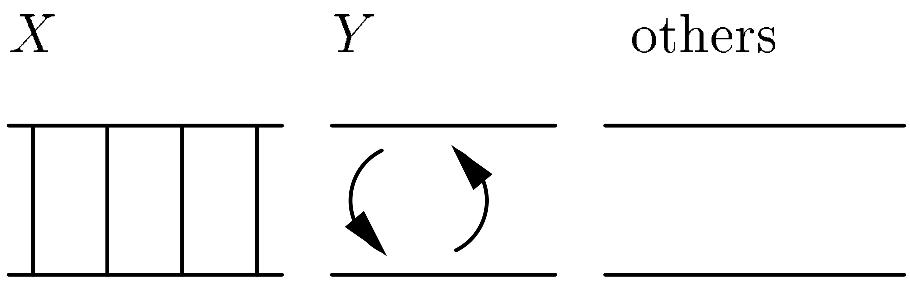
## Multivalued Dependencies

The *multivalued dependency* $X \twoheadrightarrow Y$ holds in a relation $R$ if whenever we have two tuples of $R$ that agree in all the attributes of $X$, then we can swap their $Y$ components and get two new tuples that are also in $R$.



## Example

`Drinkers(name, addr, phones,beersLiked)` with MVD `name` $\twoheadrightarrow$ `phones`. If `Drinkers` has the two tuples:

| name | addr | phones | beersLiked |
|---|---|---|---|
| sue | $a$ | $p1$ | $b1$ |
| sue | $a$ | $p2$ | $b2$ |

it must also have the same tuples with `phones` components swapped:

| name | addr | phones | beersLiked |
|---|---|---|---|
| sue | $a$ | $p1$ | $b2$ |
| sue | $a$ | $p2$ | $b1$ |

- Note: we must check this condition for *all* pairs of tuples that agree on `name`, not just one pair.

## MVD Rules

1. Every FD is an MVD.

   ✦ Because if $X \rightarrow Y$, then swapping $Y$'s between tuples that agree on $X$ doesn't create new tuples.

   ✦ Example, in `Drinkers`: `name` $\twoheadrightarrow$ `addr`.

2. *Complementation*: if $X \twoheadrightarrow Y$, then $X \twoheadrightarrow Z$, where $Z$ is all attributes not in $X$ or $Y$.

   ✦ Example: since `name` $\twoheadrightarrow$ `phones` holds in `Drinkers`, so does `name` $\twoheadrightarrow$ `addr beersLiked`.

## Splitting Doesn't Hold

Sometimes you need to have several attributes on the right of an MVD. For example:

`Drinkers(name, areaCode, phones, beersLiked, beerManf)`

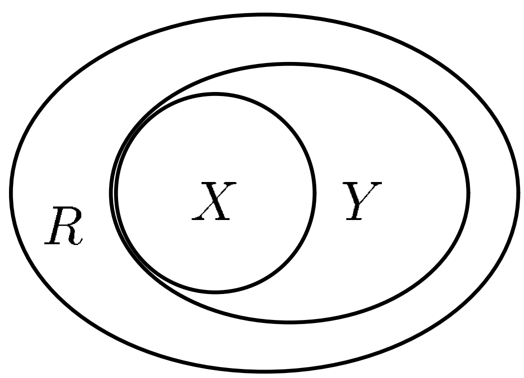
| name | areaCode | phones | BeersLiked | beerManf |
|---|---|---|---|---|
| Sue | 650 | 555-1111 | Bud | A.B. |
| Sue | 650 | 555-1111 | WickedAle | Pete's |
| Sue | 415 | 555-9999 | Bud | A.B. |
| Sue | 415 | 555-9999 | WickedAle | Pete's |

- `name` $\rightarrow\!\!\rightarrow$ `areaCode phones` holds, but neither `name` $\rightarrow\!\!\rightarrow$ `areaCode` nor `name` $\rightarrow\!\!\rightarrow$ `phones` do.

