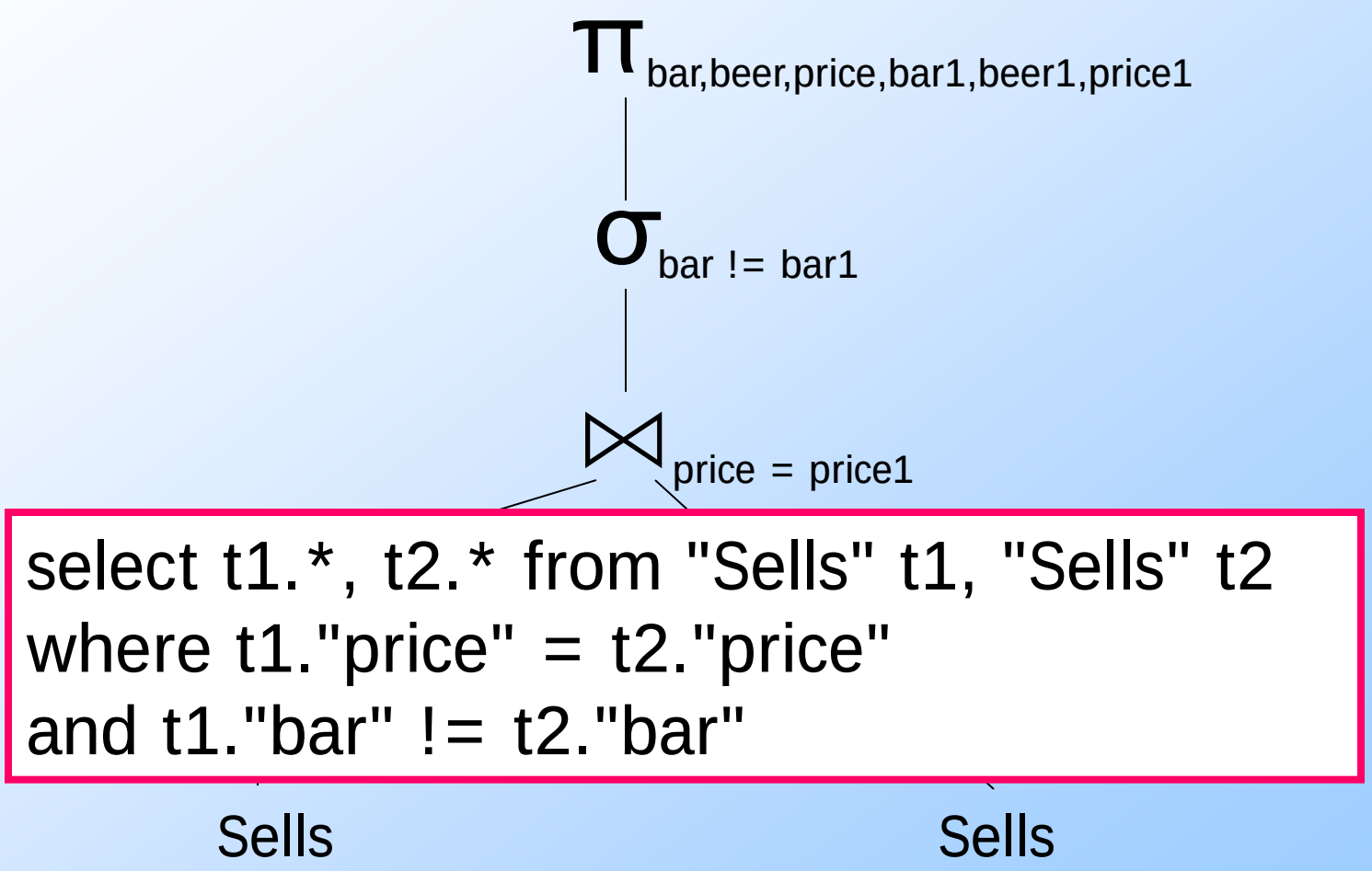
Example: Self-Join 2

- Using Sells(bar, beer, price), find prices at which two different bars offer beer.
- Strategy: by renaming, define a copy of Sells, called S(bar1, beer1, price). The theta join consists of all beers having the same price. Bars must be different, i.e. bar1 ≠ bar.

The Tree


π bar,beer,price,bar1,beer1,price1
σ bar != bar1
⋈ price = price1
ρ S(bar1, beer1, price1)
Sells
Sells

The Tree


π bar,beer,price,bar1,beer1,price1
σ bar != bar1
⋈ price = price1
select t1.*, t2.* from "Sells" t1, "Sells" t2
where t1."price" = t2."price"
and t1."bar" != t2."bar"
Sells
Sells

„group by“ Clause

- First group data and then aggregate: sum(), count(), avg() ...

- Average beer price in the bars:

```
select "bar", avg("price")
from "Sells"
group by "bar"
```

- Average price of beer brands:

```
select "beer", avg("price")
from "Sells"
group by "beer"
```

Schemas for Results

- Union, intersection, and difference: the schemas of the two operands must be the same, so use that schema for the result.
- Selection: schema of the result is the same as the schema of the operand.
- Projection: list of attributes tells us the schema.

Schemas for Results --- (2)

- Product: schema is the attributes of both relations.
  - Use R.*A*, etc., to distinguish two attributes named *A*.
- Theta-join: same as product.
- Natural join: union of the attributes of the two relations.
- Renaming: the operator tells the schema.

Relational Algebra on Bags

- A *bag* (or *multiset*) is like a set, but an element may appear more than once.
- Example: {1,2,1,3} is a bag.
- Example: {1,2,3} is also a bag that happens to be a set.

