Slides to be used with Ullman/Widom *A First Course in Database Systems*, Prentice-Hall, 1997

Oracle 8 Nested Tables

Another structuring tool provided in Oracle 8 is the ability to have a relation with an attribute whose value is not just an object, but a (multi)set of objects, i.e., a relation.

- Keyword `THE` allows us to treat a nested relation as a regular relation, e.g., in `FROM` clauses.
- Keywords `CAST(MULTISET(...))` let us turn the result of a query into a nested relation.

Defining Table Types

If we have an object type, we can create a new type that is a bag of that type by `AS TABLE OF`.

Example

Suppose we have a more complicated beer type:

```
CREATE TYPE BeerType AS OBJECT (
    name CHAR(20),
    kind CHAR(5),
    color CHAR(5)
);
/
```

We may create a type that is a (nested) table of objects of this type by:

```
CREATE TYPE BeerTableType AS
    TABLE OF BeerType;
/
```

Now, we can define a relation of manufacturers that will nest their beers inside.

- In a sense, we normalize an unnormalized relation, since other data about the manufacturer appears only once no matter how many beers they produce.

```
CREATE TABLE Manfs (
    name CHAR(30),
    addr CHAR(50),
    beers BeerTableType
);
```

Querying With Nested Tables

An attribute that is a nested table can be printed like any other attribute.

- The value has two type constructors, one for the table, one for the type of its tuples.

Example

List the beers made by Anheuser-Busch.

```
SELECT beers
FROM Manfs
WHERE name = 'Anheuser Busch';
```

- A single value will be printed, looking something like:

```
BeerTableType(
    BeerType('Bud', 'lager', 'yello'),
    BeerType('Lite', 'malt', 'pale'),...
)
```

Operating on Nested Tables

Use THE to get the nested table itself, then treat it like any other relation.

Example

Find the ales made by Anheuser-Busch.

```
SELECT bb.name
FROM THE(
    SELECT beers
    FROM Manfs
    WHERE name = 'Anheuser Busch'
) bb
WHERE bb.kind = 'ale';
```

Casting to Create Nested Tables

Create a value for a nested table by using a select-from-where query and "casting" it to the table type.

Example

- Suppose we have a relation `Beers(beer, manf)`, where `beer` is a `BeerType` object and `manf` its manufacturer.
- We want to insert into `Manfs` a tuple for Pete's Brewing Co., with all the beers brewed by Pete's (according to `Beers`) in one nested table.

```
INSERT INTO Manfs VALUES(
    'Pete''s', 'Palo Alto',
    CAST(
        MULTISET(
            SELECT bb.beer
            FROM Beers bb
            WHERE bb.manf = 'Pete''s'
        ) AS BeerType
    )
);
```

Return to Normalization

Recall that we learned how to "normalize" relations (= put them in BCNF) by *decomposing* their schemas into two or more sets of attributes.

- We acted as if the decomposition was OK; the new relations were good substitutes for the original relation.
- It turns out to be OK when the decomposition is governed by a BCNF violation, but may not be OK otherwise.

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

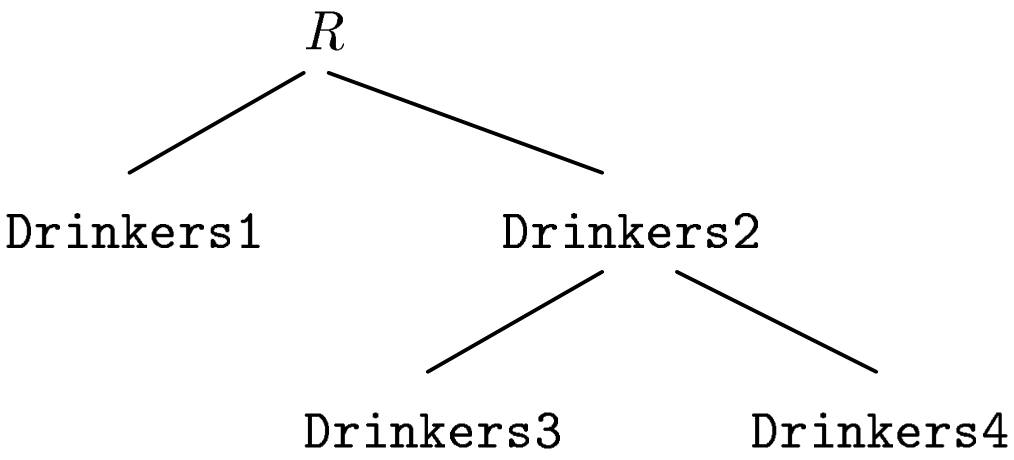
- Suppose R is decomposed into S and T. We project R onto S and onto T.

