DW schema and the problem of views

- ▷ DW is multidimensional
- ▷ Schemas: Stars and constellations
- ▷ Typical DW queries, TPC-H and TPC-R benchmarks
- ▷ Views and their materialization

Main references [PJ01, C^{+}01, Joh02, Kot02]

Multidimensional Data Model

Warehouse data is often *thought of* as multidimensional, yet frequently *stored* as relations.

Multidimensional Data Model

- ▷ Multidimensional Data Model(s)
- ▷ Registrar’s Example

A Conceptual Multidimensional Data Model

Data are

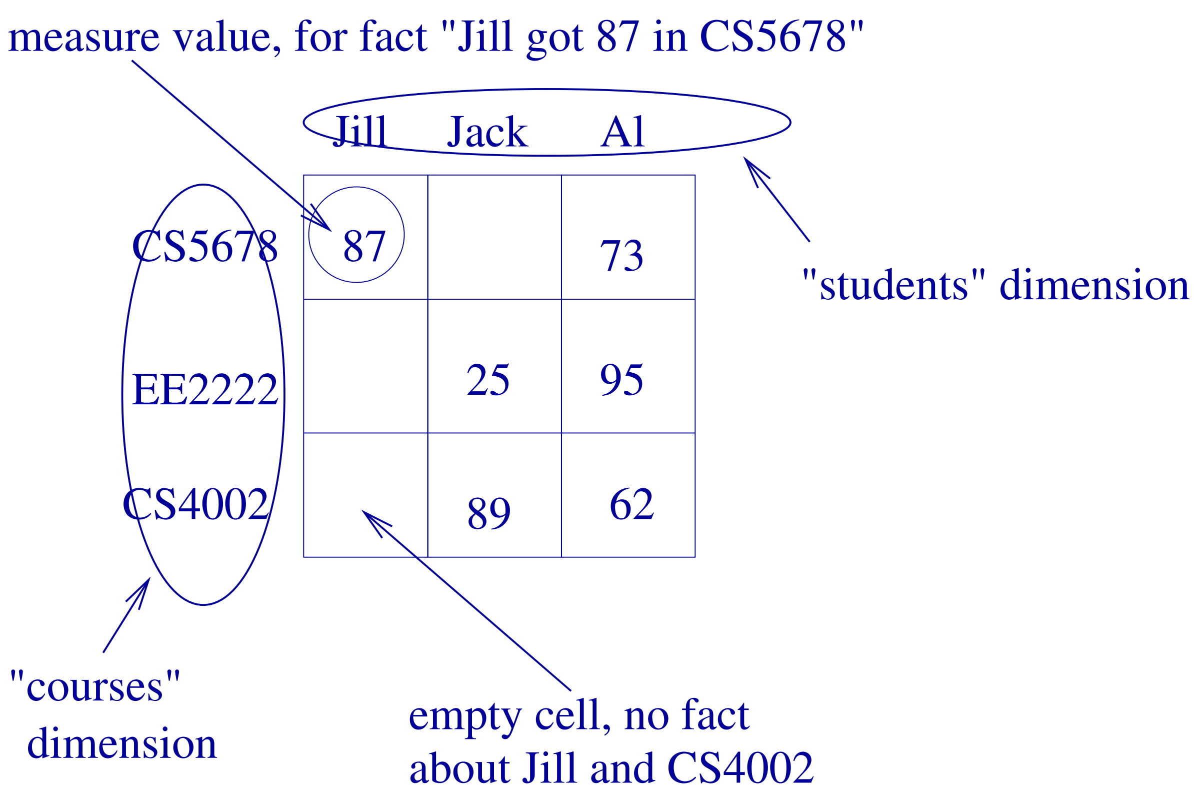
- ▷ facts, eg "Jill took CS5678"

  facts have numeric *measure values* (eg 87%)

- ▷ dimensions, eg "Jill is a *grad* student"

  "CS4002 is an *undergrad* course."

