MCDB: The Monte Carlo Database System

Chris Jermaine

Rice U.

Joint work with:

Peter Haas, Ravi Jampani, Luis Perez, Subi Arumugam, Foula Vagena

…many others…

Motivating Scenario...

- We have archived billions of sales records and want to know:

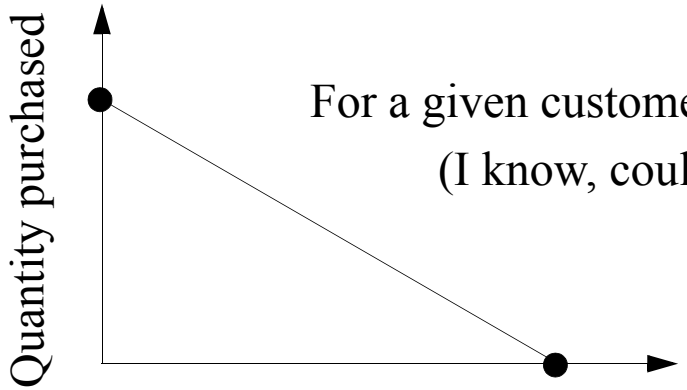
  "What would my profits have been in '08 if I'd cut all of my margins by 10%?"

- How might we use archived data to guess this answer?
  - — Need to "guess" each customer's demand at new price
  - — Use to build a hypothetical, revised version of Lineitem
  - — Finally, query this hypothetical table

- Here's one possibility
  - — ...utilizing the Bayesian approach

Step 1: A Stochastic Demand Model

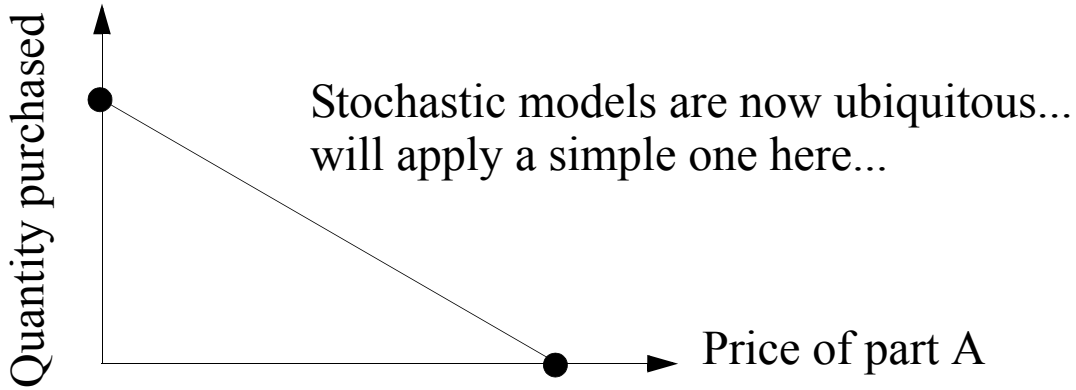
- First, define a customer demand model...



For a given customer, demand is a linear function
(I know, could do better than linear...)

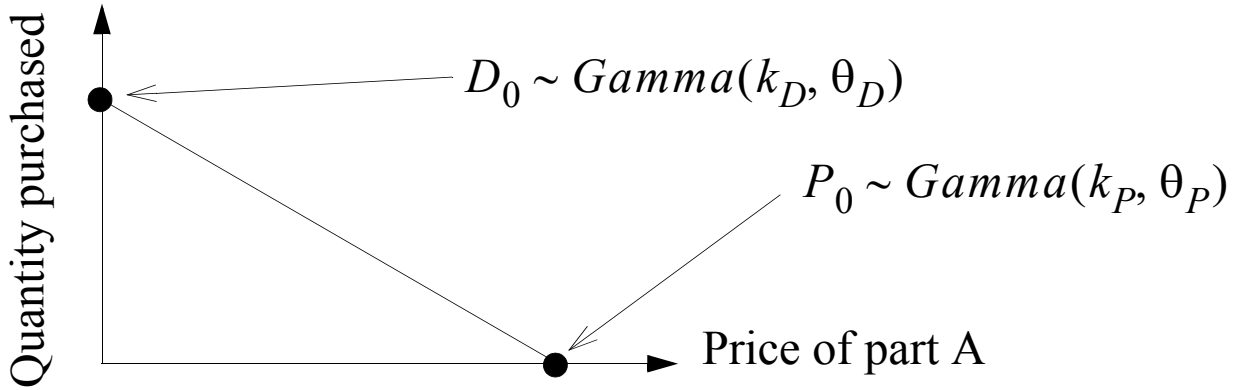
Step 1: A Stochastic Demand Model

- First, define a customer demand model...



Step 1: A Stochastic Demand Model

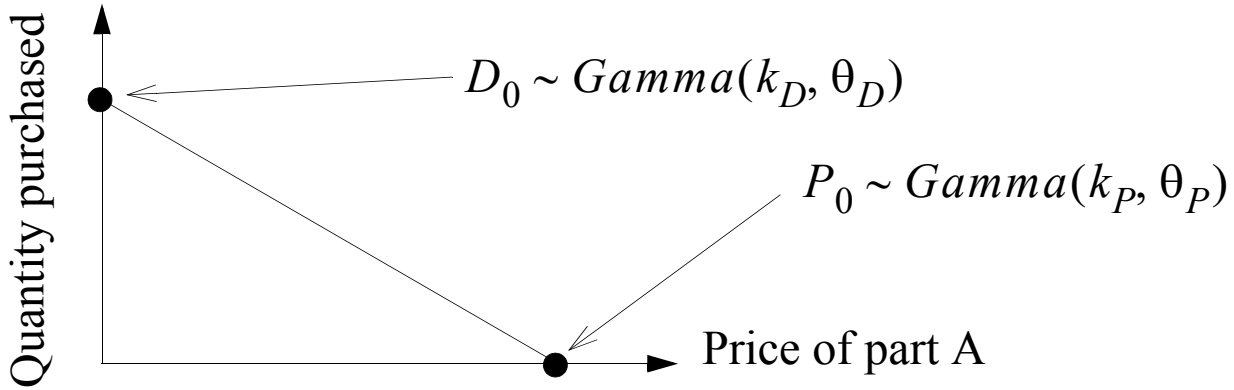
- First, define a customer demand model...



Demand curve is generated via samples from twin Gamma distributions

Step 1: A Stochastic Demand Model

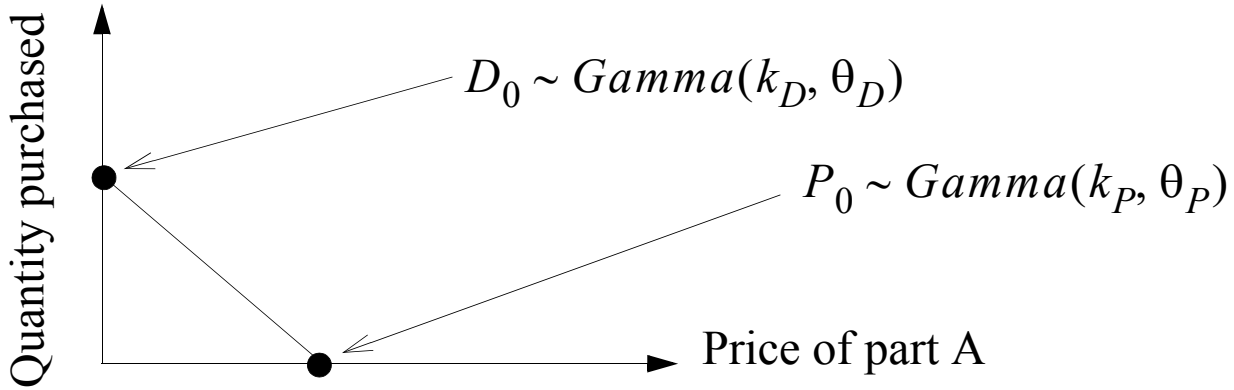
- First, define a customer demand model...



Distribution over D_0, P_0 defines a whole family of possible demand curves...

Step 1: A Stochastic Demand Model

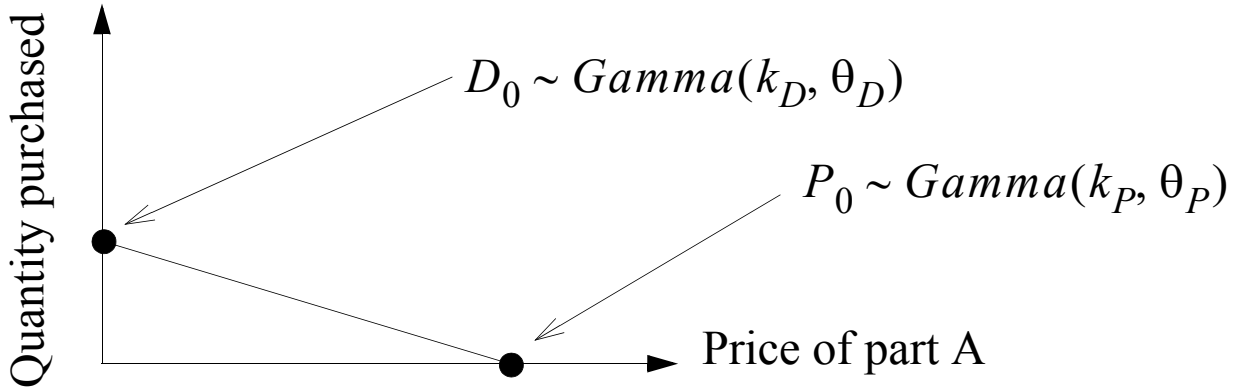
- First, define a customer demand model...



Here's a new, possible demand curve...

Step 1: A Stochastic Demand Model

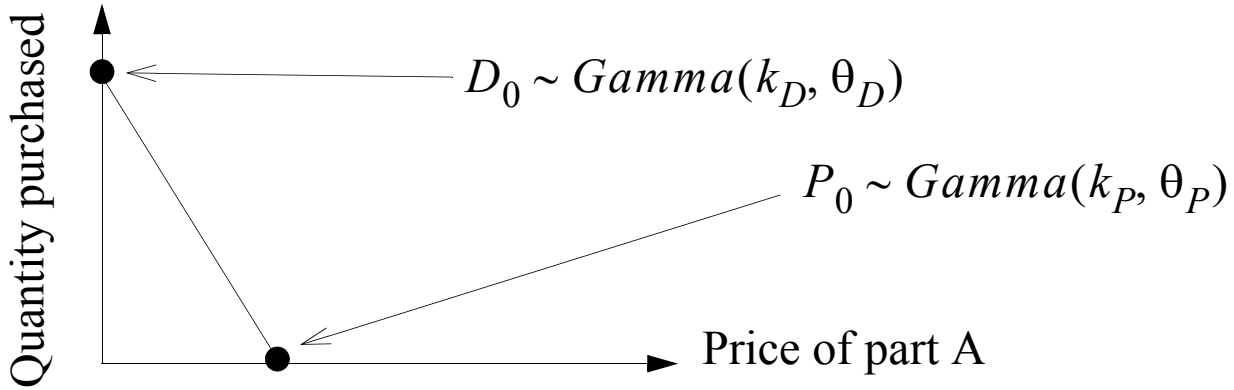
- First, define a customer demand model...



And another...

Step 1: A Stochastic Demand Model

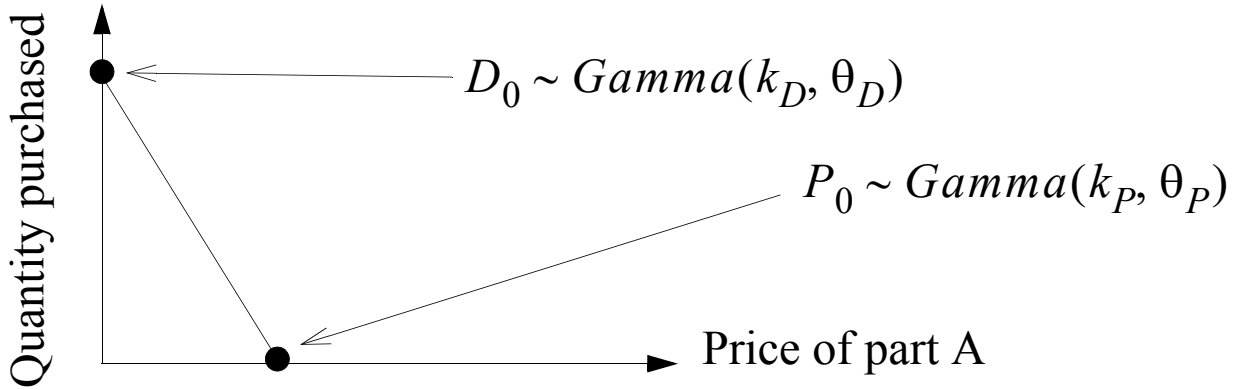
- First, define a customer demand model...



And so on...

Step 1: A Stochastic Demand Model

- First, define a customer demand model...



- The PDF $F(f \mid k_P, k_D, \theta_P, \theta_D)$ is what is known as a "prior distribution" in Bayesian stats

Step 2: "Learn" the Model

- To apply this model, we need to "learn" the prior
  - — That is, choose k_P, k_D, θ_P, θ_D to be realistic for sales in 2008
- Reasonable tactic: use the warehouse data to perform an MLE
- Several ways this might happen:

(1) Can we derive a closed form solution? If, for example:

$$k_P = \frac{a^2 + b^2 \times (\ldots)}{c \times (\ldots)^2}$$

Then I'd write a bunch of SQL aggregate queries to extract a, b, c from the data

(2) On the other hand, if MLE requires numerical methods over base data, I'd sample or extract representative data from the warehouse, load into Matlab, and go from there...