Why Bags?

- SQL, the most important query language for relational databases, is actually a bag language.
- Some operations, like projection, are more efficient on bags than sets.

Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets.
- Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates.
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Example: Bag Selection

R(

A,	B
1	2
5	6
1	2

)

$\sigma_{A+B<5}(R) =$

A	B
1	2
1	2

Example: Bag Projection

R(

A,	B
1	2
5	6
1	2

)

π_A (R) =

A
1
5
1

Example: Bag Product

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R X S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	3	4
5	6	7	8
1	2	3	4
1	2	7	8

Example: Bag Theta-Join

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

$R \bowtie_{R.B<S.B} S =$

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	7	8
1	2	3	4
1	2	7	8

Bag Union

- An element appears in the union of two bags the sum of the number of times it appears in each bag.
- Example: $\{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$

Bag Intersection

- An element appears in the intersection of two bags the minimum of the number of times it appears in either.
- Example: {1,2,1,1} ∩ {1,2,1,3} = {1,1,2}.

Bag Difference

- An element appears in the difference $A - B$ of bags as many times as it appears in A, minus the number of times it appears in B.
  - But never less than 0 times.
- Example: {1,2,1,1} – {1,2,3} = {1,1}.

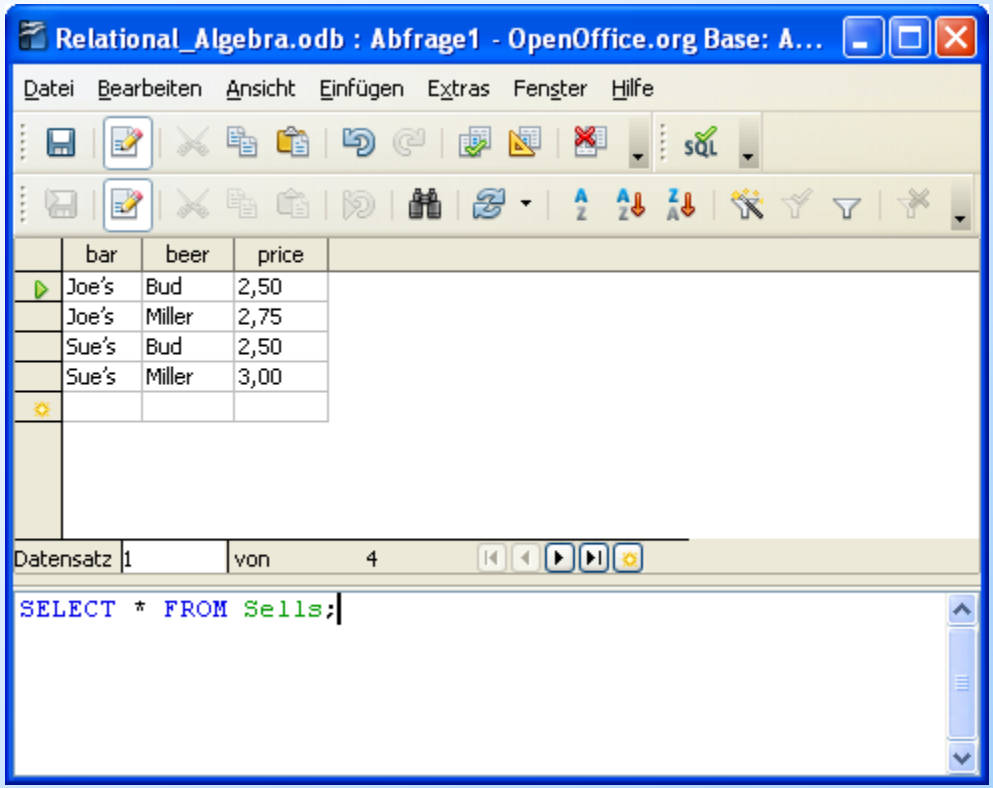
Beware: Bag Laws != Set Laws

- Some, but *not all* algebraic laws that hold for sets also hold for bags.
- Example: the commutative law for union ($R \cup S = S \cup R$) *does* hold for bags.
  - Since addition is commutative, adding the number of times x appears in R and S doesn't depend on the order of R and S.