## 4NF

Eliminate redundancy due to multiplicative effect of MVD's.

- Roughly: treat MVD's as FD's for decomposition, but not for finding keys.
- Formally: $R$ is in Fourth Normal Form if whenever MVD $X \twoheadrightarrow Y$ is *nontrivial* ($Y$ is not a subset of $X$, and $X \cup Y$ is not all attributes), then $X$ is a superkey.
    - ✦ Remember, $X \rightarrow Y$ implies $X \twoheadrightarrow Y$, so 4NF is more stringent than BCNF.
- Decompose $R$, using 4NF violation $X \twoheadrightarrow Y$, into $XY$ and $X \cup (R - Y)$.



## Example

`Drinkers(name, addr, phones, beersLiked)`

- FD: `name` $\rightarrow$ `addr`
- Nontrivial MVD's: `name` $\twoheadrightarrow$ `phones` and `name` $\twoheadrightarrow$ `beersLiked`.
- Only key: {`name`, `phones`, `beersLiked`}
- All three dependencies above violate 4NF.
- Successive decomposition yields 4NF relations:

  D1(<u>name</u>, addr)
  D2(<u>name</u>, <u>phones</u>)
  D3(<u>name</u>, <u>beersLiked</u>)

## Relational Algebra

A small set of operators that allow us to manipulate relations in limited but useful ways. The operators are:

1. Union, intersection, and difference: the usual set operators.
   - ✦ But the relation schemas must be the same.
2. *Selection*: Picking certain rows from a relation.
3. *Projection*: Picking certain columns.
4. *Products and joins*: Composing relations in useful ways.
5. *Renaming* of relations and their attributes.

## Selection

$$R_1 = \sigma_C(R_2)$$

where $C$ is a condition involving the attributes of relation $R_2$.

## Example

Relation `Sells`:

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

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

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

## Projection

$$R_1 = \pi_L(R_2)$$

where $L$ is a list of attributes from the schema of $R_2$.

## Example

$\pi_{beer,price}$(`Sells`)

| beer | price |
| --- | --- |
| Bud | 2.50 |
| Miller | 2.75 |
| Coors | 3.00 |

- Notice elimination of duplicate tuples.

**Product**

$$R = R_1 \times R_2$$

pairs each tuple $t_1$ of $R_1$ with each tuple $t_2$ of $R_2$ and puts in $R$ a tuple $t_1 t_2$.

**Theta-Join**

$$R = R_1 \underset{C}{\bowtie} R_2$$

is equivalent to $R = \sigma_C(R_1 \times R_2)$.

## Example

`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` $\underset{Sells.Bar=Bars.Name}{\bowtie}$ `Bars`

| 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. |

## Natural Join

$$R = R_1 \bowtie R_2$$

calls for the theta-join of $R_1$ and $R_2$ with the condition that all attributes of the same name be equated. Then, one column for each pair of equated attributes is projected out.

## Example

Suppose the attribute `name` in relation `Bars` was changed to `bar`, to match the bar name in `Sells`.

`BarInfo = Sells` $\bowtie$ `Bars`

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

## Renaming

$\rho_{S(A_1,\ldots,A_n)}(R)$ produces a relation identical to $R$ but named $S$ and with attributes, in order, named $A_1, \ldots, A_n$.

## Example

`Bars` =

| name | addr |
|---|---|
| Joe's | Maple St. |
| Sue's | River Rd. |

$\rho_{R(bar,addr)}(\texttt{Bars})$ =

| bar | addr |
|---|---|
| Joe's | Maple St. |
| Sue's | River Rd. |

- The name of the above relation is $R$.

**Combining Operations**

Algebra =

1. Basis arguments +
2. Ways of constructing expressions.

For relational algebra:

1. Arguments = variables standing for relations + finite, constant relations.
2. Expressions constructed by applying one of the operators + parentheses.

- Query = expression of relational algebra.