Step 3: Apply the Model - the Theory

- Now we have a prior model (PDF) for demand function *f*:

$$F(f \mid k_P, k_D, \theta_P, \theta_D)$$

- Problem: the actual demand curve for each customer is unseen
- But can use observed demand to obtain a *posterior* demand model
- Ex: for *i*th sale, a customer bought *d* units at price *p*
- Then posterior demand model is given by:

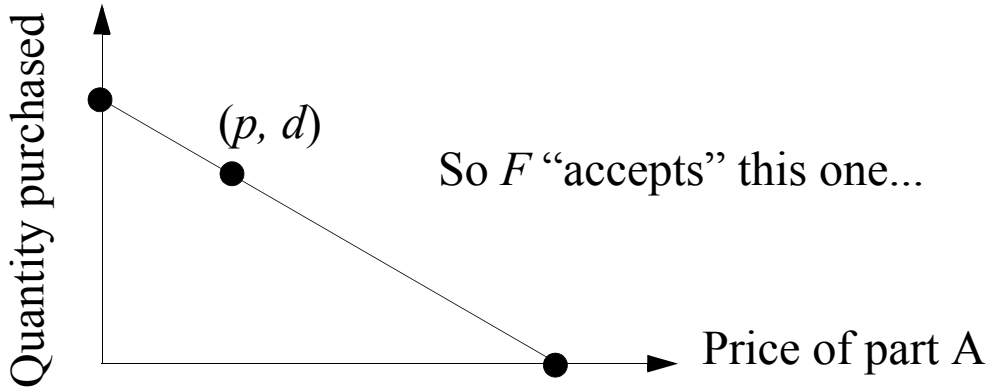
$$F(f_i \mid k_P, k_D, \theta_P, \theta_D, f_i(p) = d)$$

Step 3: Apply the Model - the Theory

- Intuitively, $F(f_i \mid f_i(p) = d)$ gives non-zero "weight" to all demand functions thru the point (p, d)

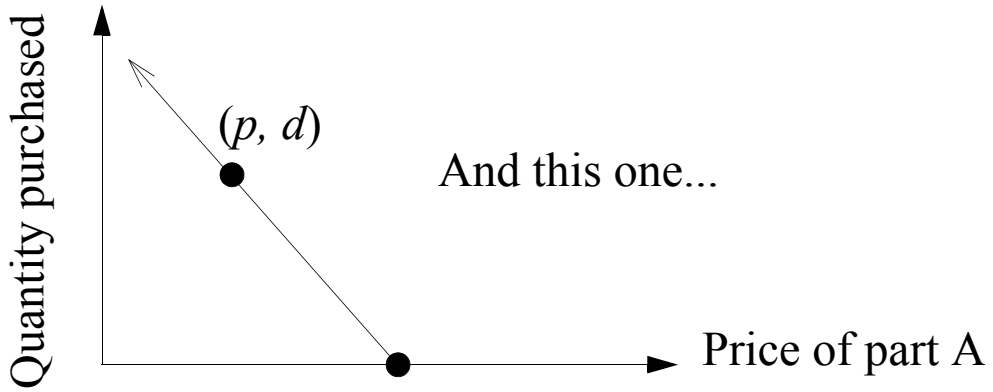
Step 3: Apply the Model - the Theory

- Intuitively, $F(f_i \mid f_i(p) = d)$ gives non-zero "weight" to all demand functions thru the point (p, d)



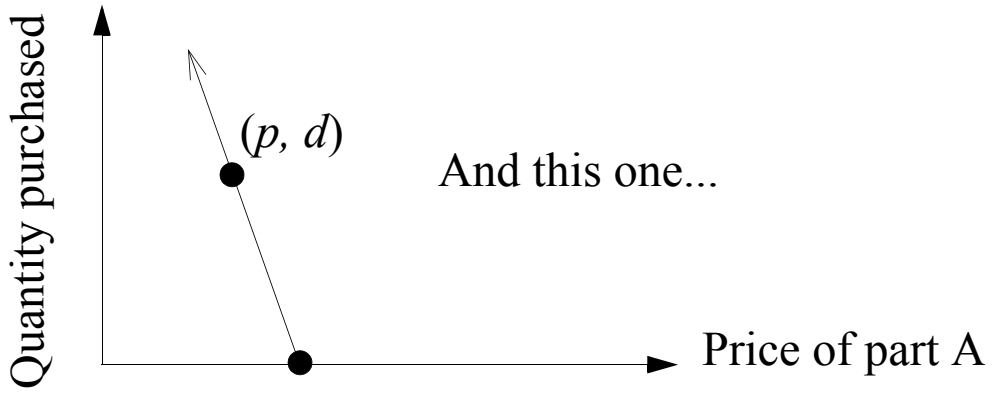
Step 3: Apply the Model - the Theory

- Intuitively, $F(f_i \mid f_i(p) = d)$ gives non-zero "weight" to all demand functions thru the point (p, d)



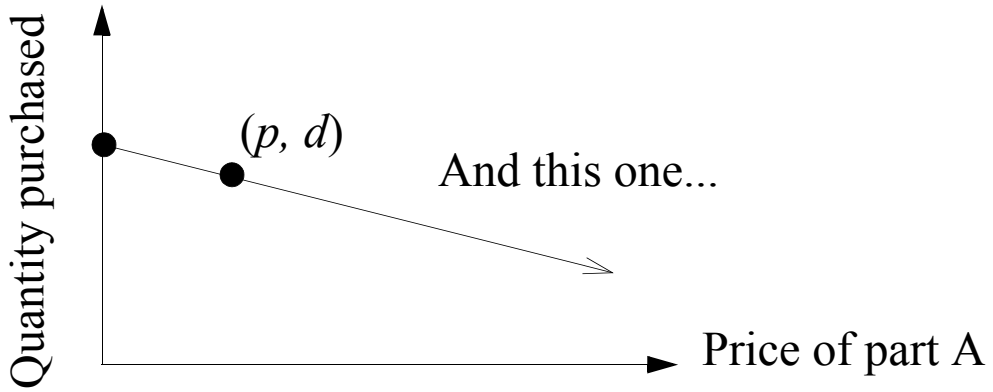
Step 3: Apply the Model - the Theory

- Intuitively, $F(f_i \mid f_i(p) = d)$ gives non-zero "weight" to all demand functions thru the point (p, d)



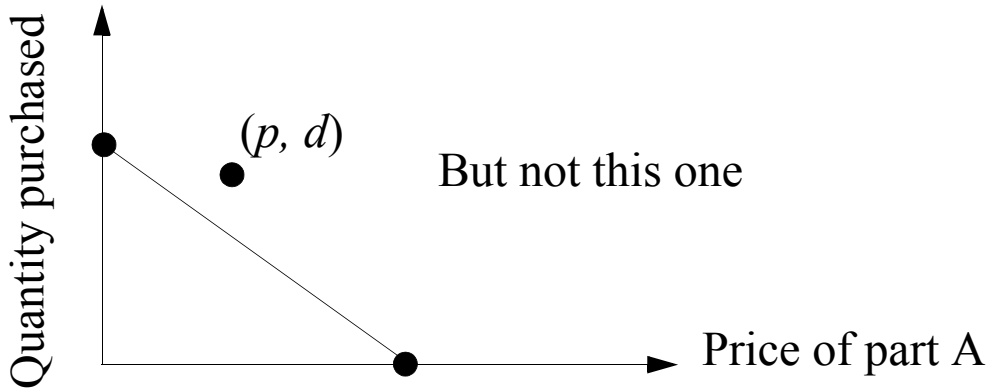
Step 3: Apply the Model - the Theory

- Intuitively, $F(f_i \mid f_i(p) = d)$ gives non-zero "weight" to all demand functions thru the point (p, d)



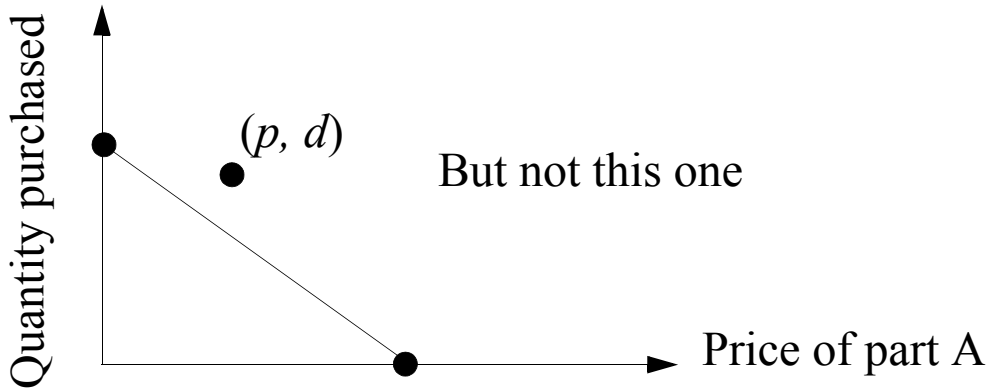
Step 3: Apply the Model - the Theory

- Intuitively, $F(f_i \mid f_i(p) = d)$ gives non-zero "weight" to all demand functions thru the point (p, d)



Step 3: Apply the Model - the Theory

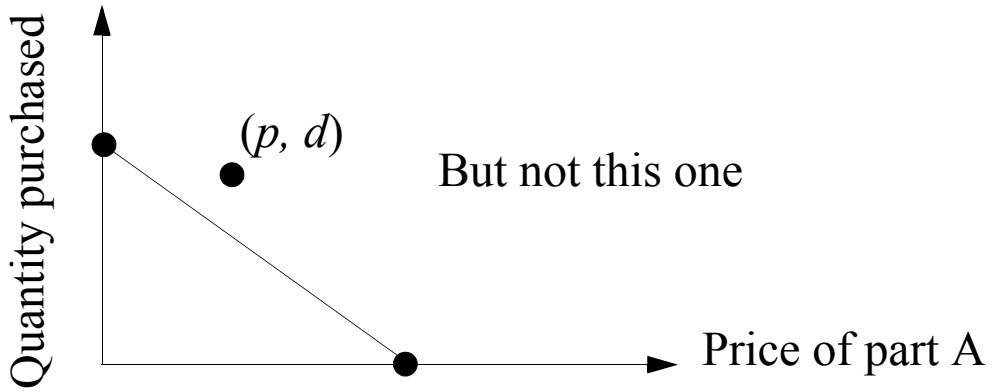
- Intuitively, $F(f_i \mid f_i(p) = d)$ gives non-zero "weight" to all demand functions thru the point (p, d)



- Those demand curves that are given non-zero weight are ordered exactly as the prior would order them

Step 3: Apply the Model - the Theory

- Intuitively, $F(f_i \mid f_i(p) = d)$ gives non-zero "weight" to all demand functions thru the point (p, d)



- Those demand curves that are given non-zero weight are ordered exactly as the prior would order them
- To "guess" the customer's demand at new price p':
  - — Sample a possible f_i from the PDF $F(f_i \mid k_P, k_D, \theta_P, \theta_D, f_i(p) = d)$
  - — Compute $f_i(p')$, and we have a new demand!

Step 3: Apply the Model - the Practice

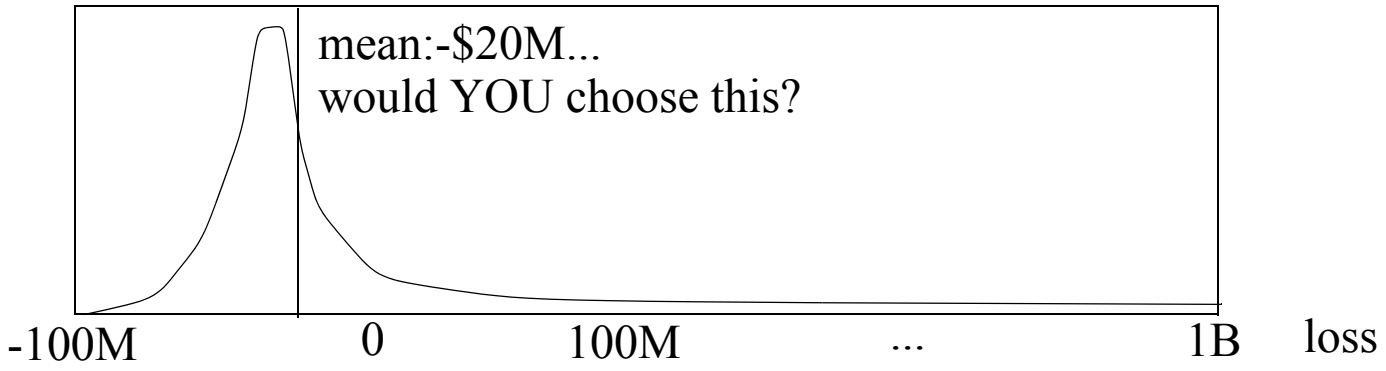
- To actually **compute** the overall profit under new prices:
  - — I'd need to sample an f_i from $F(f_i \mid f_i(p) = d)$ for every sale from 2008
  - — Evaluate $f_i(p')$ for all those sales
  - — Compute the new profits

Does Current DB Tech Support This?

- No! Why? Many difficulties...
- Most significant: no support for quntifiction of uncertainty/risk

Systematic Quant. of Uncertainty/Risk

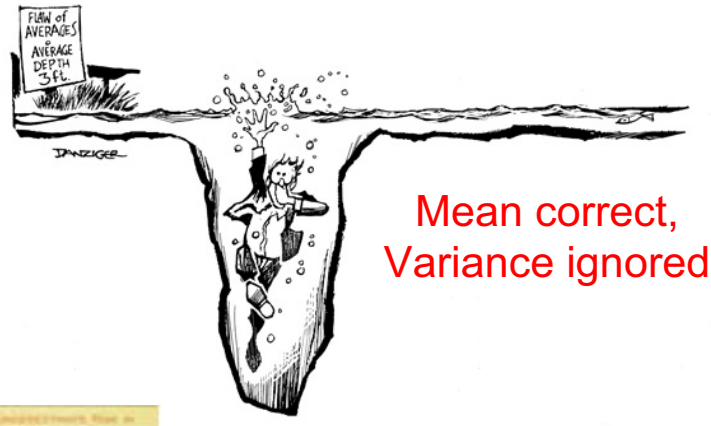
- Big advantage of stocastic models
  - — And especially of the Bayesian approach
  - — You don't just get one picture of the world
  - — You get a whole distribution of possibilities
  - — Which give you a complete picture of how bad/good things can get
- Increasing realization that ignorance of systematic risk is bad...