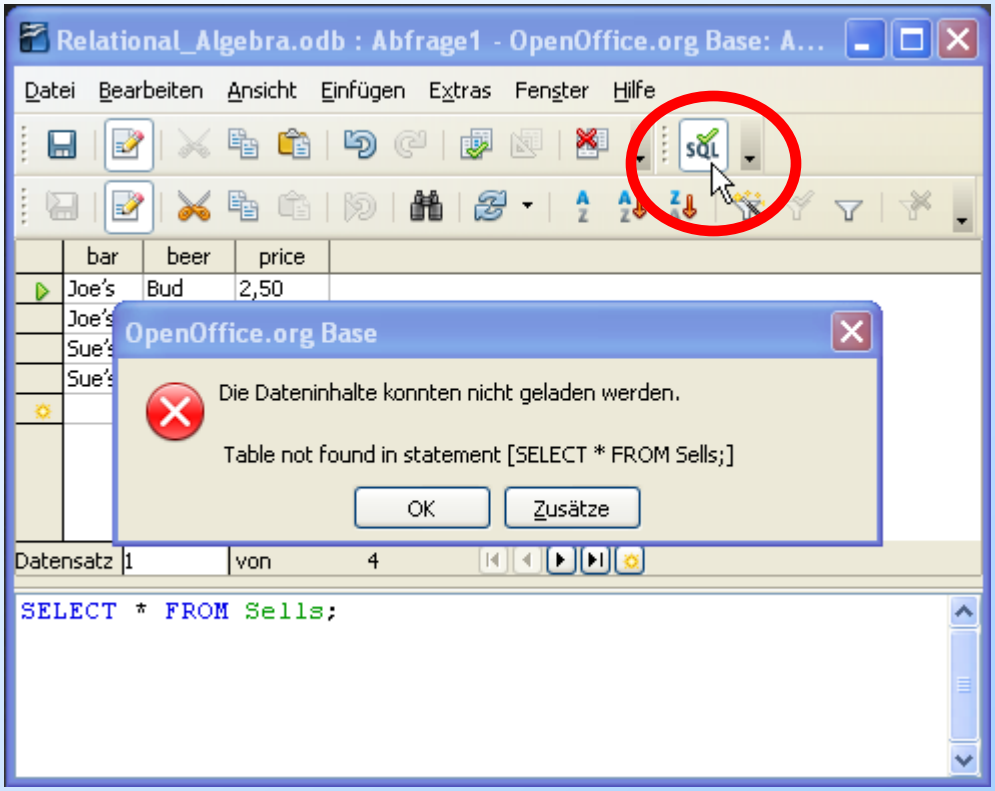
Example: A Law That Fails

- Set union is *idempotent*, meaning that $S \cup S = S$.
- However, for bags, if x appears n times in S, then it appears $2n$ times in $S \cup S$.
- Thus $S \cup S$!= S in general.
  - e.g., {1} ∪ {1} = {1,1} != {1}.