## Operator Precedence

The normal way to group operators is:

1. Unary operators $\sigma$, $\pi$, and $\rho$ have highest precedence.

2. Next highest are the "multiplicative" operators, $\bowtie$, $\underset{C}{\bowtie}$, and $\times$.

3. Lowest are the "additive" operators, $\cup$, $\cap$, and $-$.

- But there is no universal agreement, so we always put parentheses *around* the argument of a unary operator, and it is a good idea to group all binary operators with parentheses *enclosing* their arguments.

## Example

Group $R \cup \sigma S \bowtie T$ as $R \cup (\sigma(S) \bowtie T)$.

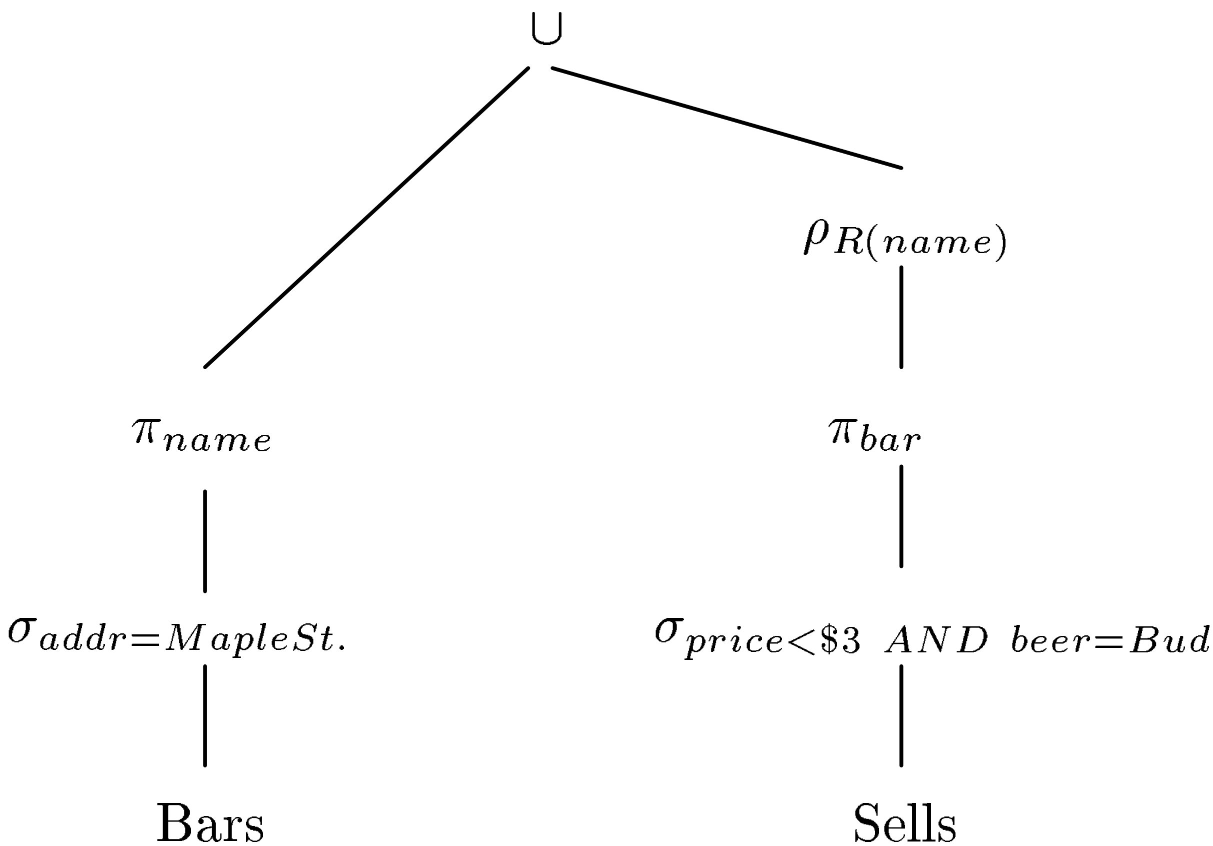
## Each Expression Needs a Schema

- If $\cup$, $\cap$, $-$ applied, schemas are the same, so use this schema.
- Projection: use the attributes listed in the projection.
- Selection: no change in schema.
- Product $R \times S$: use attributes of $R$ and $S$.
  - But if they share an attribute $A$, prefix it with the relation name, as $R.A$, $S.A$.
- Theta-join: same as product.
- Natural join: use attributes from each relation; common attributes are merged anyway.