A *cube* represents facts, by mapping dimension combos to measures.


measure value, for fact "Jill got 87 in CS5678"
Jill
Jack
Al
CS5678
EE2222
CS4002
87
73
25
95
89
62
"students" dimension
"courses"
dimension
empty cell, no fact
about Jill and CS4002

More on cubes

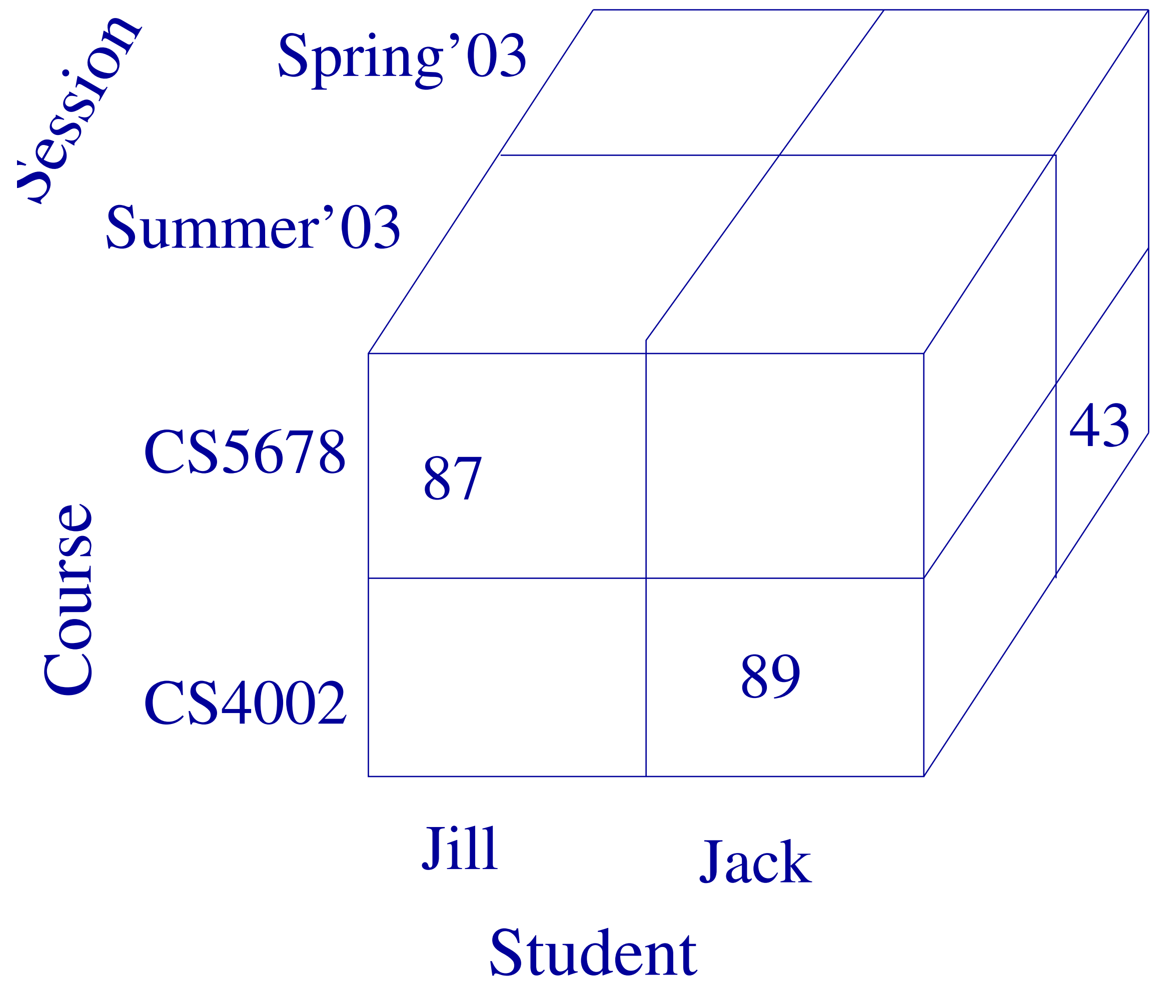
Cube generalizes idea of spreadsheet. Can have any number of dimensions.

Another measure for fact “Jill took CS5678” is the 503 times she yawned.

Registrar's Example

3-d cube, $\mathrm{Session} \times \mathrm{Student} \times \mathrm{Course} \to \mathrm{Grade}$

mapping session, student, course to grades.


Session
Spring'03
Summer'03
CS5678
87
43
Course
CS4002
89
Jill
Jack
Student

Registrar’s High-Dimensional Cube

Realistic cubes can have dozens of dimensions.