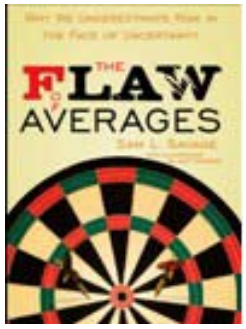
Systematic Quant. of Uncertainty/Risk

- Worthwhile polemic is Sam Savage (Stanford) *Flaw of Averages*
- Savage describes two "f/laws":

Flaw of averages (weak form):



Mean correct,
Variance ignored



Sam Savage's book

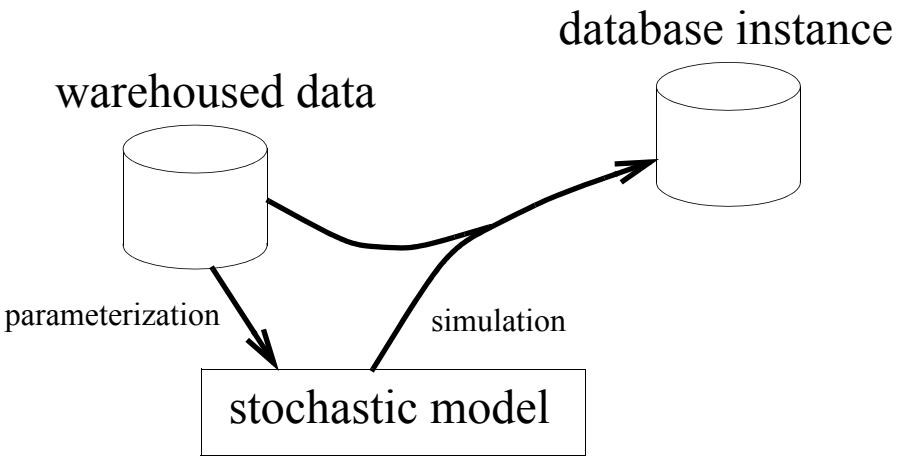
Flaw of averages (strong form):

Wrong value of mean:
$f(E[X]) \neq E[f(X)]$

Can We Fix This?

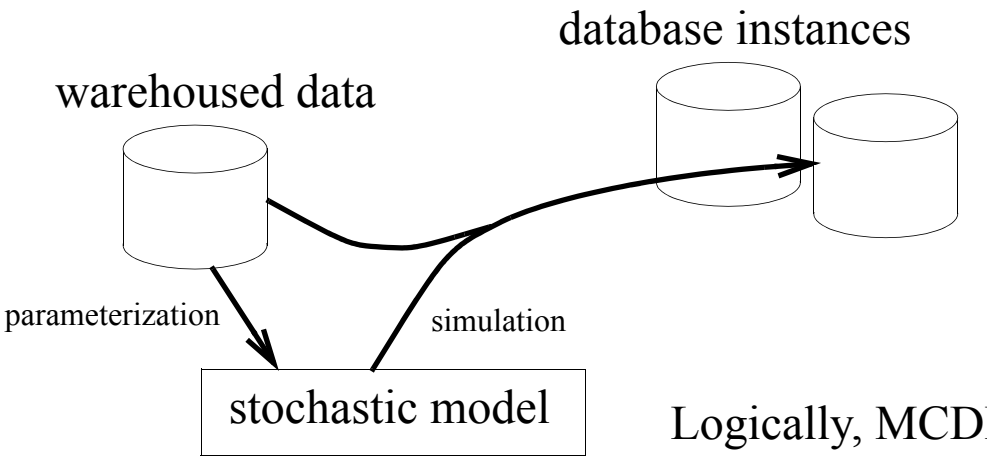
Our System: MCDB

- SQL-based parallel database/simulation engine
- User-specified models stochastically generate database instances
  - — Instance combines "real" data (e.g., existing Customer table)
  - — and "hypothetical" data (e.g., hypothetical LineitemNew)



Our System: MCDB

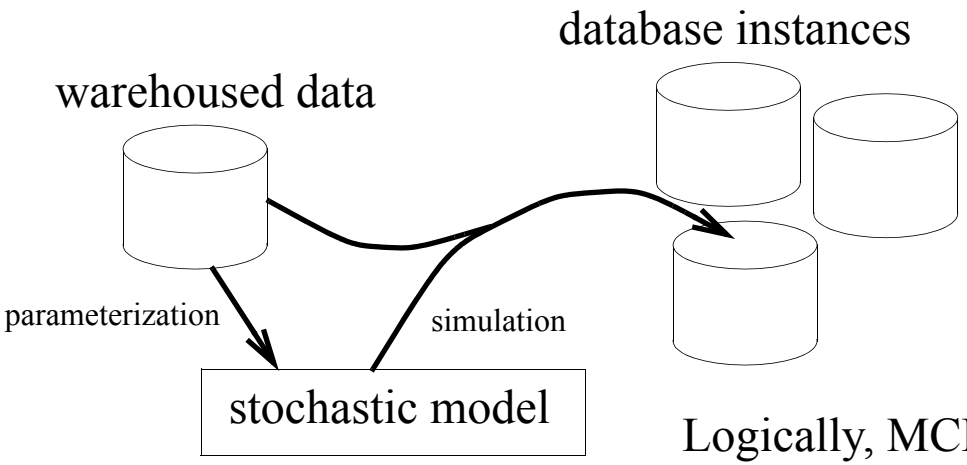
- SQL-based parallel database/simulation engine
- User-specified models stochastically generate database instances
  - — Instance combines "real" data (e.g., existing Customer table)
  - — and "hypothetical" data (e.g., hypothetical LineitemNew)



Logically, MCDB generates many database instances ("possible worlds")

Our System: MCDB

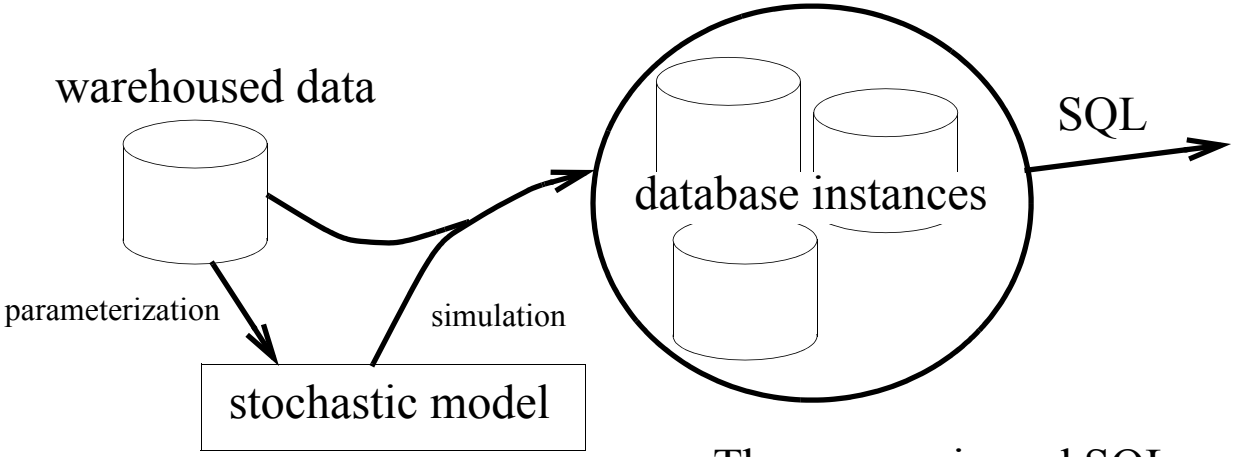
- SQL-based parallel database/simulation engine
- User-specified models stochastically generate database instances
  - — Instance combines "real" data (e.g., existing Customer table)
  - — and "hypothetical" data (e.g., hypothetical LineitemNew)



Logically, MCDB generates many database instances ("possible worlds")

Our System: MCDB

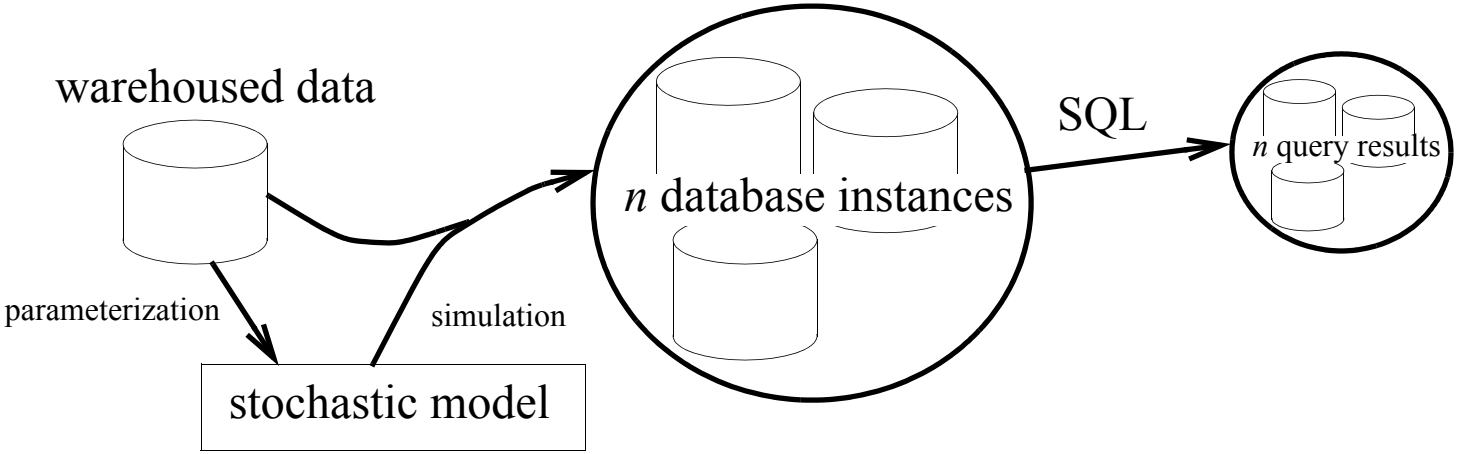
- SQL-based parallel database/simulation engine
- User-specified models stochastically generate database instances
  - — Instance combines "real" data (e.g., existing Customer table)
  - — and "hypothetical" data (e.g., hypothetical LineitemNew)



Then a user-issued SQL query
is simultaneously evaluated over all instances

Our System: MCDB

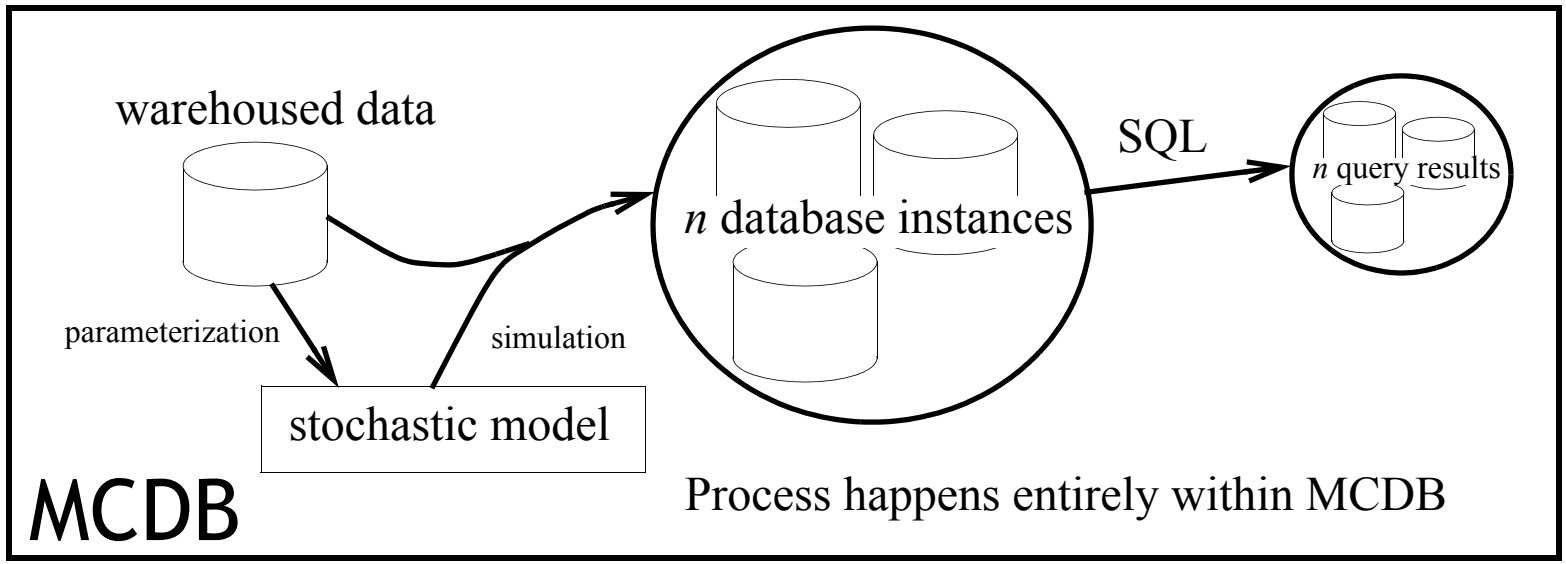
- SQL-based parallel database/simulation engine
- User-specified models stochastically generate database instances
  - — Instance combines "real" data (e.g., existing Customer table)
  - — and "hypothetical" data (e.g., hypothetical LineitemNew)



Gives an empirical distribution of result sets

Our System: MCDB

- SQL-based parallel database/simulation engine
- User-specified models stochastically generate database instances
  - — Instance combines "real" data (e.g., existing Customer table)
  - — and "hypothetical" data (e.g., hypothetical LineitemNew)



How Do We Pull This Off?

- At the core: stochastic model is encapsulated in user-defined "VG Function"
- Invoked by MCDB to produce simulated database data
- Implemented as (possibly) user-defined C++ class dynamically linked into MCDB at run time, will move to R eventually

Invoking VG Functions

- MCDB has a special, stochastic `CREATE TABLE` statement
- Stochastic tables can be queried like any other
- Example; assuming:
  - `SBP_PARAM(MEAN, STD)`
  - `PATIENTS(PID, GENDER)`
- To create a stochastic table, we might have:

```
CREATE TABLE SBP_DATA(PID, GENDER, SBP) AS
  FOR EACH p in PATIENTS
    WITH SBP AS Normal (
      (SELECT s.MEAN, s.STD
       FROM SPB_PARAM s))
    SELECT p.PID, p.GENDER, b.VALUE
    FROM SBP b
```

A More Complicated Example

- Want to generate a stochastic `CUST` table
- Income is stochastic
  - Generated using Gamma distribution
  - Shape in `MONEY_SHAPE (CID, SHAPE)`
  - Scale depends on customer's region: `MONEY_SCALE (REGION, SCALE)`
  - Shift is always the same: `MONEY_SHIFT (SHIFT)`
- Customer's exact city is also stochastic
  - `CITIES (NAME, REGION, PROB)`

A More Complicated Example

- We have:

```
CREATE TABLE CUST(CID, GENDER, MONEY, LIVES_IN) AS
  FOR EACH d in CUST_ATTRS
    WITH MONEY AS Gamma(
      (SELECT n.SHAPE
       FROM MONEY SHAPE n
       WHERE n.CID = d.CID),
      (SELECT sc.SCALE
       FROM MONEY_SCALE sc
       WHERE sc.REGION = d.REGION),
      (SELECT SHIFT
       FROM MONEY_SHIFT))
    WITH LIVES_IN AS DiscreteChoice (
      (SELECT c.NAME, c.PROB
       FROM CITIES c
       WHERE c.REGION = d.REGION))
    SELECT d.CID, d.GENDER, m.VALUE, l.VALUE
    FROM MONEY m, LIVES_IN l
```

Correlations Are Easily Handled

- Within-tuple correlations easy:
  - — Just define a multi-dimensional VG function

```
CREATE TABLE CUST(CID, GENDER, MONEY, LIVES_IN) AS
  FOR EACH d in CUST ATTRS
    WITH MLI AS MyJointDistribution (...)
    SELECT d.CID, d.GENDER, MLI.VALUE1, MLI.VALUE2
    FROM MLI
```

Cross-Tuple Correlations

- Say we want to generate correlated sensor readings
- Have the following DB tables:
  - Location, group ID in `S_PARAMS (ID, LAT, LONG, GID)`
  - Mean reading in `MEANS (ID, MEAN)`
  - Variances, correlations in `COVARS (ID1, ID2, COVAR)`

Cross-Tuple Correlations

- Can use:

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S PARAMS)
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
      FROM MEANS m, SENSOR PARAMS ss
      WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
      FROM COVARS c, SENSOR PARAMS ss
      WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

VG Function Interface

- To write a VG function, implement C++ class with four methods
- Called in sequence, one sequence per `for each` tuple

VG Function Interface

- To write a VG function, implement C++ class with four methods
- Called in sequence, one sequence per `for each` tuple
- `Initialize`: set up internal data structures
  - — Accepts random number seed as input
  - — That way, output is repeatable!