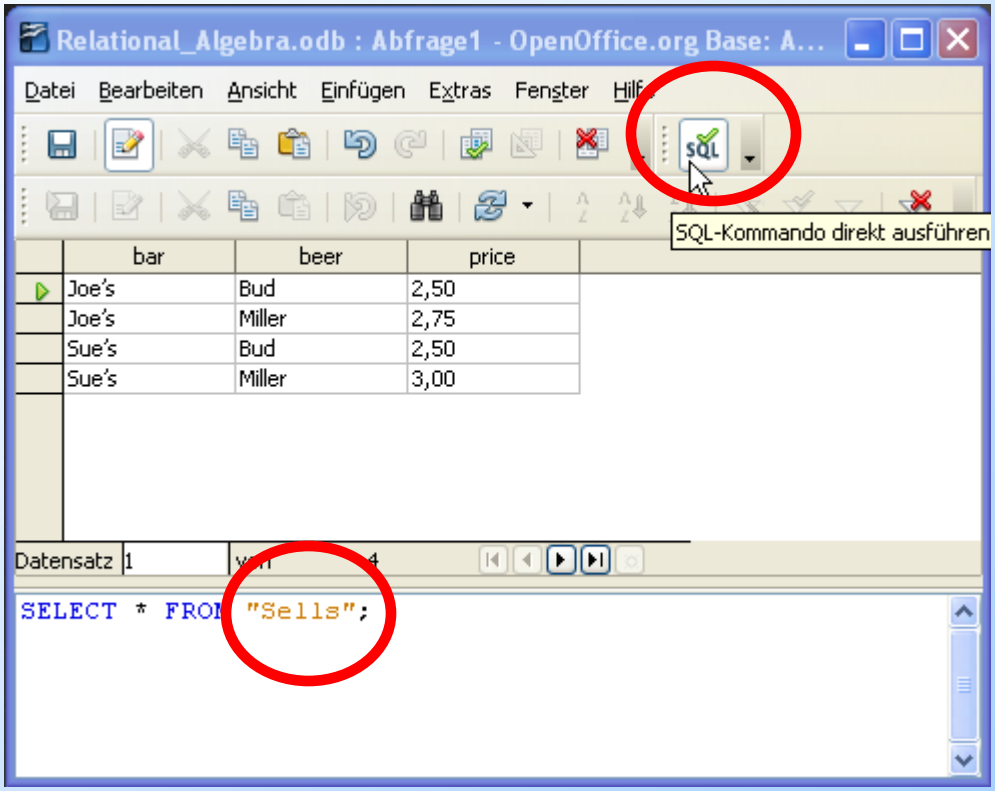
Simple selection



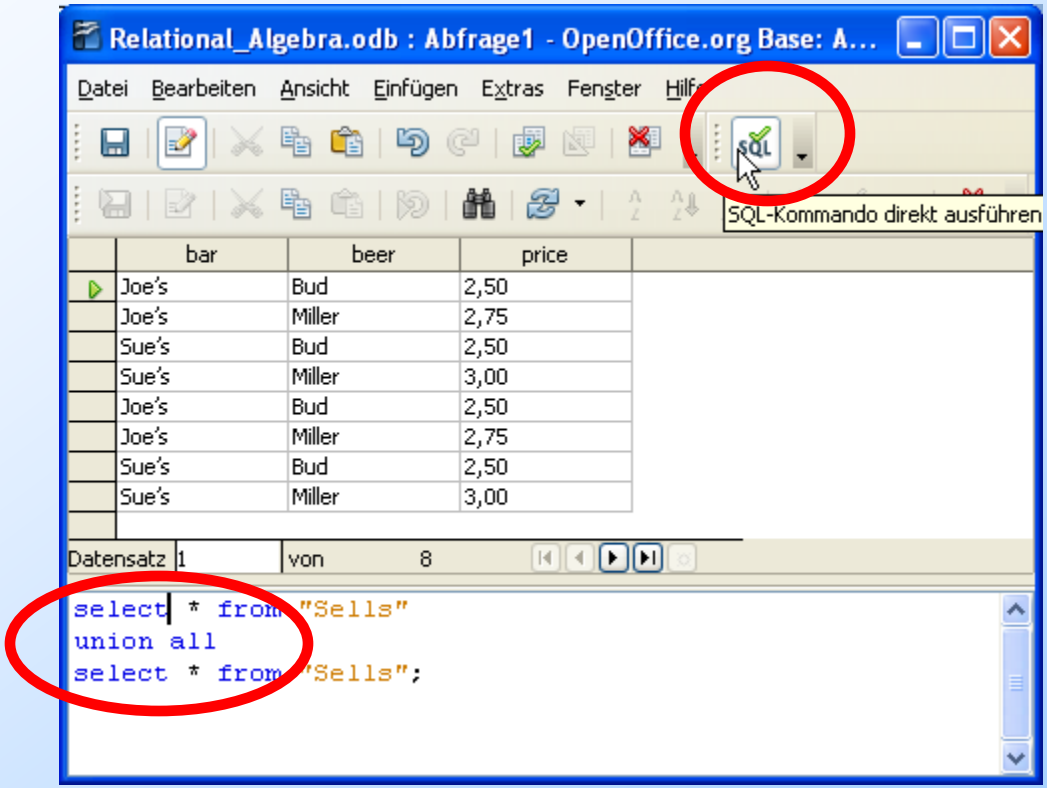
Same selection fails in native SQL mode: table name not recognized.



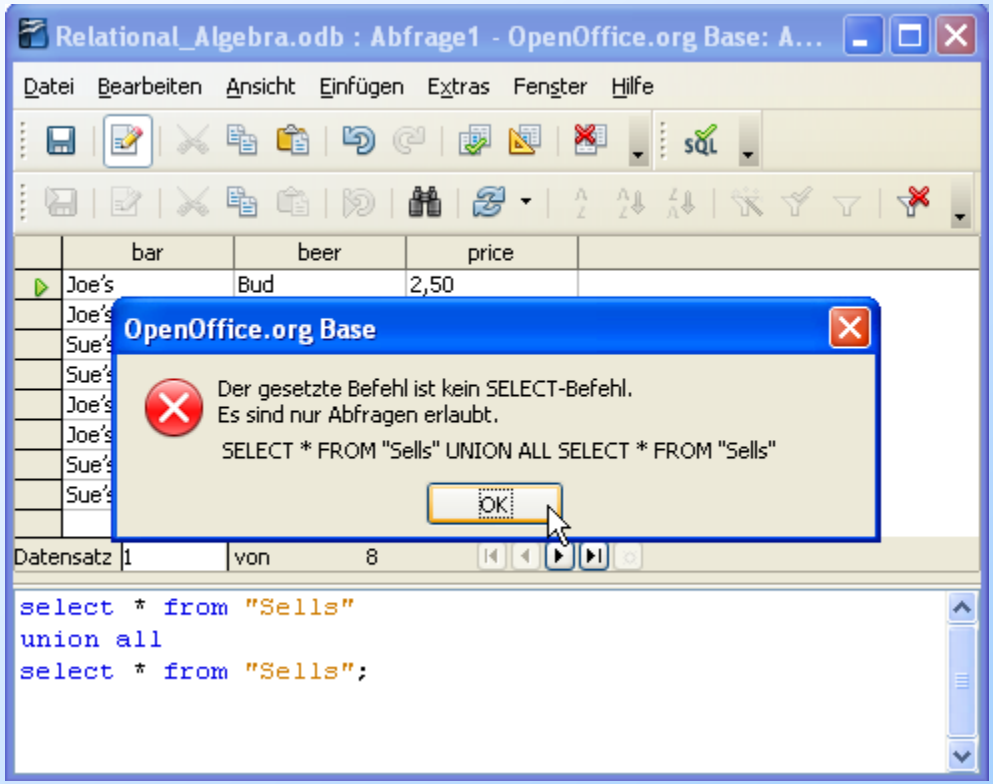
Selection with quotation marks works in native SQL mode.



UNION and UNION ALL only work in native SQL mode.



UNION and UNION ALL do not work in design mode.



Native SQL mode vs. design mode

Sie können Ihre Abfrage direkt im SQL-Code formulieren. Dabei ist jedoch zu beachten, dass die spezielle Syntax von Ihrem verwendeten Datenbank-system abhängt.
Wenn Sie den SQL-Code von Hand eingeben, können Sie SQL-spezifische Abfragen anlegen, die nicht von der grafischen Oberfläche im Abfrageentwurf unterstützt werden. Diese Abfragen müssen im Native-SQL-Modus ausgeführt werden.

(OpenOffice Base help)

Native SQL mode vs. design mode

Im Native-SQL-Modus haben Sie die Möglichkeit, SQL-Befehle einzugeben, die nicht von OpenOffice.org interpretiert, sondern direkt an die Datenquelle übergeben werden. Wenn Sie diese Änderungen nicht in der Entwurfsansicht anzeigen, kann nicht mehr in die Entwurfsansicht zurückgeschaltet werden.
Bei einem Native SQL wird der SQL-String direkt an das angeschlossene Datenbank-System weitergegeben, ohne vorher vom OpenOffice.org ausgewertet zu werden. Greifen Sie beispielsweise per ODBC-Schnittstelle auf eine Datenbank zu, dann wird der SQL-String an den ODBC-Treiber übergeben und von diesem verarbeitet.