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

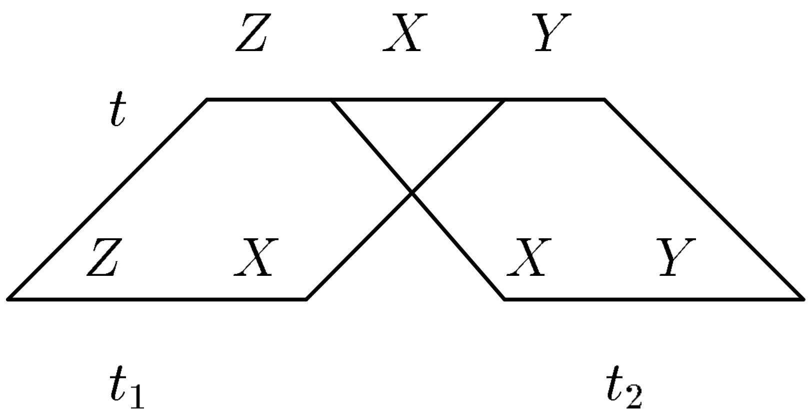
Suppose we decompose a relation with schema XYZ into XY and XZ and project the relation for XYZ onto XY and XZ. Then $XY \bowtie XZ$ is *guaranteed* to reconstruct XYZ if and only if either $X \rightarrow Y$ or $X \rightarrow Z$ holds.

- Notice that whenever we decompose because of a BCNF violation, one of these FD's must hold.

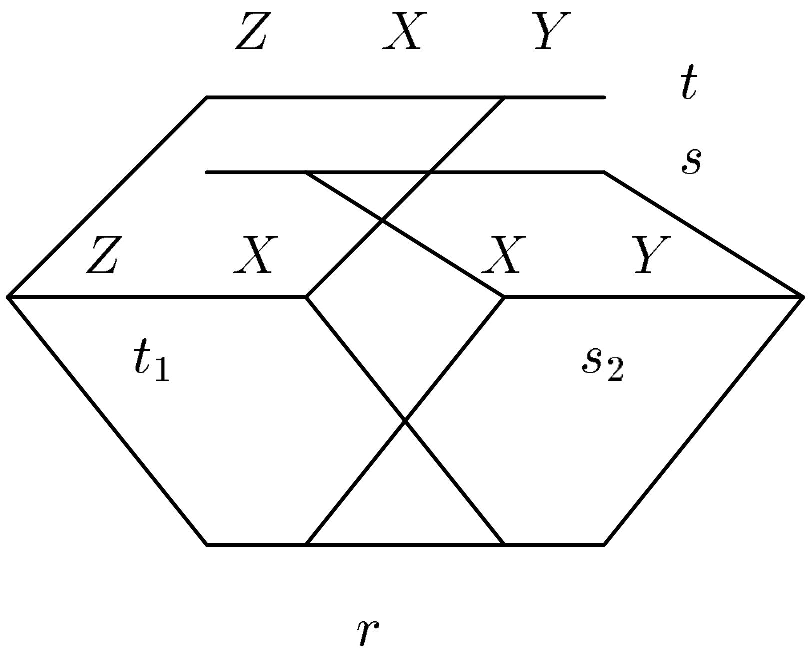
Proof (if)

1. Anything you project comes back in the join.

   - Doesn't depend on FD's.



2. Anything that comes back in the join was in the original XYZ.



- Notice that t_1 and s_2 agree on X; therefore so do t and s.
- If $X \rightarrow Y$, then $r = t$.
- If $X \rightarrow Z$, then $r = s$.
- Either way, r is in original XYZ.

Proof (only-if)

If neither $X \rightarrow Y$ nor $X \rightarrow Z$ holds, then we can find an example XYZ relation where the project-join returns too much.

Z	X	Y
$z1$	x	$y1$
$z2$	x	$y2$

Z	X
$z1$	x
$z2$	x

X	Y
x	$y1$
x	$y2$

Z	X	Y
$z1$	x	$y1$
$z1$	x	$y2$
$z2$	x	$y1$
$z2$	x	$y2$

Application to BCNF Decomposition

- When we decompose R into S and T, it is because there is a FD of the form $(S \cap T) \rightarrow (T - S)$.
- Thus, we can always reconstruct R from S and T.