VG Function Interface

- To write a VG function, implement C++ class with four methods
- Called in sequence, one sequence per `for each` tuple
- `Initialize`: set up internal data structures
  - — Accepts random number seed as input
  - — That way, output is repeatable!
- `TakeParams`: parameterize the VG function
  - — Called once for each parameter tuple

VG Function Interface

- To write a VG function, implement C++ class with four methods
- Called in sequence, one sequence per `for each` tuple
- `Initialize`: set up internal data structures
  - — Accepts random number seed as input
  - — That way, output is repeatable!
- `TakeParams`: parameterize the VG function
  - — Called once for each parameter tuple
- `OutputVals`: produce one set of stochastic attribute values
  - — Repeatedly called until a `null` is returned

VG Function Interface

- To write a VG function, implement C++ class with four methods
- Called in sequence, one sequence per `for each` tuple
- `Initialize`: set up internal data structures
  - — Accepts random number seed as input
  - — That way, output is repeatable!
- `TakeParams`: parameterize the VG function
  - — Called once for each parameter tuple
- `OutputVals`: produce one set of stochastic attribute values
  - — Repeatedly called until a `null` is returned
- `Finalize`: delete internal data structures

Example

gid
1
2
3
4

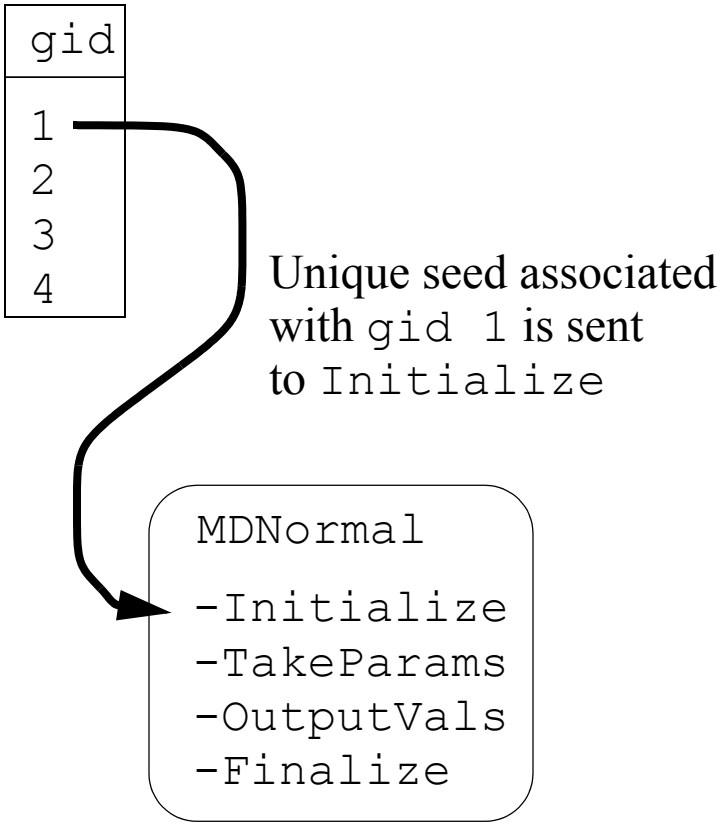
MCDB cycles thru gids

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
 FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
 WITH TEMP AS MDNormal(
    (SELECT m.ID, m.MEAN
      FROM MEANS m, SENSOR_PARAMS ss
      WHERE m.ID = ss.ID AND ss.GID = g.GID),
    (SELECT c.ID1, c.ID2, c.COVAR
      FROM COVARS c, SENSOR_PARAMS ss
      WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
 SELECT s.ID, s.LAT, s.LONG, t.VALUE
 FROM SENSOR_PARAMS s, TEMP t
 WHERE s.ID = t.ID
```

MDNormal

-Initialize
-TakeParams
-OutputVals
-Finalize

Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

Example

gid
2
3
4

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
      FROM MEANS m, SENSOR_PARAMS ss
      WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
      FROM COVARS c, SENSOR_PARAMS ss
      WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

MDNormal

-Initialize
-TakeParams
-OutputVals
-Finalize

Subqueries are evaluated for `gid 1`

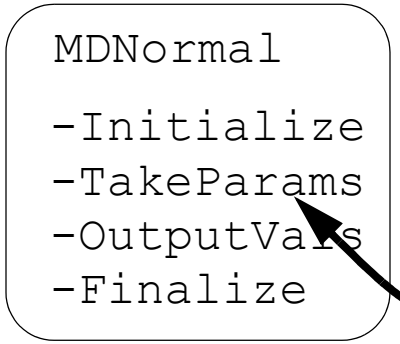
ID	MEAN
1	72.5
2	77.9

ID1	ID2	COVAR
1	1	22.5
2	1	-5.8
1	2	-5.8