Some SQL commands

```
DESIGN MODE
1. select bar,beer,price,addr from Sells,Bars where bar=name;
2. select * from Sells, Bars where bar=name;
3. select Sells.bar, Bars.* from Sells, Bars;
4. select t1.*, t2.* from Sells t1, Bars t2 where t1.bar=t2.name;
5. select * from Sells where name like 'Jo%';
6. select * from Sells where name = 'Joe's';
7. select * from Sells where name = ' Joe's'; ↓ blank in const.
8. select t1.*,t2.* from Sells t1, Sells t2;
   where t1.price = t2.price and t1.bar != t2.bar;
9. select beer,bar,price from Sells where price between 2 and 2.6;
10.select beer,bar,price from Sells
   where price >= 2 and price <= 2.6; (same as above)
```

Some SQL commands

```
DRUNKEN PIRATE BREWERY
1. SELECT "Salary Type", SUM("Salary Amount")
   FROM "Employee" GROUP BY "Salary Type";
2. SELECT "Customer State", count("Customer State")
   FROM "Customer" group by "Customer State";
```

Exercises with
For Ladies Database

Paste table "Customer"

1. Open "For_Ladies_Imported.odb" in OpenOffice Base
2. Open "For_Ladies_Decomposed.ods" in OpenOffice Calc
3. Select in OOCalc tuples of "Customer" including heading, copy (<Ctrl>-C)
4. Change to OOBase and paste (<Ctrl>-V): Table name "Customer", all columns, no special column settings, no primary key.

Paste table "Customer"

5. In OOBase right klick on table "Customer" and select "edit" ("bearbeiten")
6. Change data type of attribute "Customer_ID" to integer[INTEGER]
7. Select primary key: Right klick on the row heading (left most column) of row "Customer_ID" and select "primary key" from context menu.
8. Save table changes.

Paste table "Employee"

1. Follow the same procedure as with table "Customer".

Paste table "Product"

1. In OOCalc change numeric format of cells to 'English (USA)' with decimal separator '.'
2. Copy-Paste to OOBase.
3. Change attribute types where necessary and select primary key. Use attribute type "float" for floating point numbers and "integer" for integer.

Paste table "Order"

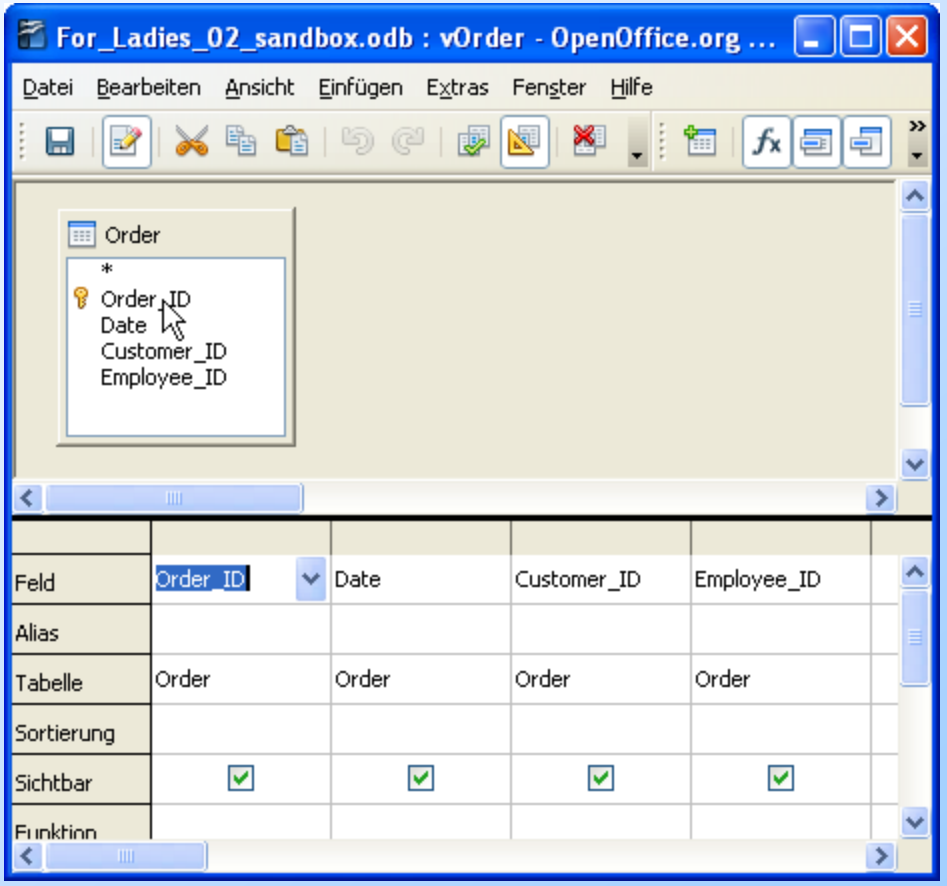
1. Change date format of cells to ISO: 'JJJJ-MM-TT'
2. Copy/paste to OOBase.
3. Change attribute types where necessary and select primary key.

Paste table "Order Detail"

1. Copy-Paste to OOBase.
2. Change attribute types where necessary.
3. Primary key is composed of two attributes (Order_ID, Product_ID): select both lines (holding <Ctrl>-key down), right klick and choose primary key.