- Renaming: whatever it says.

## Example

Find the bars that are either on Maple Street or sell Bud for less than $3.

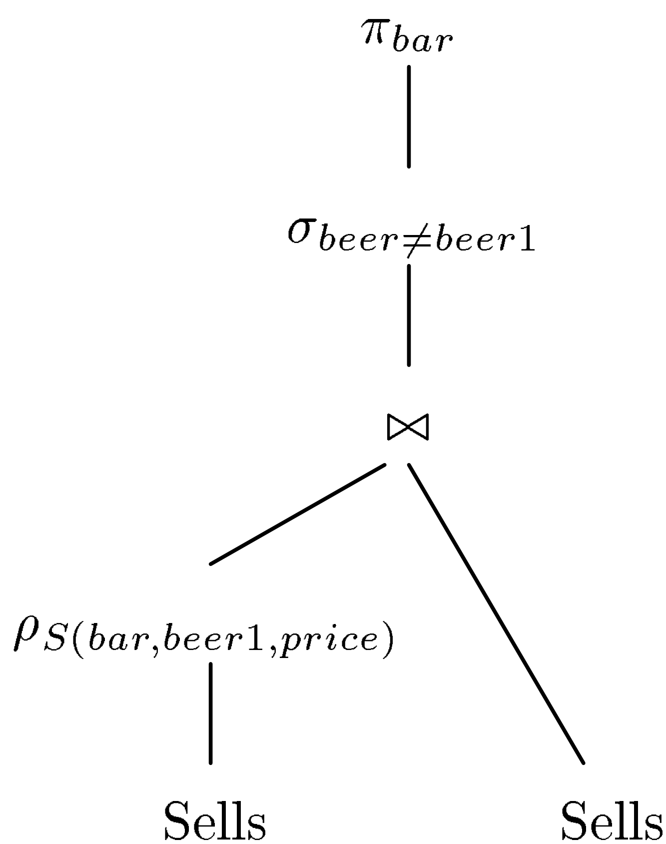
```
Sells(bar, beer, price)
Bars(name, addr)
```



## Example

Find the bars that sell two different beers at the same price.

`Sells(bar, beer, price)`



## Linear Notation for Expressions

- Invent new names for intermediate relations, and assign them values that are algebraic expressions.
- Renaming of attributes implicit in schema of new relation.

## Example

Find the bars that are either on Maple Street or sell Bud for less than \$3.

```
Sells(bar, beer, price)
Bars(name, addr)
```

R1(name) := $\pi_{name}(\sigma_{addr=Maple\ St.}(\text{Bars}))$

R2(name) :=

$\pi_{bar}(\sigma_{beer=Bud\ AND\ price<\$3}(\text{Sells}))$

R3(name) := R1 $\cup$ R2

## Why Decomposition "Works"?

What does it mean to "work"? Why can't we just tear sets of attributes apart as we like?

- Answer: the decomposed relations need to represent the same information as the original.
    - ✦ We must be able to reconstruct the original from the decomposed relations.