2	2	35.2

Example

gid
2
3
4

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```



All of the results are sent to `TakeParams`

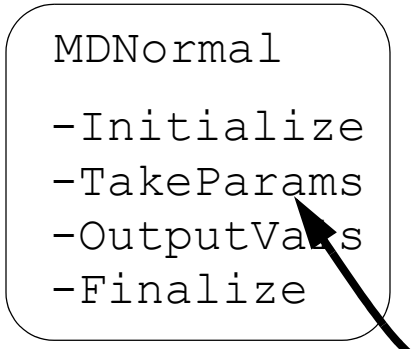
ID	MEAN
1	72.5
2	77.9

ID1	ID2	COVAR
1	1	22.5
2	1	-5.8
1	2	-5.8
2	2	35.2

Example

gid
2
3
4

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```



All of the results are sent to `TakeParams`

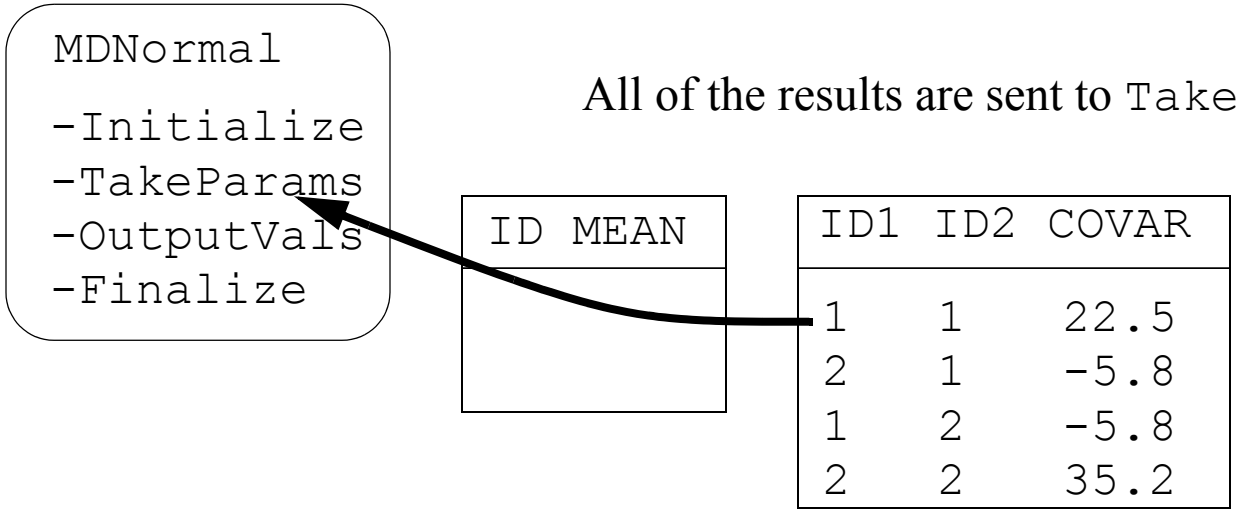
ID	MEAN
2	77.9

ID1	ID2	COVAR
1	1	22.5
2	1	-5.8
1	2	-5.8
2	2	35.2

Example

gid
2
3
4

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

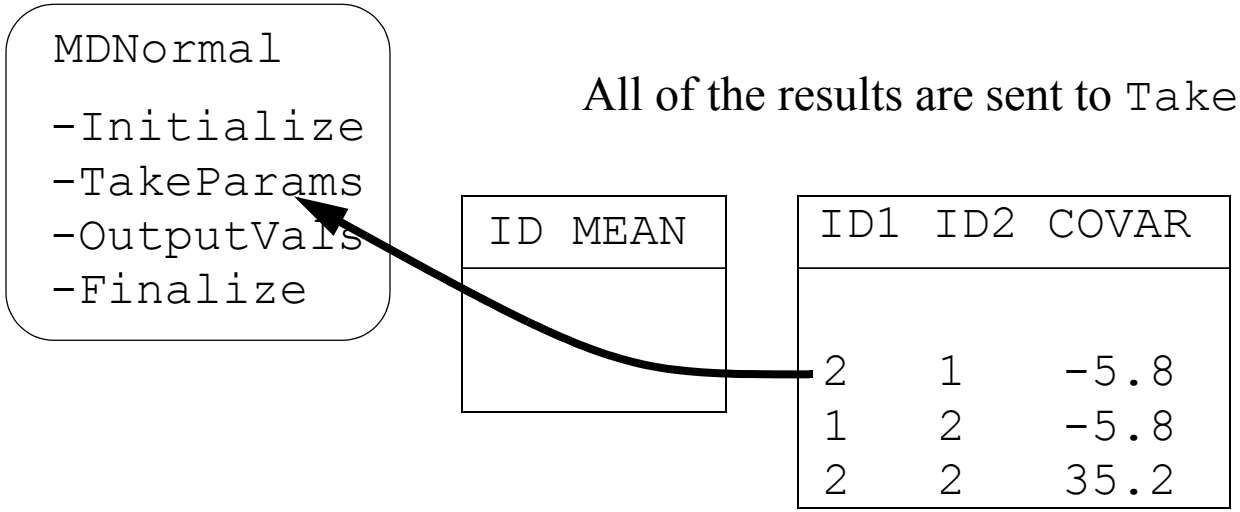


All of the results are sent to `TakeParams`

Example

gid
2
3
4

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

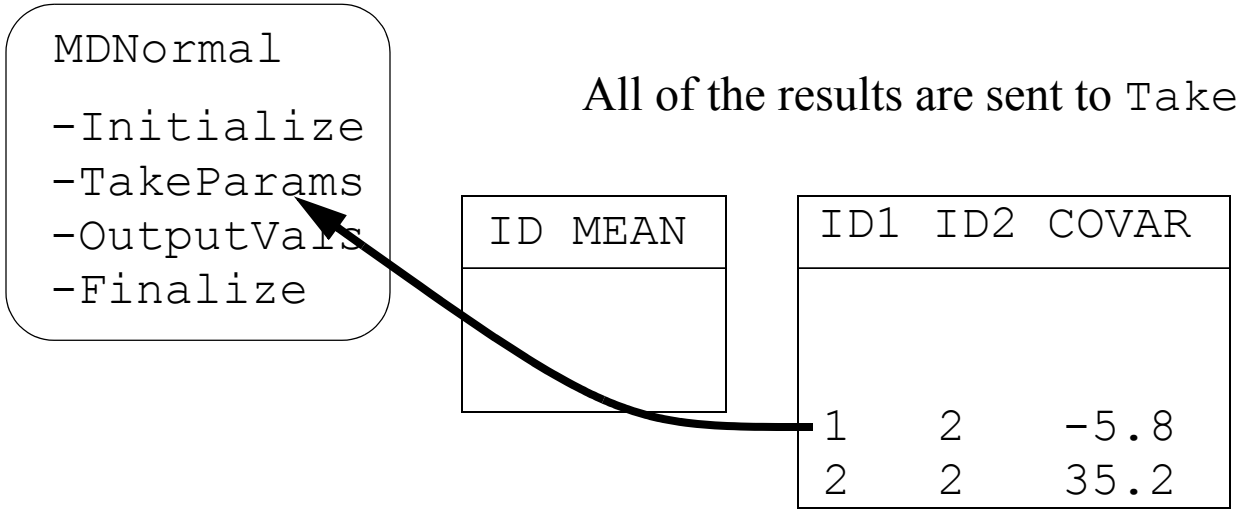


All of the results are sent to `TakeParams`

Example

gid
2
3
4

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

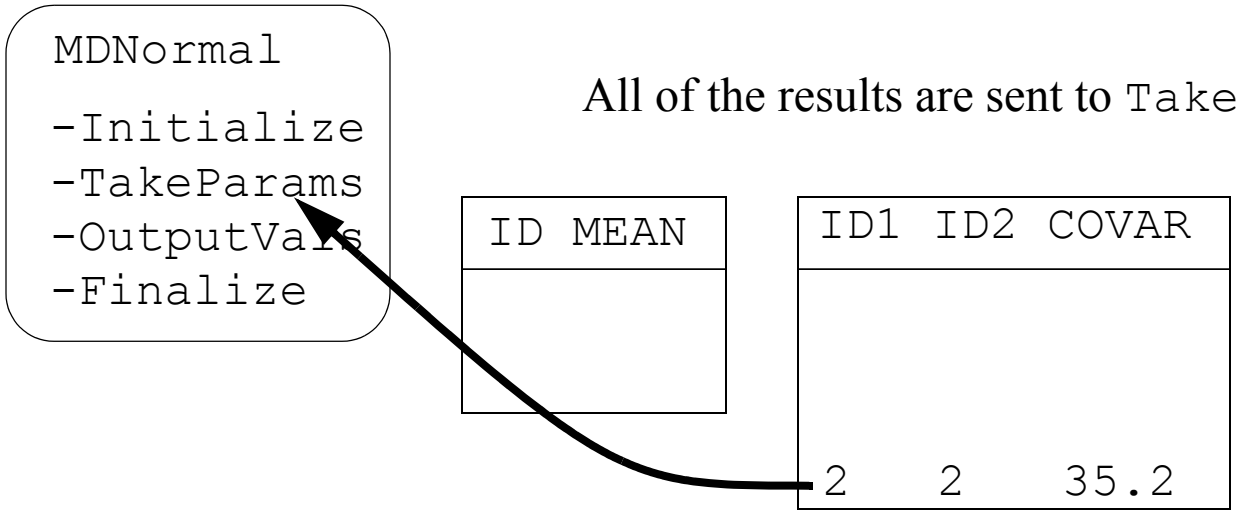


All of the results are sent to `TakeParams`

Example

gid
2
3
4

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```



All of the results are sent to `TakeParams`

Example

gid
2
3
4

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

MDNormal

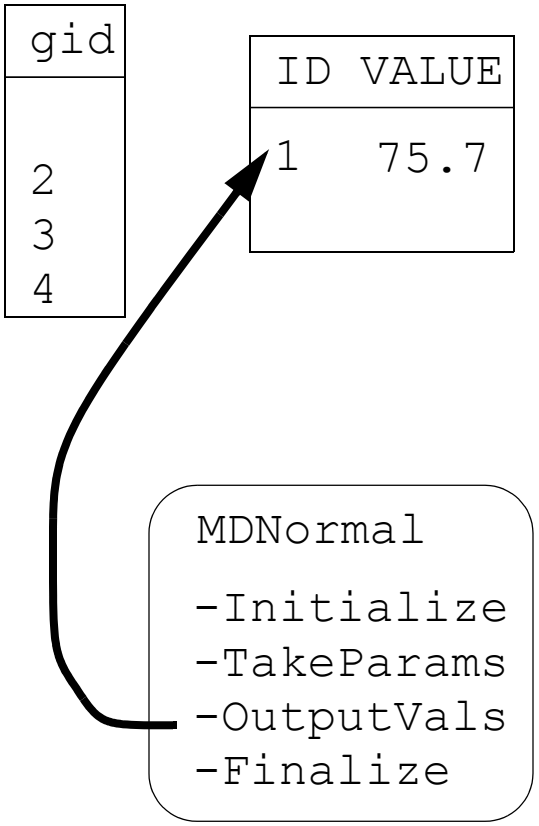
-Initialize
-TakeParams
-OutputVals
-Finalize

All of the results are sent to TakeParams

ID MEAN

ID1 ID2 COVAR

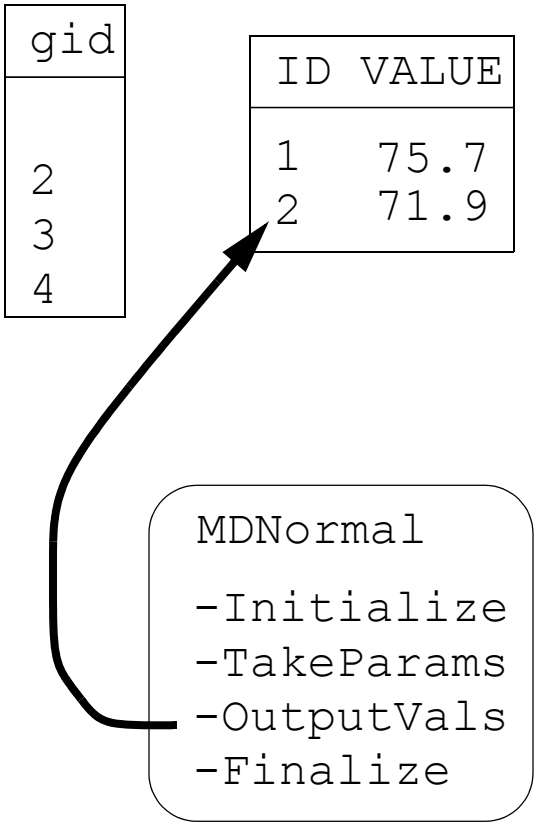
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

`OutputVals` is then called until a null

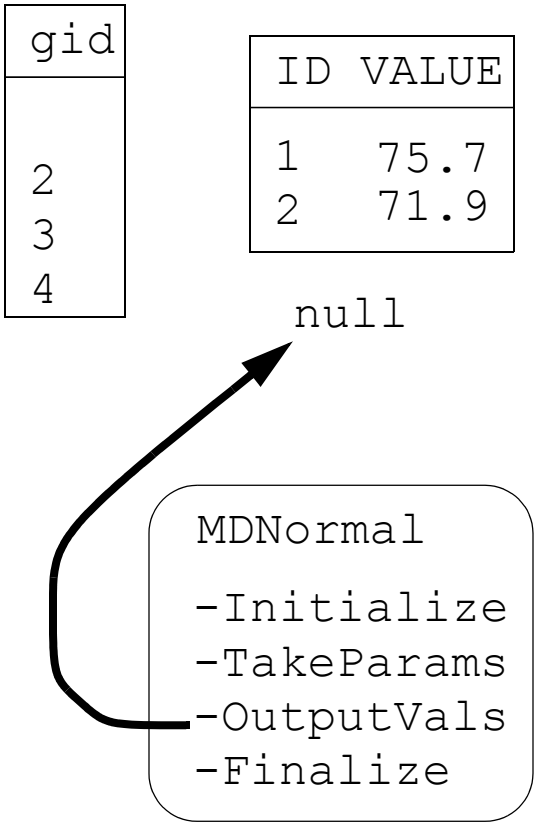
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

`OutputVals` is then called until a null

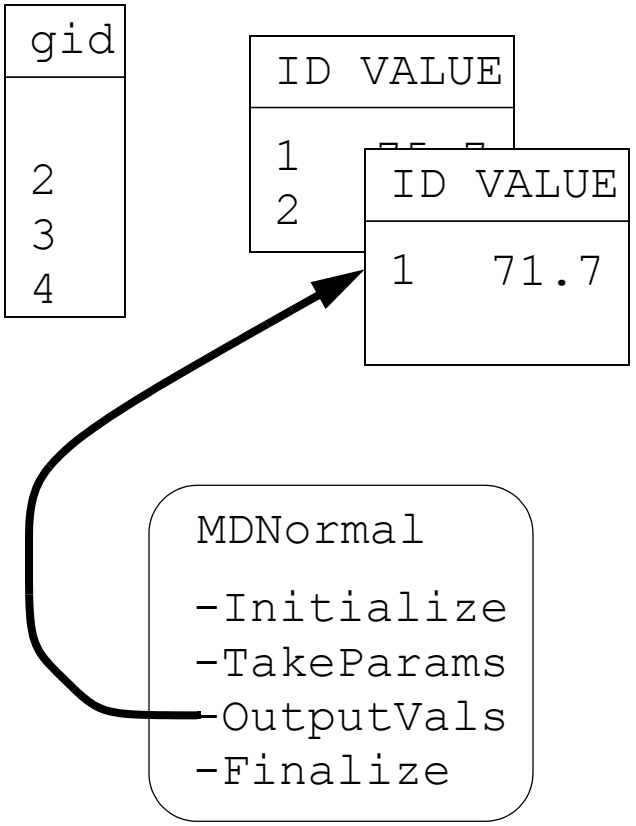
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

`OutputVals` is then called until a null

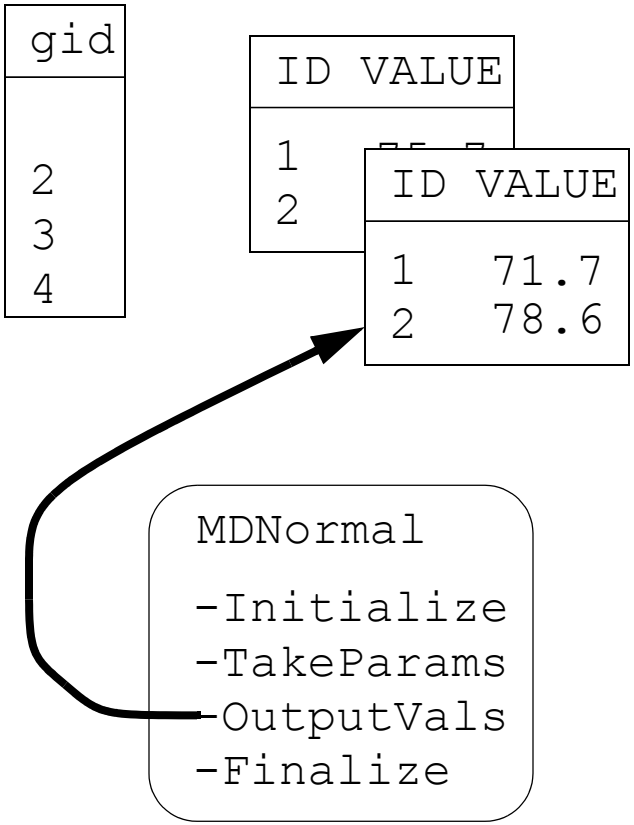
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

This completes MC iteration 1...
...new cycle of `OutputVals` calls gives next iter

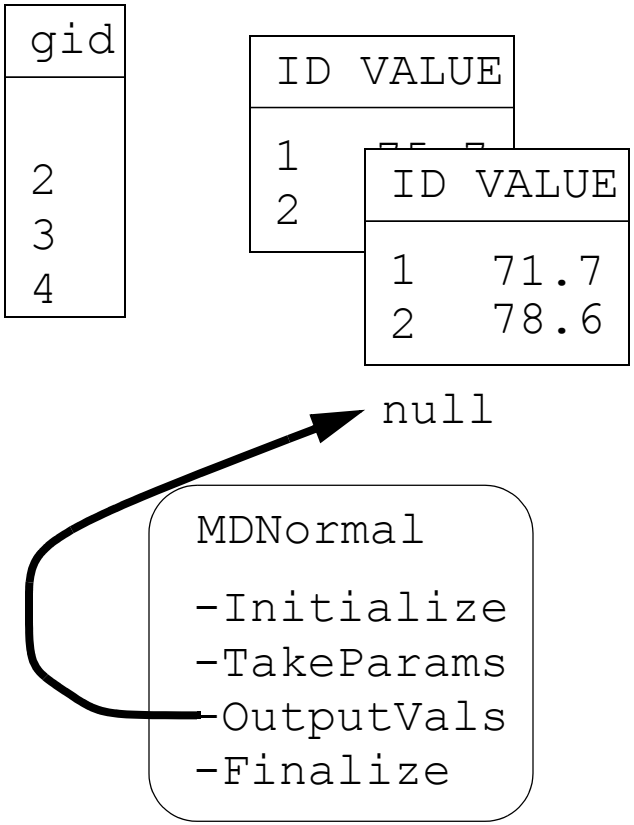
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

This completes MC iteration 1...
...new cycle of `OutputVals` calls gives next iter

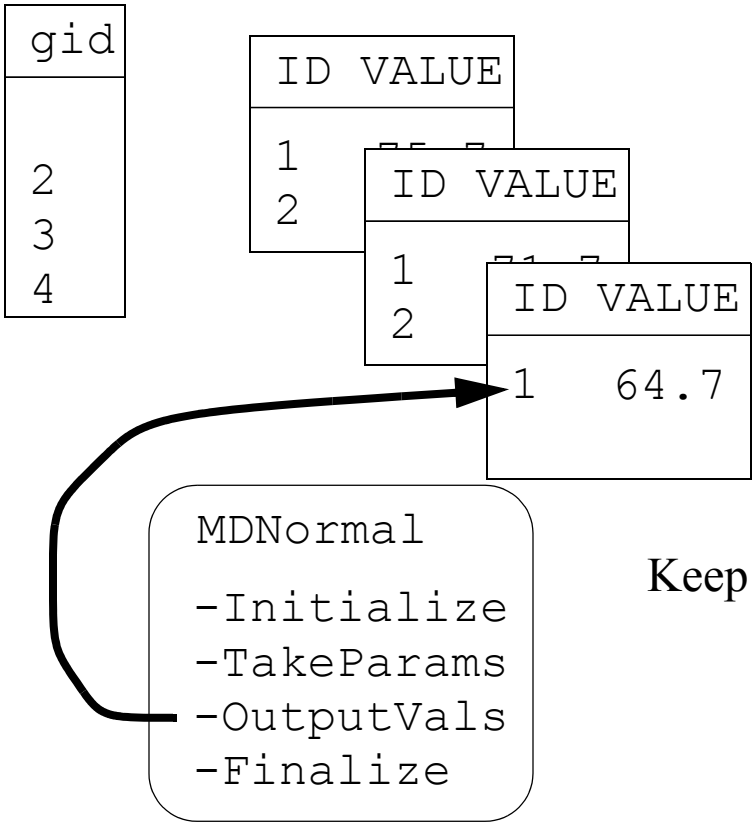
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

This completes MC iteration 1...
...new cycle of `OutputVals` calls gives next iter

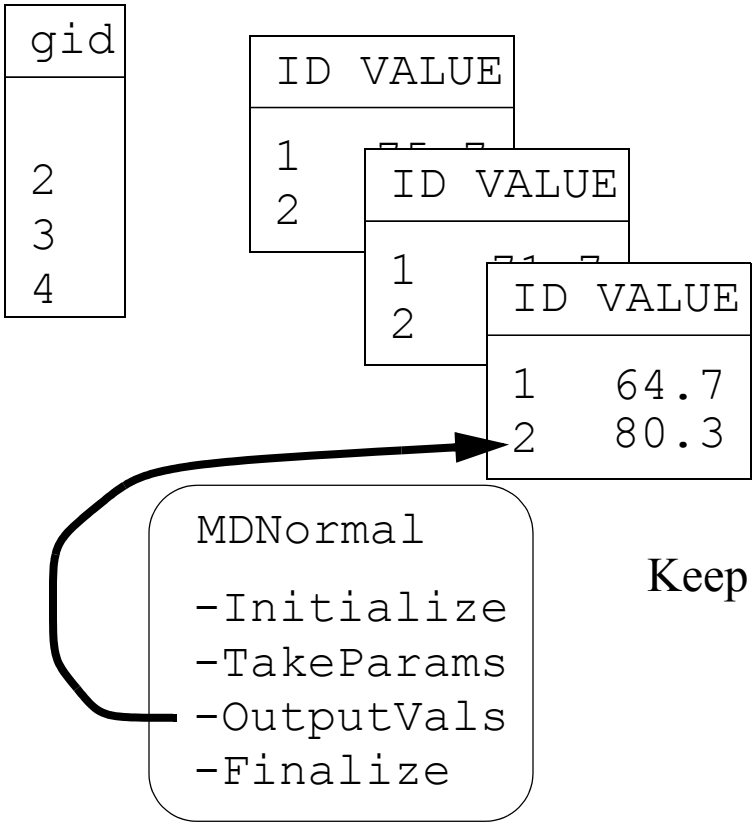
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
  SELECT s.ID, s.LAT, s.LONG, t.VALUE
  FROM SENSOR_PARAMS s, TEMP t
  WHERE s.ID = t.ID
```