Create view ("Abfrage") vOrder in design mode

vOrder (Order_ID , Date , Customer_ID , Employee_ID)

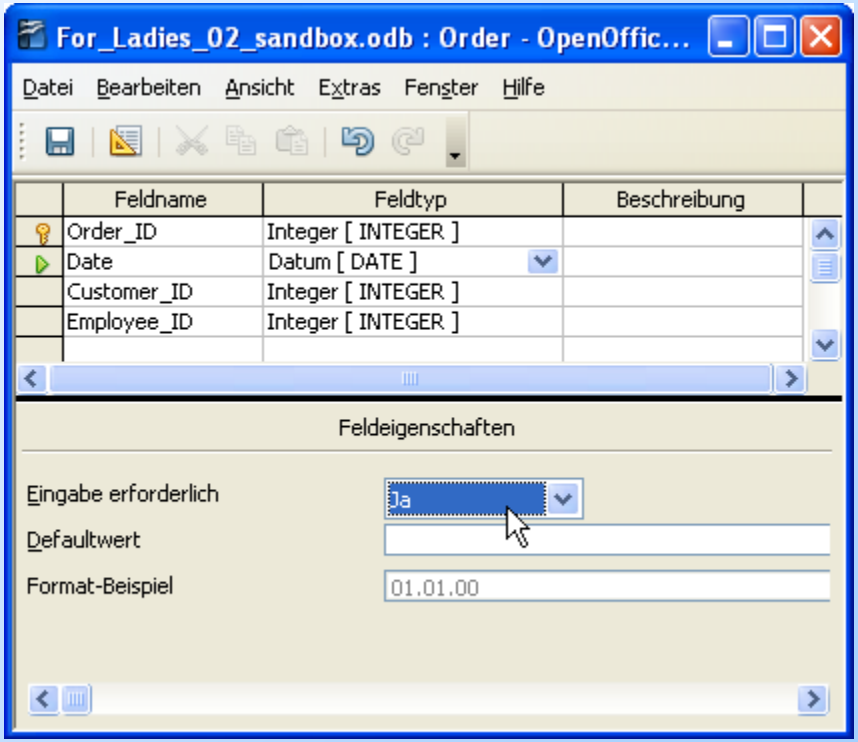


Insert new "Order"

1. In table Order (or view vOrder) klick on the last empty line and insert the following tuples (press <enter> after each new line):
   (9 , , 4 , 4)
   (10 , , ,)
2. What do you experience?
   What do you think?

Add constraints to table "Order" for integrity and correctness

HSQLDB, direct SQL:
alter table "Order" alter column "Date" set not null

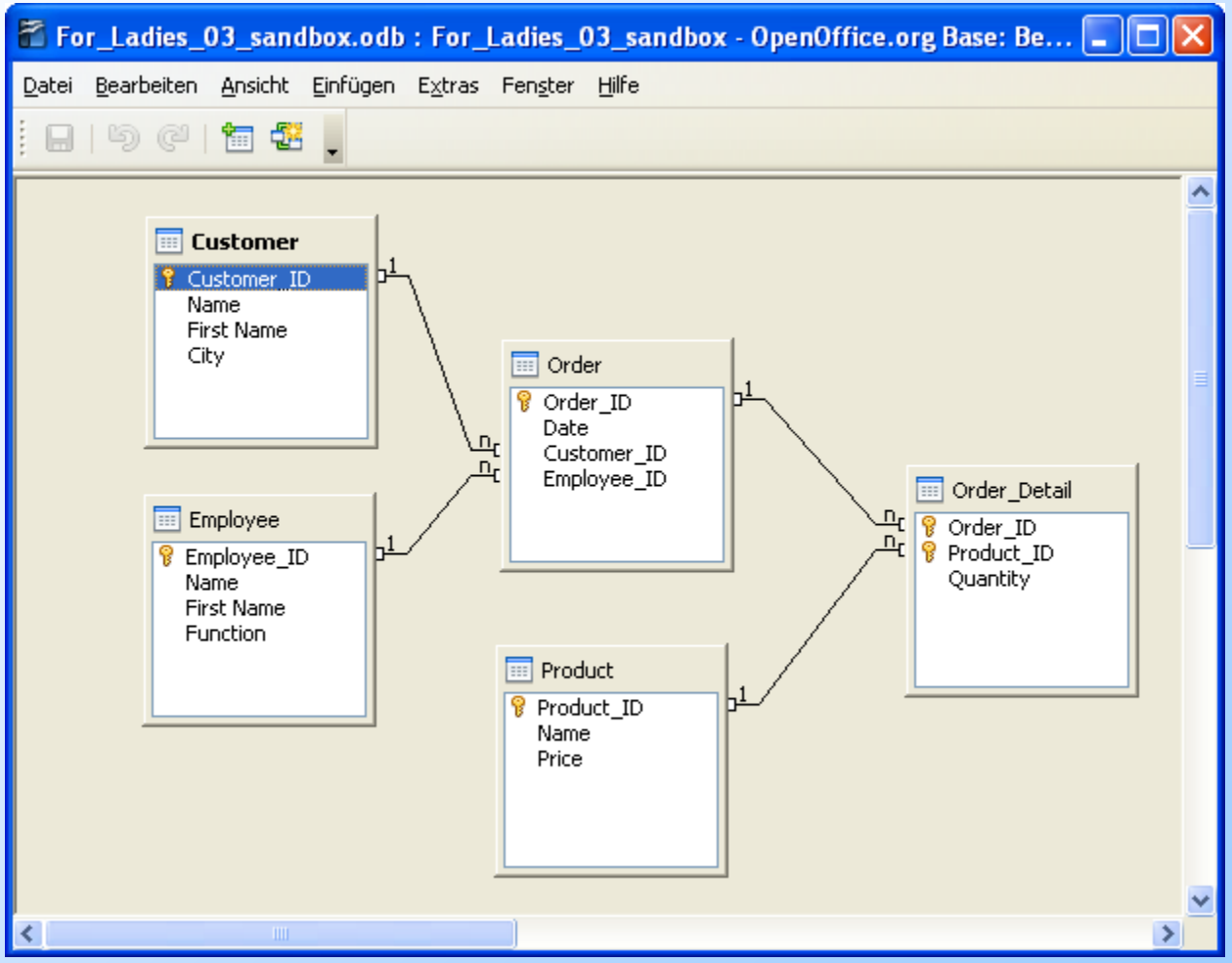


Insert new "Order"