## Projection and Join Connect the Original and Decomposed Relations

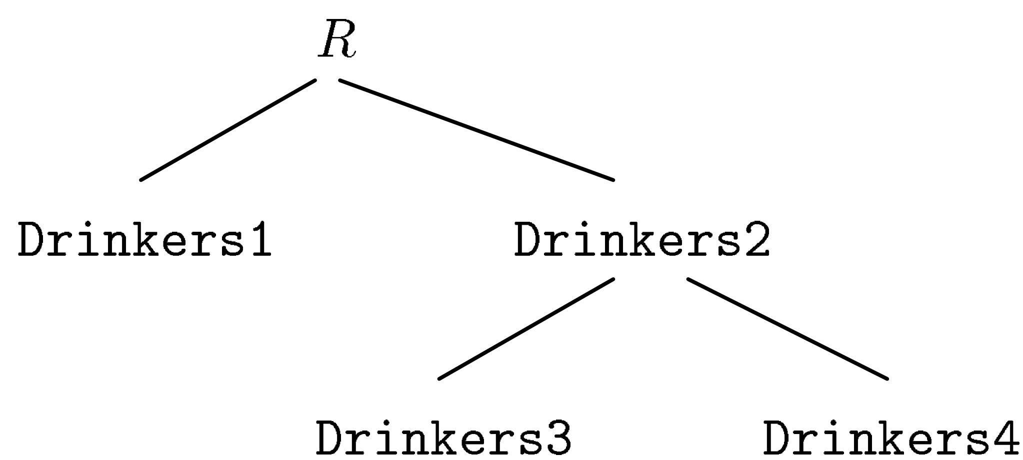
- Suppose $R$ is decomposed into $S$ and $T$. We project $R$ onto $S$ and onto $T$.

**Example**

$R =$

| name | addr | beersLiked | manf | favoriteBeer |
|---|---|---|---|---|
| Janeway | Voyager | Bud | A.B. | WickedAle |
| Janeway | Voyager | WickedAle | Pete's | WickedAle |
| Spock | Enterprise | Bud | A.B. | Bud |

- Recall we decomposed this relation as:



- Project onto `Drinkers1(name, addr, favoriteBeer)`:

| name | addr | favoriteBeer |
|---|---|---|
| Janeway | Voyager | WickedAle |
| Spock | Enterprise | Bud |

- Project onto `Drinkers3(beersLiked, manf)`:

| beersLiked | manf |
|---|---|
| Bud | A.B. |
| WickedAle | Pete's |

- Project onto `Drinkers4(name, beersLiked)`:

| name | beersLiked |
|---|---|
| Janeway | Bud |
| Janeway | WickedAle |
| Spock | Bud |

## Reconstruction of Original

Can we figure out the original relation from the decomposed relations?

- Sometimes, if we natural join the relations.

## Example

`Drinkers3` $\bowtie$ `Drinkers4` =

| name | beersLiked | manf |
|---|---|---|
| Janeway | Bud | A.B. |
| Janeway | WickedAle | Pete's |
| Spock | Bud | A.B. |

- Join of above with `Drinkers1` = original $R$.

**Theorem**

Suppose we decompose a relation with schema $XYZ$ into $XY$ and $XZ$ and project the relation for $XYZ$ onto $XY$ and $XZ$. Then $XY \bowtie XZ$ is *guaranteed* to reconstruct $XYZ$ if and only if $X \twoheadrightarrow Y$ (or equivalently, $X \twoheadrightarrow Z$.

- Usually, the MVD is really a FD, $X \rightarrow Y$ or $X \rightarrow Z$.
- BCNF: When we decompose $XYZ$ into $XY$ and $XZ$, it is because there is a FD $X \rightarrow Y$ or $X \rightarrow Z$ that violates BCNF.
    - ✦ Thus, we can always reconstruct $XYZ$ from its projections onto $XY$ and $XZ$.
- 4NF: when we decompose $XYZ$ into $XY$ and $XZ$, it is because there is an MVD $X \twoheadrightarrow Y$ or $X \twoheadrightarrow Z$ that violates 4NF.
    - ✦ Again, we can reconstruct $XYZ$ from its projections onto $XY$ and $XZ$.