Keep going to produce as many instances as needed

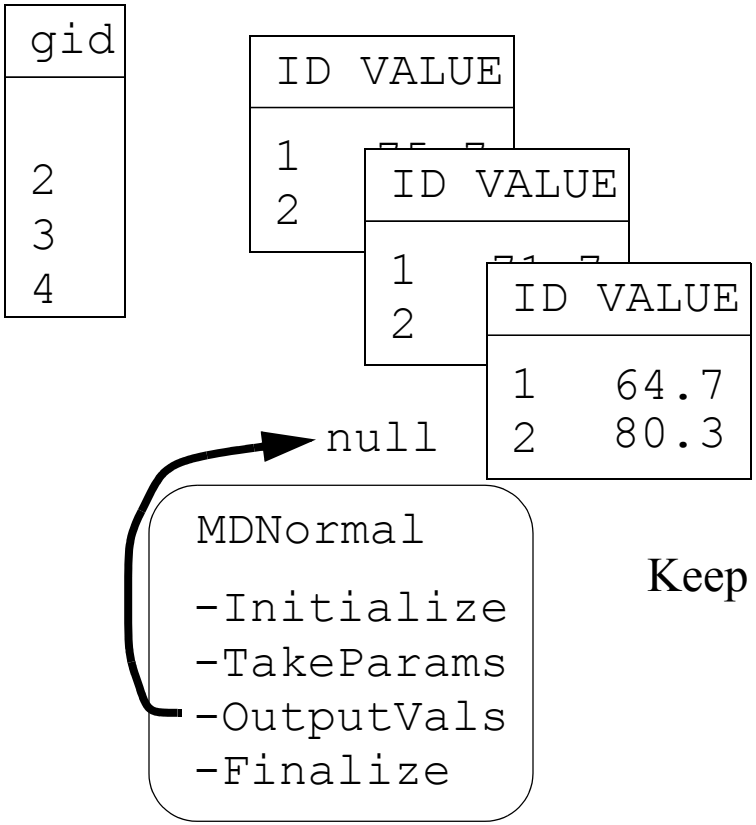
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
  SELECT s.ID, s.LAT, s.LONG, t.VALUE
  FROM SENSOR_PARAMS s, TEMP t
  WHERE s.ID = t.ID
```

Keep going to produce as many instances as needed

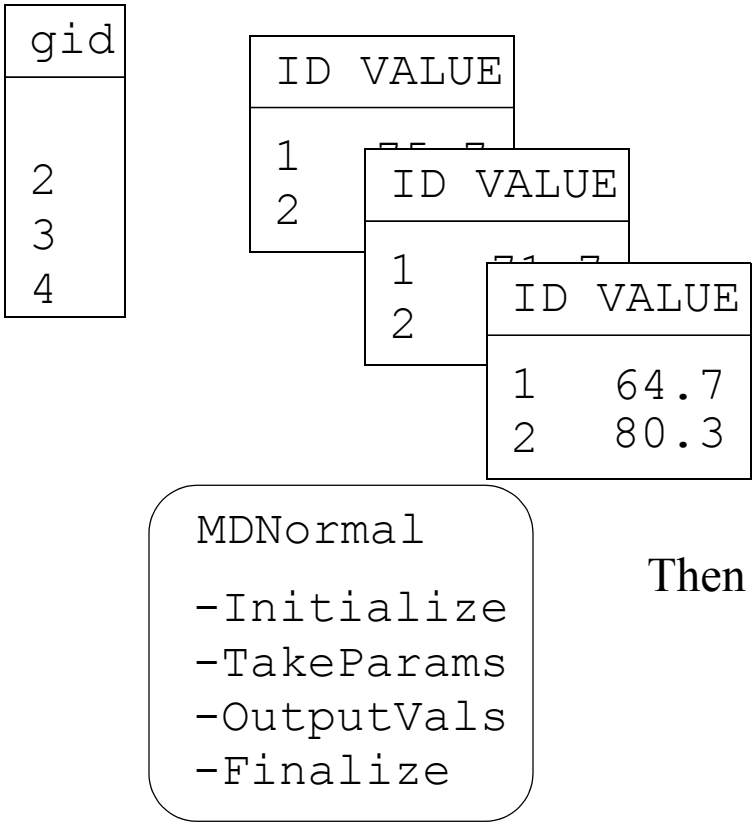
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
  SELECT s.ID, s.LAT, s.LONG, t.VALUE
  FROM SENSOR_PARAMS s, TEMP t
  WHERE s.ID = t.ID
```

Keep going to produce as many instances as needed

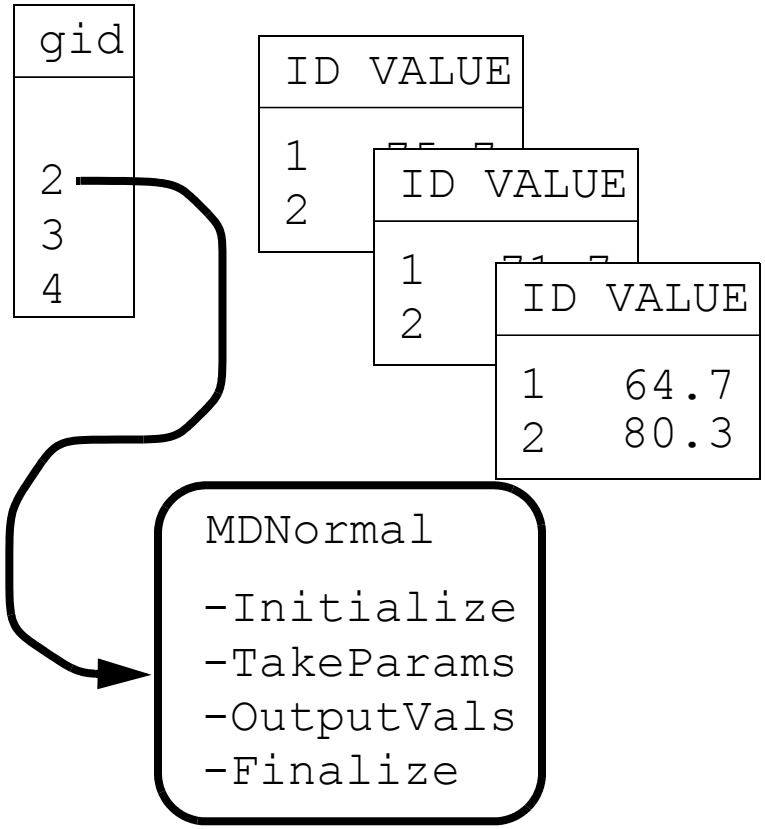
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
  SELECT s.ID, s.LAT, s.LONG, t.VALUE
  FROM SENSOR_PARAMS s, TEMP t
  WHERE s.ID = t.ID
```

Then a call to `Finalize` gets ready for `gid 2`

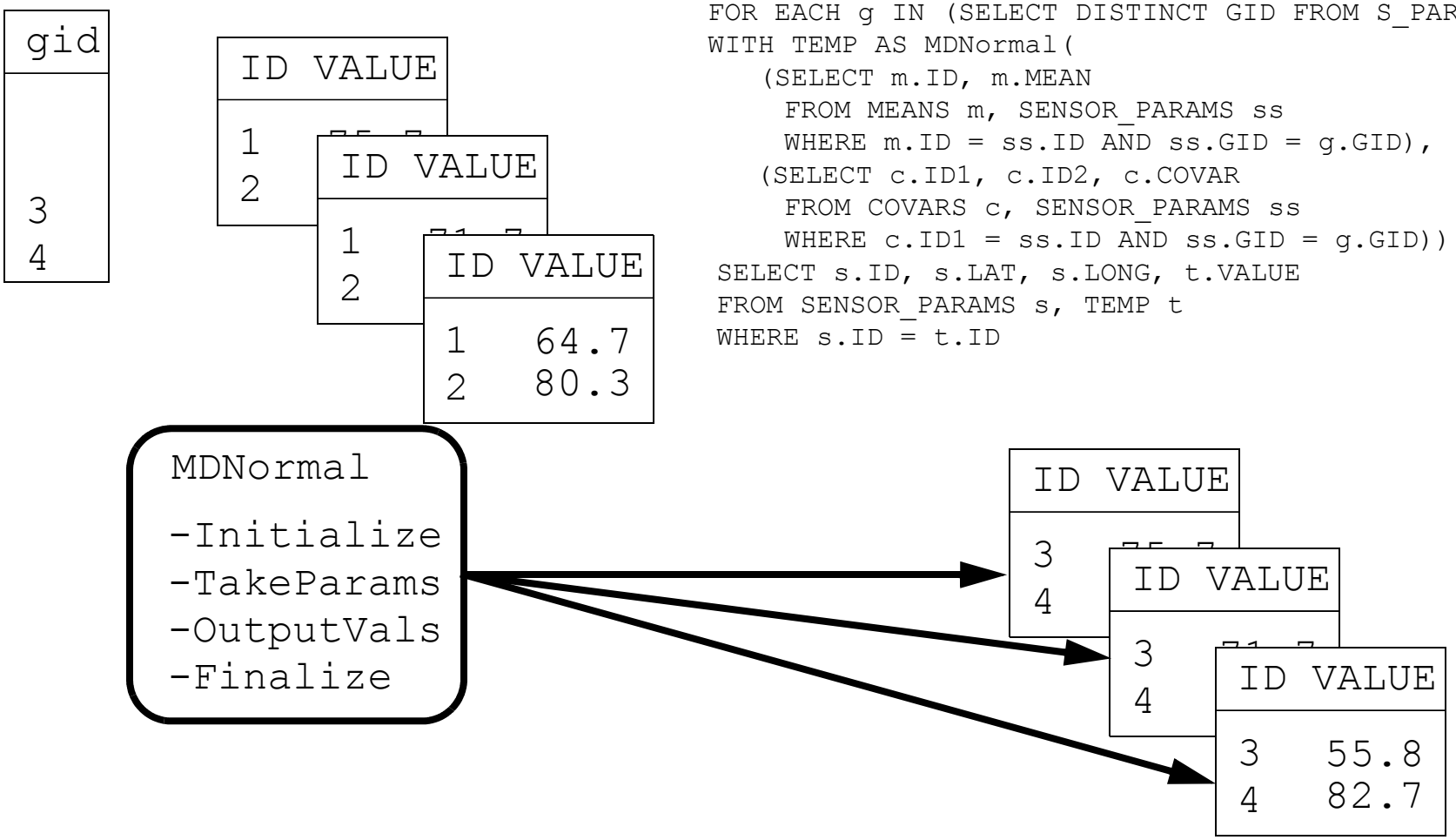
Example



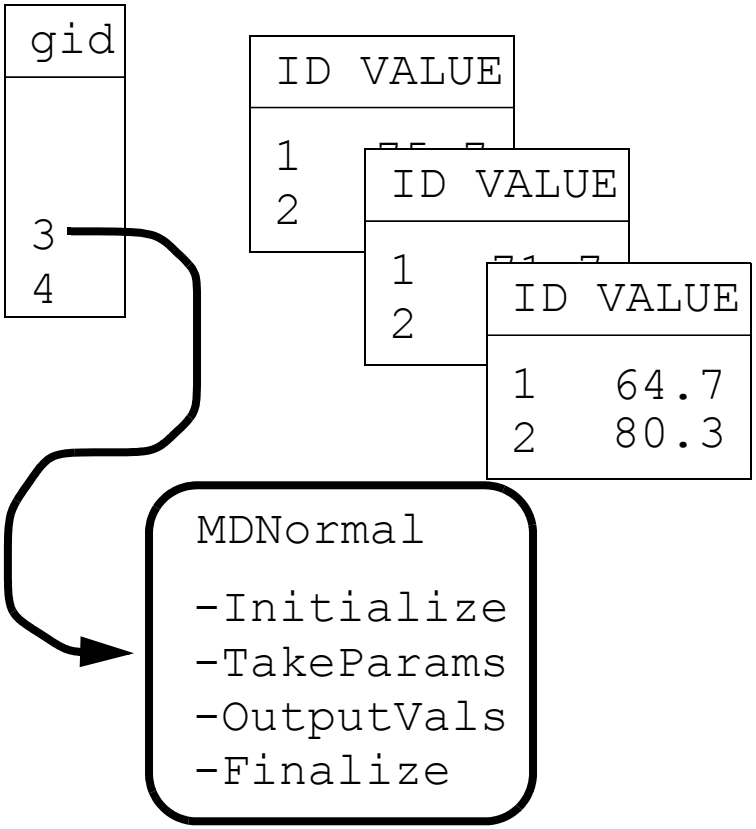

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
  SELECT s.ID, s.LAT, s.LONG, t.VALUE
  FROM SENSOR_PARAMS s, TEMP t
  WHERE s.ID = t.ID
```