1. Insert the following tuple:
   (11 , 1.12.09 , 5 , 5)

2. What do you experience?
   What do you think?

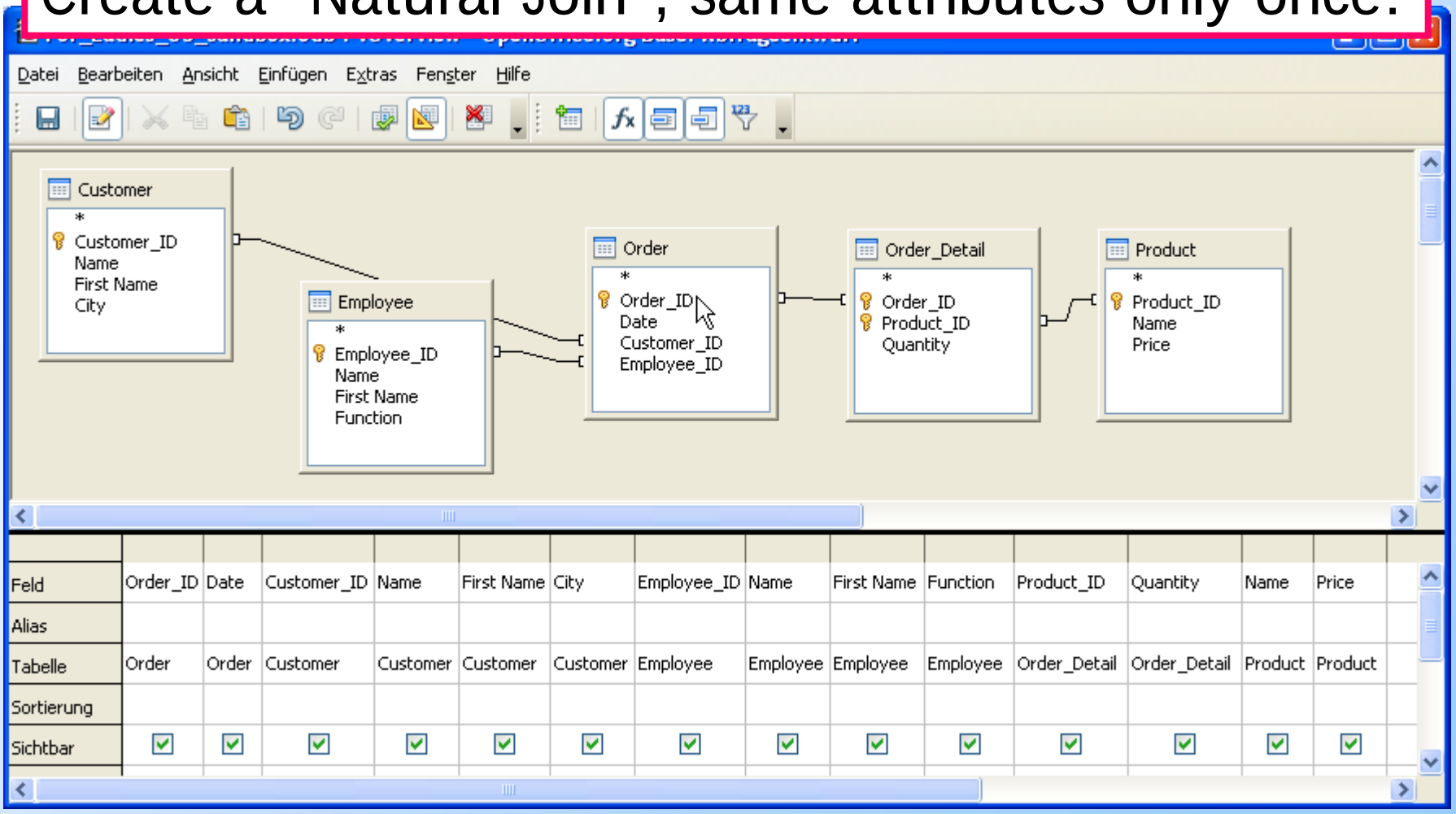
Referential Integrity: Relationships (Beziehungen)



Let's start analysis

Create view "vOverview" with all tables and distinct columns

Create a "Natural Join", same attributes only once:

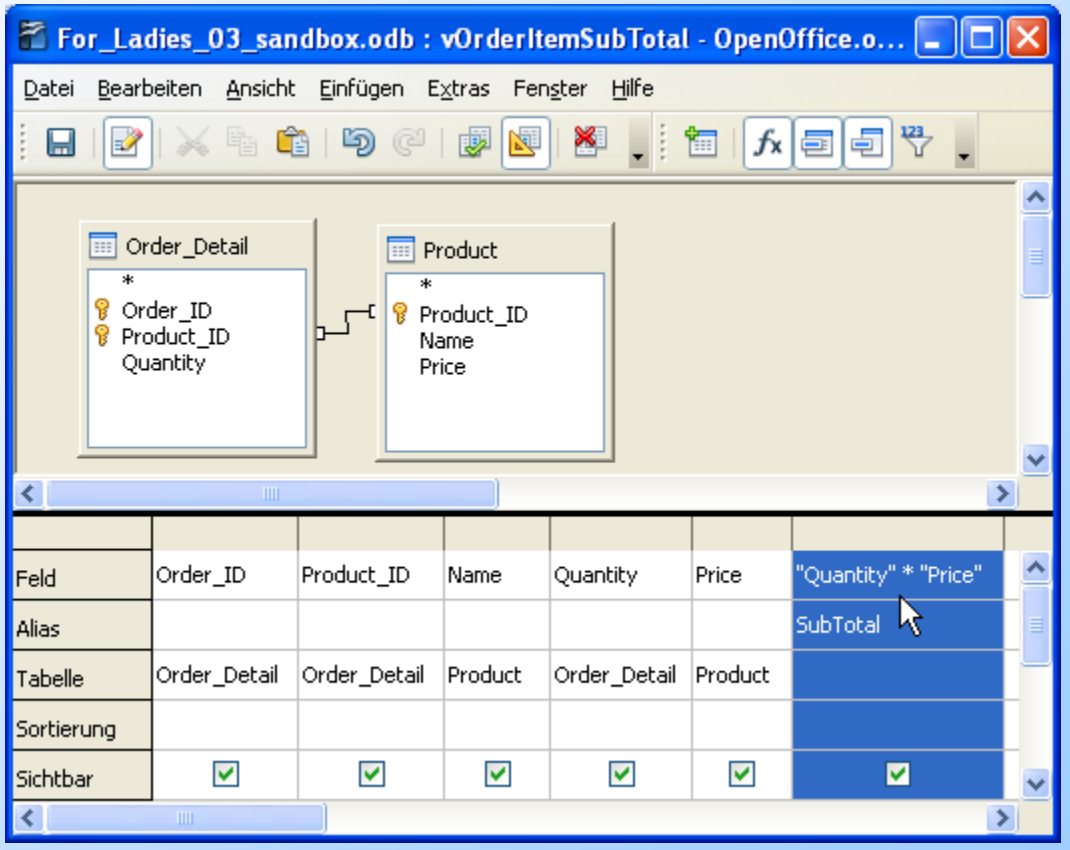


vOverview

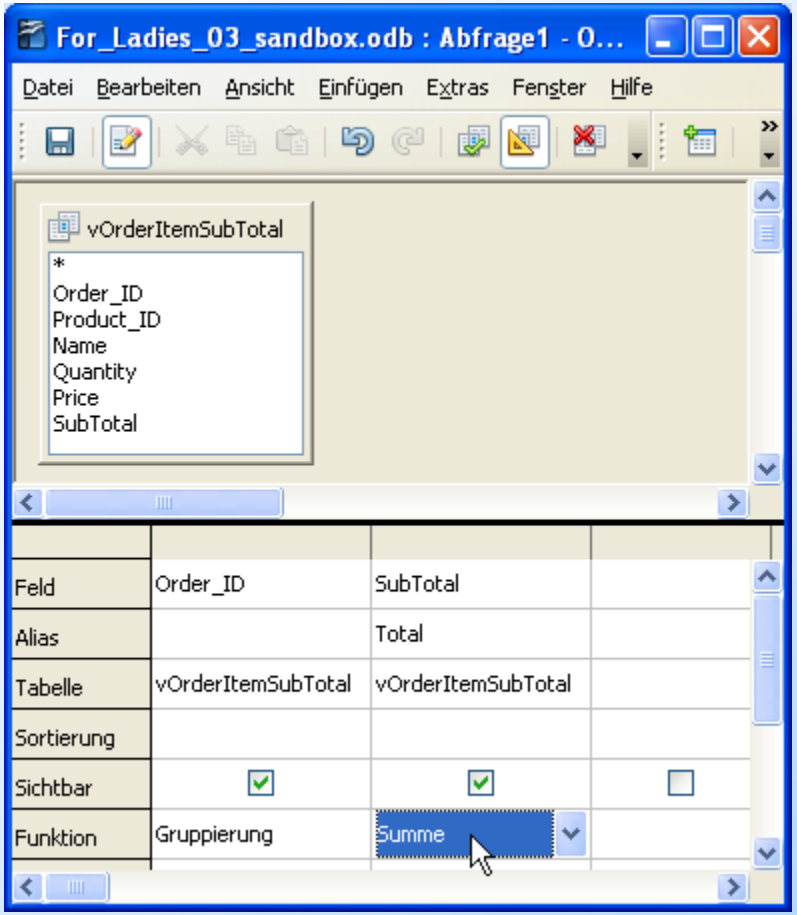
- vOverview looks like initial spreadsheet.
- What is the difference?
- What have we accomplished?
- Check anomalies …

Create view "vOrderItemSubTotal": extended projection

"Quantity" * "Price" as "SubTotal"



Create view "vOrderTotal": "group by"-clause




```
select "Order_ID",
sum("SubTotal") as "Total"
from "vOrderItemSubTotal"
group by "Order_ID"
```

Now we have:

- vOverview: like initial spreadsheet start
- vOrderItemSubTotal: Price * Quantity
- vOrderTotal: Total of each order

- Generate a list of orders with customer, employee, and total of each order.

Play with referential integrity

1. Insert new customer.
2. Caputure an order (Order ID 111)
   for this customer.
3. Fill the order with products.
4. Look at the data with view "vOverview".
5. Delete Order 111.
6. What happened to order details?
   What happened to the products? Explain!

Let's create a simple DB application: Form ("Formular")

- Create view vOrderCustEmp comprising the tables Order, Customer and Employee.

- Together we create the form based on vOrderCustEmp with subform vOrderItemTotal