Example

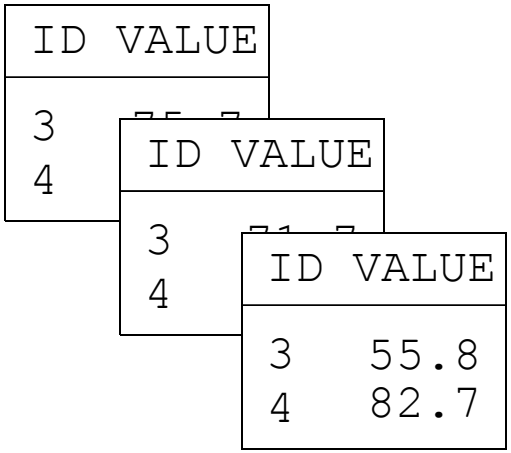
```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```



Example

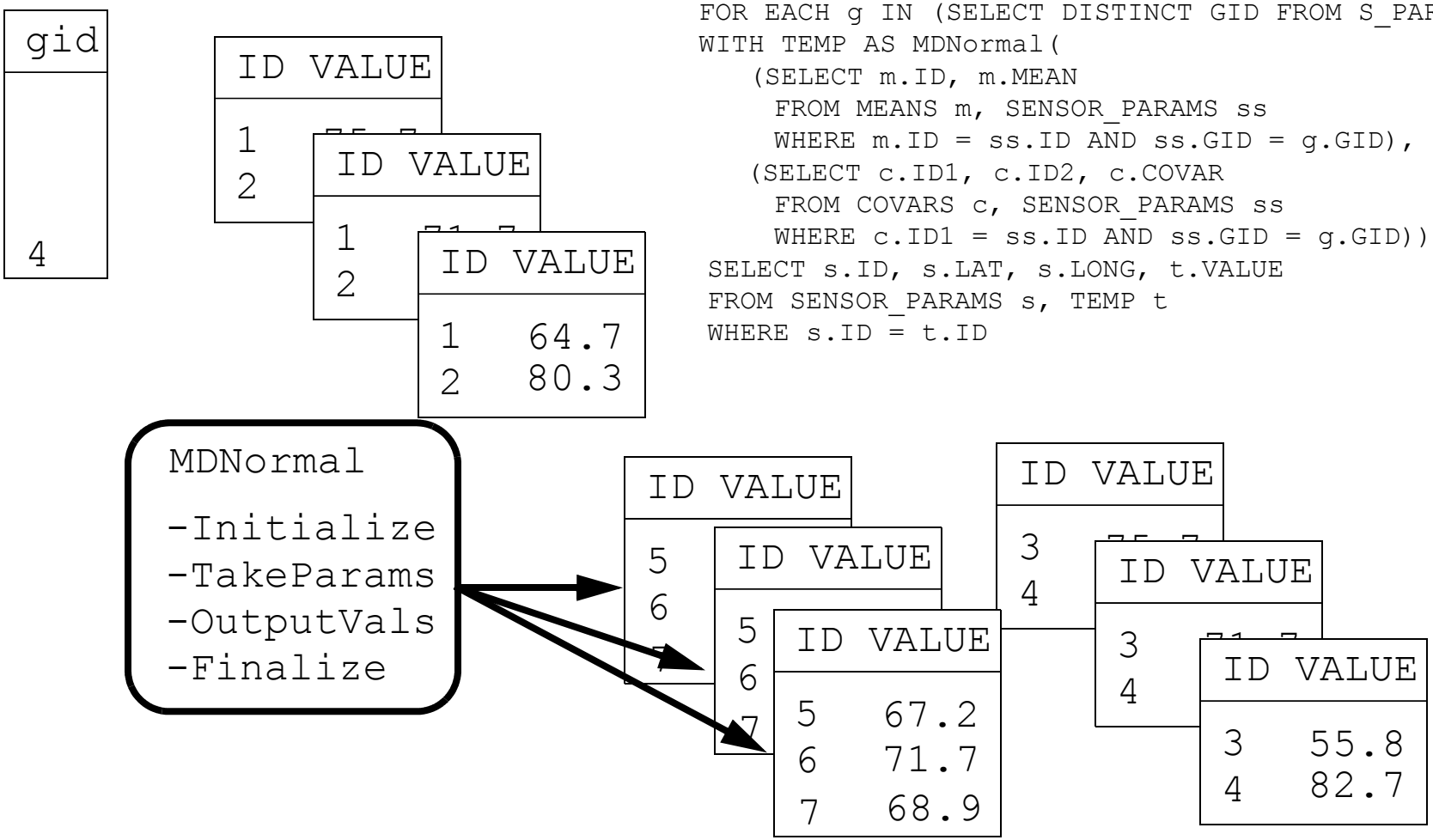



```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

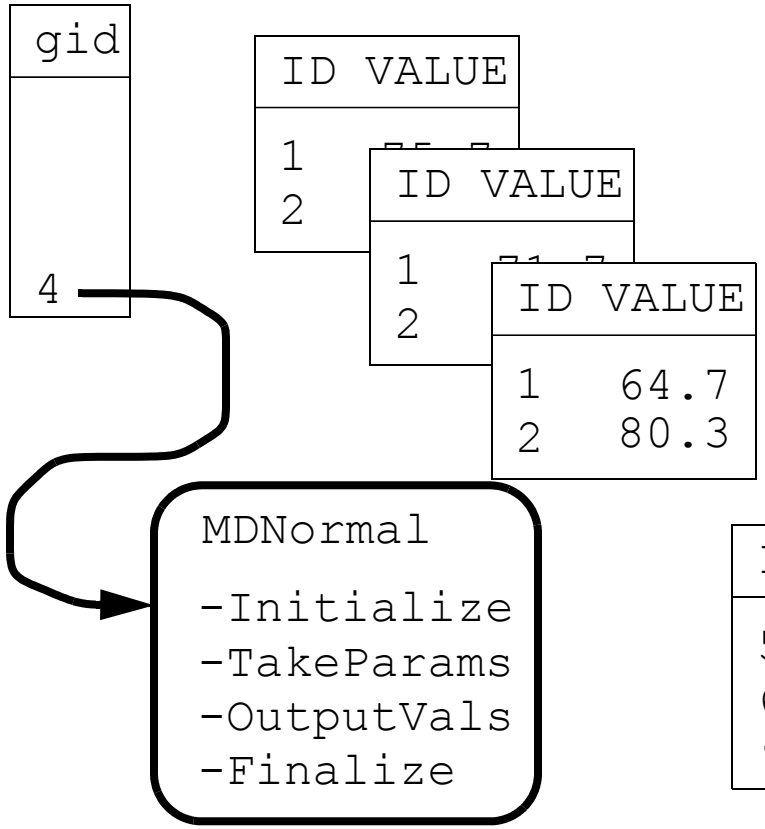


Example

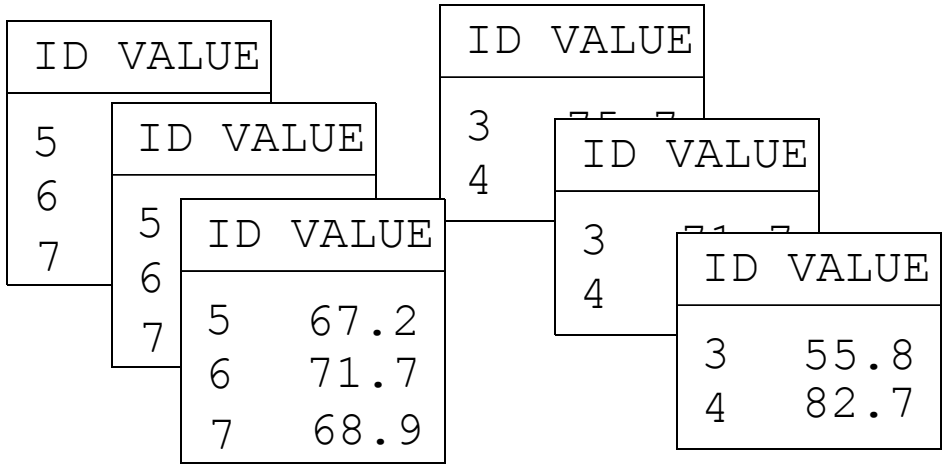
```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```



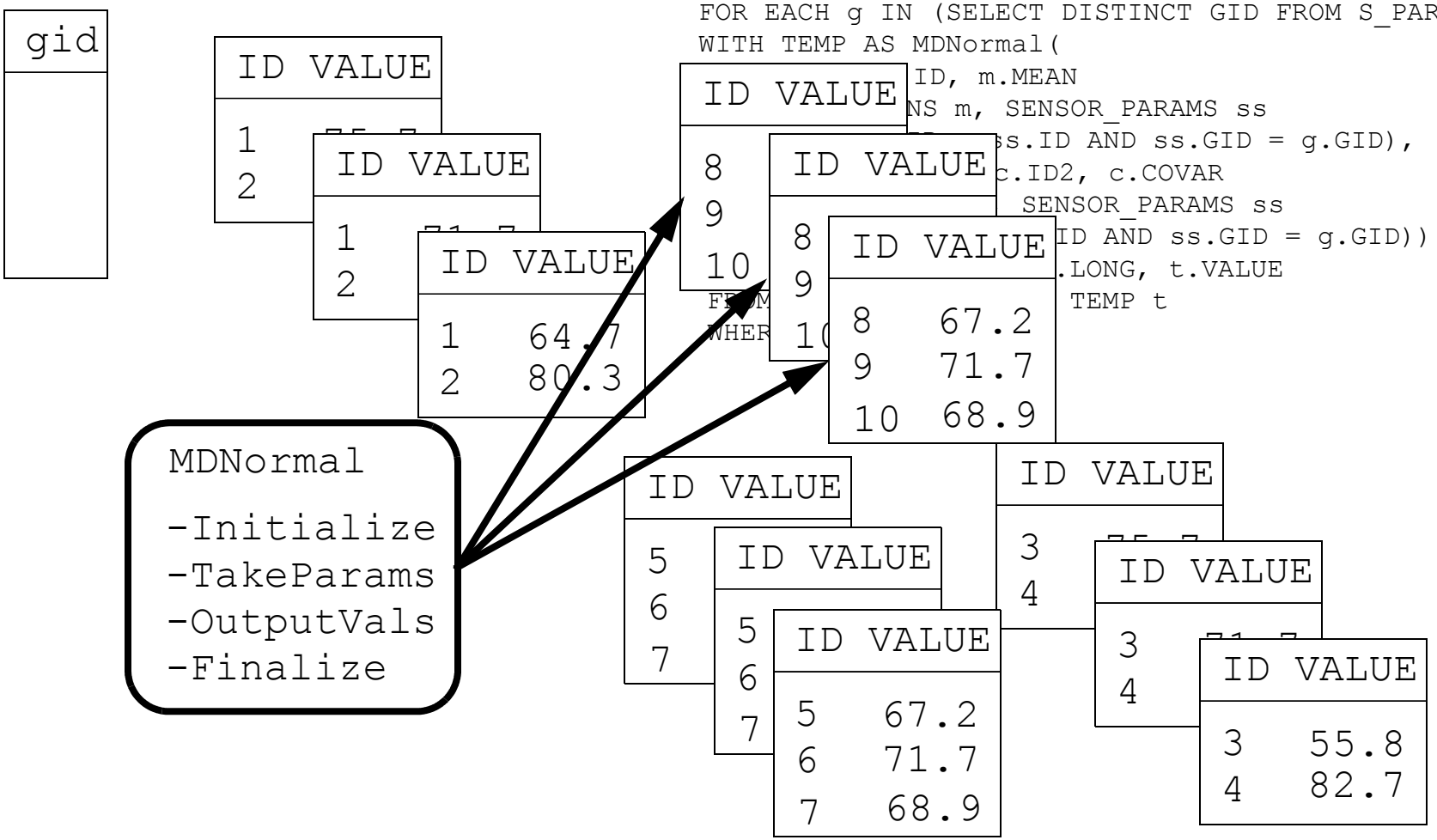
Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

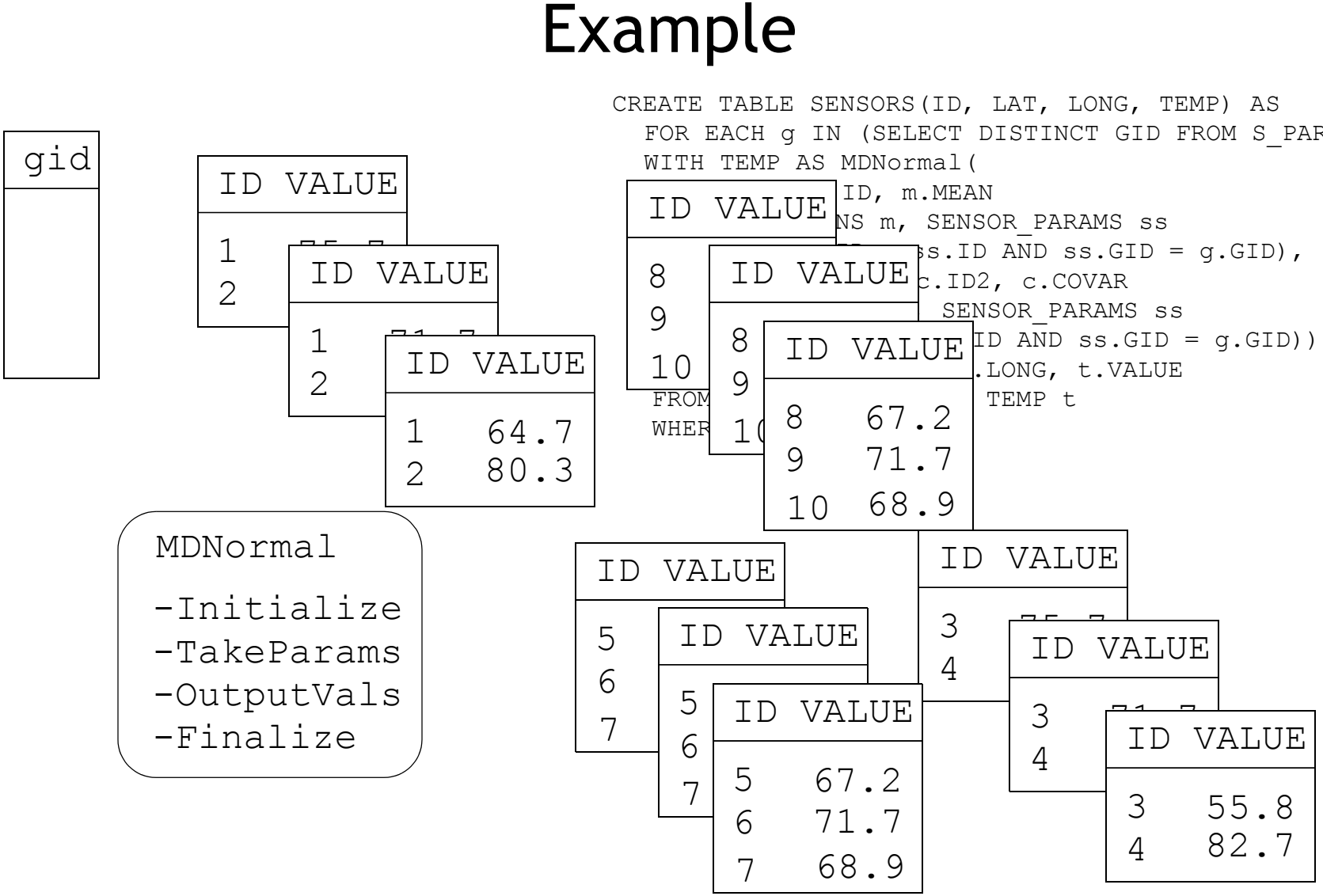


Example




```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
                ID, m.MEAN
              NS m, SENSOR_PARAMS ss
                ss.ID AND ss.GID = g.GID),
                c.ID2, c.COVAR
                 SENSOR_PARAMS ss
                ID AND ss.GID = g.GID))
              .LONG, t.VALUE
FR
WHER
                 TEMP t
```

Example


gid
ID VALUE
1
2
ID VALUE
1
2
ID VALUE
1 64.7
2 80.3
MDNormal
-Initialize
-TakeParams
-OutputVals
-Finalize
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
WITH TEMP AS MDNormal(
ID, m.MEAN
NS m, SENSOR_PARAMS ss
ss.ID AND ss.GID = g.GID),
c.ID2, c.COVAR
SENSOR_PARAMS ss
ID AND ss.GID = g.GID))
.LONG, t.VALUE
TEMP t
FROM
WHER
ID VALUE
8
9
10
ID VALUE
8
9
ID VALUE
8 67.2
9 71.7
10 68.9
ID VALUE
5
6
7
ID VALUE
5
6
7
ID VALUE
5 67.2
6 71.7
7 68.9
ID VALUE
3
4
ID VALUE
3
4
ID VALUE
3 55.8
4 82.7

Query Processing

- Clearly, don't want to generate 1000 DB instances, query each

Query Processing

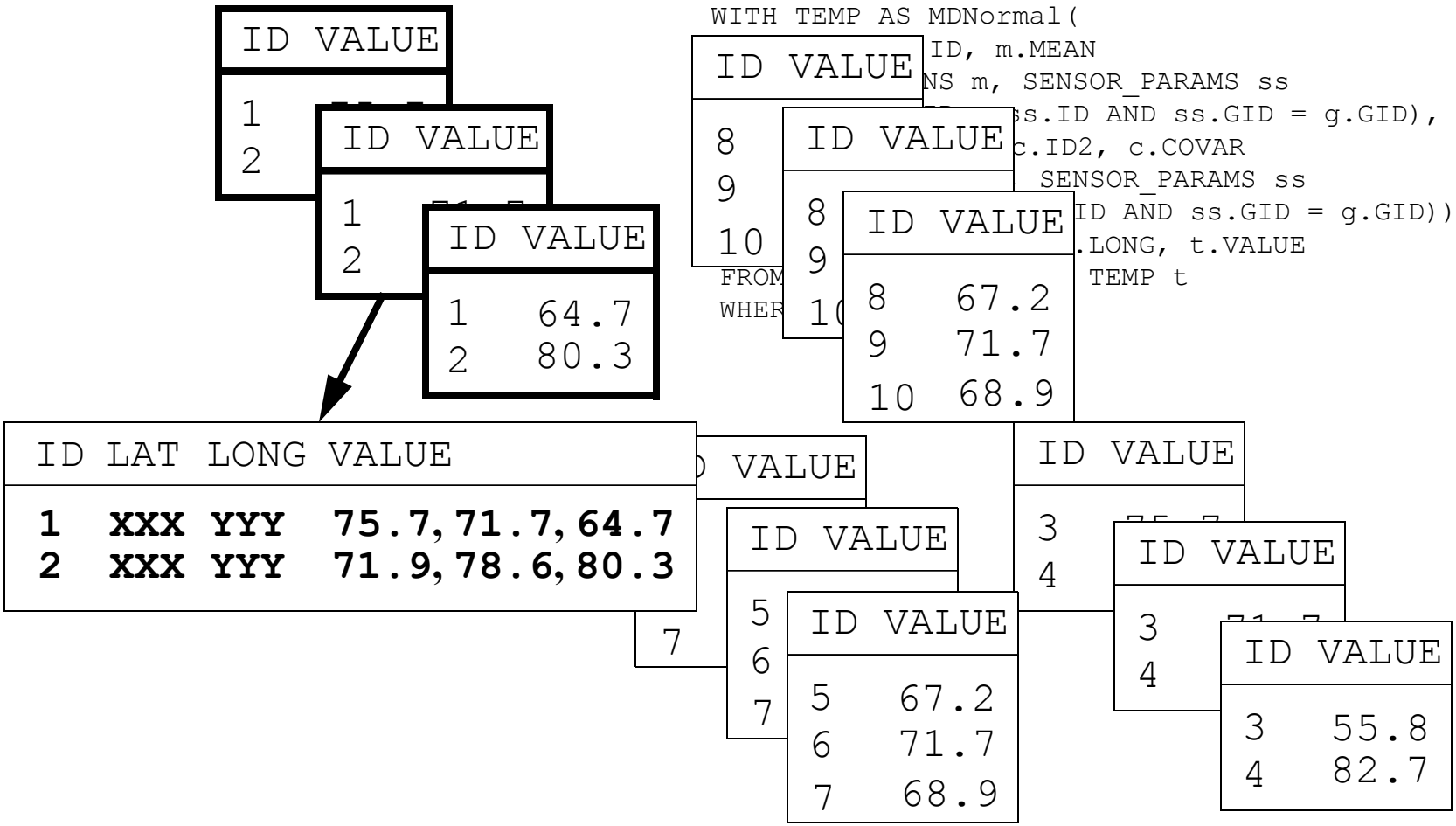
- Clearly, don't want to generate 1000 DB instances, query each
- MCDB groups DB instances into *tuple bundles*

Query Processing

- Clearly, don't want to generate 1000 DB instances, query each
- MCDB groups DB instances into *tuple bundles*
- These bundle constant values with multiple stochastic values
  - That way, can operate on all DB instances simultaneously

Tuple Bundles

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
                                ID, m.MEAN
                               NS m, SENSOR_PARAMS ss
                               ss.ID AND ss.GID = g.GID),
                               c.ID2, c.COVAR
                                SENSOR_PARAMS ss
                               ID AND ss.GID = g.GID))
                               .LONG, t.VALUE
  FROM                          TEMP t
  WHER
```



Tuple Bundles

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
                ID, m.MEAN
              NS m, SENSOR_PARAMS ss
                ss.ID AND ss.GID = g.GID),
                c.ID2, c.COVAR
                 SENSOR_PARAMS ss
                ID AND ss.GID = g.GID))
              .LONG, t.VALUE
  FROM          TEMP t
  WHER
```

ID	VALUE
8	
9	
10	

ID	VALUE
8	
9	
10	

ID	VALUE
8	67.2
9	71.7
10	68.9

ID	VALUE
3	
4	

ID	VALUE
3	
4	

ID	VALUE
3	55.8
4	82.7

ID	VALUE
5	
6	
7	

ID	VALUE
5	67.2
6	71.7
7	68.9

ID	LAT	LONG	VALUE
1	XXX	YYY	75.7, 71.7, 64.7
2	XXX	YYY	71.9, 78.6, 80.3
3	**XXX**	**YYY**	**55.8, 62.9, 58.8**
4	**XXX**	**YYY**	**84.7, 77.3, 82.7**

Tuple Bundles

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
                                ID, m.MEAN
                               NS m, SENSOR_PARAMS ss
                               ss.ID AND ss.GID = g.GID),
                               c.ID2, c.COVAR
                                SENSOR_PARAMS ss
                               ID AND ss.GID = g.GID))
                               .LONG, t.VALUE
  FROM                          TEMP t
  WHER
```

ID	VALUE
8	
9	
10	

ID	VALUE
8	
9	
10	

ID	VALUE
8	67.2
9	71.7
10	68.9

ID	LAT	LONG	VALUE
1	XXX	YYY	75.7, 71.7, 64
2	XXX	YYY	71.9, 78.6, 80
3	XXX	YYY	55.8, 62.9, 58
4	XXX	YYY	84.7, 77.3, 82
5	**XXX**	**YYY**	**75.7, 71.7, 67.2**
6	**XXX**	**YYY**	**71.9, 78.6, 71.7**
7	**XXX**	**YYY**	**55.8, 62.9, 68.9**
8	XXX	YYY	84.7, 77.3, 67.2
9	XXX	YYY	55.8, 62.9, 71.7
10	XXX	YYY	84.7, 77.3, 68.9

ID	VALUE
5	
6	
7	

ID	VALUE
5	
6	
7	

ID	VALUE
5	67.2
6	71.7
7	68.9

Tuple Bundles

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
                                ID, m.MEAN
                               NS m, SENSOR_PARAMS ss
                               s.ID AND ss.GID = g.GID),
                               c.ID2, c.COVAR
                                SENSOR_PARAMS ss
                               ID AND ss.GID = g.GID))
                               .LONG, t.VALUE
                                TEMP t
```

ID	VALUE
8	
9	
10	

ID	VALUE
8	
9	
10	

ID	VALUE
8	67.2
9	71.7
10	68.9

ID	LAT	LONG	VALUE
1	XXX	YYY	75.7, 71.7, 64.7
2	XXX	YYY	71.9, 78.6, 80.3
3	XXX	YYY	55.8, 62.9, 58.8
4	XXX	YYY	84.7, 77.3, 82.7
5	XXX	YYY	75.7, 71.7, 67.2
6	XXX	YYY	71.9, 78.6, 71.7
7	XXX	YYY	55.8, 62.9, 68.9
8	**XXX**	**YYY**	**84.7, 77.3, 67.2**
9	**XXX**	**YYY**	**55.8, 62.9, 71.7**
10	**XXX**	**YYY**	**84.7, 77.3, 68.9**

Tuple Bundles

```
CREATE TABLE SENSORS(ID, LAT, LONG, TEMP) AS
  FOR EACH g IN (SELECT DISTINCT GID FROM S_PARAM
  WITH TEMP AS MDNormal(
     (SELECT m.ID, m.MEAN
       FROM MEANS m, SENSOR_PARAMS ss
       WHERE m.ID = ss.ID AND ss.GID = g.GID),
     (SELECT c.ID1, c.ID2, c.COVAR
       FROM COVARS c, SENSOR_PARAMS ss
       WHERE c.ID1 = ss.ID AND ss.GID = g.GID))
   SELECT s.ID, s.LAT, s.LONG, t.VALUE
   FROM SENSOR_PARAMS s, TEMP t
   WHERE s.ID = t.ID
```

ID	LAT	LONG	VALUE
1	XXX	YYY	75.7, 71.7, 64.7
2	XXX	YYY	71.9, 78.6, 80.3
3	XXX	YYY	55.8, 62.9, 58.8
4	XXX	YYY	84.7, 77.3, 82.7
5	XXX	YYY	75.7, 71.7, 67.2
6	XXX	YYY	71.9, 78.6, 71.7
7	XXX	YYY	55.8, 62.9, 68.9
8	XXX	YYY	84.7, 77.3, 67.2
9	XXX	YYY	55.8, 62.9, 71.7
10	XXX	YYY	84.7, 77.3, 68.9

Tuple Bundles

- Benefit: constant time operations on constant attributes

ID	LAT	LONG	VALUE
1	XXX	YYY	75.7, 71.7, 64.7
2	XXX	YYY	71.9, 78.6, 80.3
3	XXX	YYY	55.8, 62.9, 58.8
4	XXX	YYY	84.7, 77.3, 82.7
5	XXX	YYY	75.7, 71.7, 67.2
6	XXX	YYY	71.9, 78.6, 71.7
7	XXX	YYY	55.8, 62.9, 68.9
8	XXX	YYY	84.7, 77.3, 67.2
9	XXX	YYY	55.8, 62.9, 71.7
10	XXX	YYY	84.7, 77.3, 68.9

Tuple Bundles

- Benefit: constant time operations on constant attributes
- Example: relational selection, where `ID >= 7`

ID	LAT	LONG	VALUE
1	XXX	YYY	75.7, 71.7, 64.7
2	XXX	YYY	71.9, 78.6, 80.3
3	XXX	YYY	55.8, 62.9, 58.8
4	XXX	YYY	84.7, 77.3, 82.7
5	XXX	YYY	75.7, 71.7, 67.2
6	XXX	YYY	71.9, 78.6, 71.7
7	XXX	YYY	55.8, 62.9, 68.9
8	XXX	YYY	84.7, 77.3, 67.2
9	XXX	YYY	55.8, 62.9, 71.7
10	XXX	YYY	84.7, 77.3, 68.9

ID	LAT	LONG	VALUE
8	XXX	YYY	84.7, 77.3, 67.2
9	XXX	YYY	55.8, 62.9, 71.7
10	XXX	YYY	84.7, 77.3, 68.9

Tuple Bundles

- Requires `IsPresent` attribute for ops on stochastic atts
- Example: relational selection where `VALUE > 70.0`

ID	LAT	LONG	VALUE	IsPresent
8	XXX	YYY	84.7, 77.3, 67.2	F, F, T
9	XXX	YYY	55.8, 62.9, 71.7	T, T, F
10	XXX	YYY	84.7, 77.3, 68.9	F, F, T

ID	LAT	LONG	VALUE
1	XXX	YYY	75.7, 71.7, 64.7
2	XXX	YYY	71.9, 78.6, 80.3
3	XXX	YYY	55.8, 62.9, 58.8
4	XXX	YYY	84.7, 77.3, 82.7
5	XXX	YYY	75.7, 71.7, 67.2
6	XXX	YYY	71.9, 78.6, 71.7
7	XXX	YYY	55.8, 62.9, 68.9
8	XXX	YYY	84.7, 77.3, 67.2
9	XXX	YYY	55.8, 62.9, 71.7
10	XXX	YYY	84.7, 77.3, 68.9

ID	LAT	LONG	VALUE
8	XXX	YYY	84.7, 77.3, 67.2
9	XXX	YYY	55.8, 62.9, 71.7
10	XXX	YYY	84.7, 77.3, 68.9

What Interesting Stuff Does MCDB Do?

- Efficient "value-at-risk" computation
  - — In worst (0.0001% case) how bad will my profits be?
  - — Uses novel "cloning" ideas from simulation area, plus MCMC
- Native hypothesis tests
  - — Which customers are 95% likely to decrease their purchase with a price hike?
  - — Requires generating only enough instances to answer the question
  - — Interesting foray into sequential hypothesis testing
- Recursive stochastic relation definitions
  - — Allows for full MCMC, time-tick simulations within MCDB
  - — ex: `INSTOCK`[*i*] depends on `PURCHASES`[*i* - 1], `INSTOCK`[*i* - 1], while `PURCHASES`[*i*] depends on `INSTOCK`[*i* - 1], `PURCHASES`[*i* - 1]

Thank You! Questions?