Registrar:

$$\mathrm{Session} \times \mathrm{Student} \times \mathrm{Course} \times \mathrm{Section} \times \mathrm{Campus} \times \mathrm{ForCredit} \rightarrow \mathrm{Grade}$$

Dimensions can have structure

Dimension values can be organized into containment hierarchies.

CS5678 is contained in “Grad Courses”

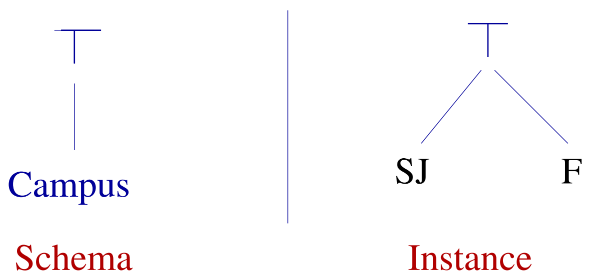
EE2222 and CS4002 are in “U/g Courses”

CS5678 and CS4002 are in “CS Courses”

Flat dimensions

$\top$ ("top") means "ALL". Simplest kind of dimension is "flat", in that every value is a child of $\top$.

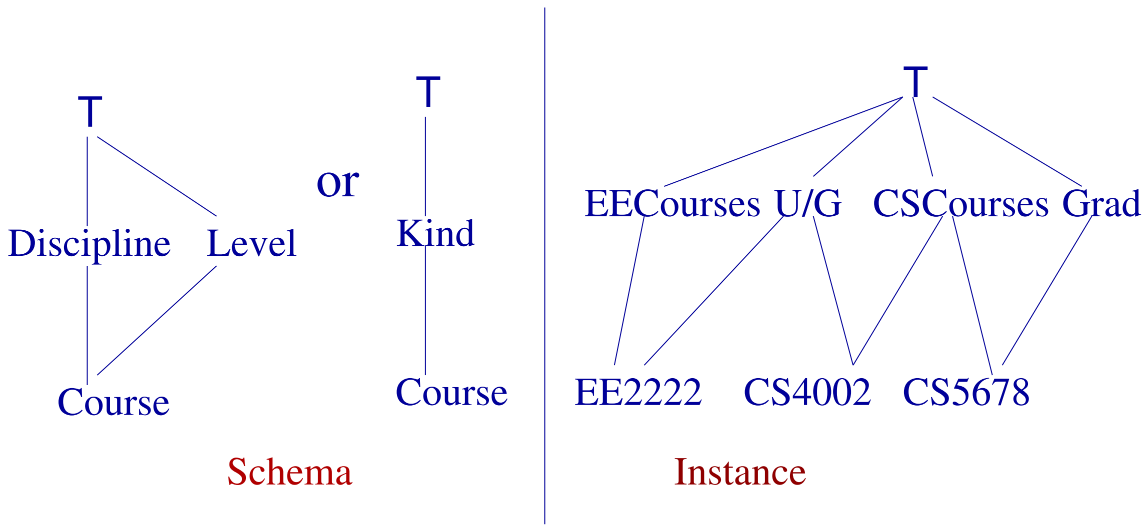
Eg, Campus values are "SJ" and "F". Or $\top$.



Schema ("Type info") vs Instance ("Values")

$\top$ overloaded as type and value; we have **3** Campus values!

Dimensions: can be tricky



How to handle this? Like multiple inheritance.

Some multidimensional models forbid general lattice: demand balanced tree structure in instance.

Measures are Interesting Too!

In Data Warehousing analyses, we aggregate.

Measure has

- ▷ set of “measure values” eg real numbers

  (report a numeric property of a fact)

- ▷ aggregation operation

  (eg SUM, AVG, MEDIAN)

  $\longrightarrow$ distributive ones are nicest (SUM).

Measures are Interesting Too! (part 2)

Best to consider the aggregation operation as “hard-coded” into a cube.

There might be more than one measure to keep track of: SUM and AVG for example.

Views

Conceptually, views are derived relations computed “on the fly”.

But high-performance database systems often secretly *materialize* (pre-compute, store and keep updated) views for later use.

It’s not the *analyst’s* business whether this is done. . . but it is a big issue for the DW *designer*.

Crudely, queries should be answered from the smallest summary table possible. Query-optimizer software should take care of this.

Aggregate Views

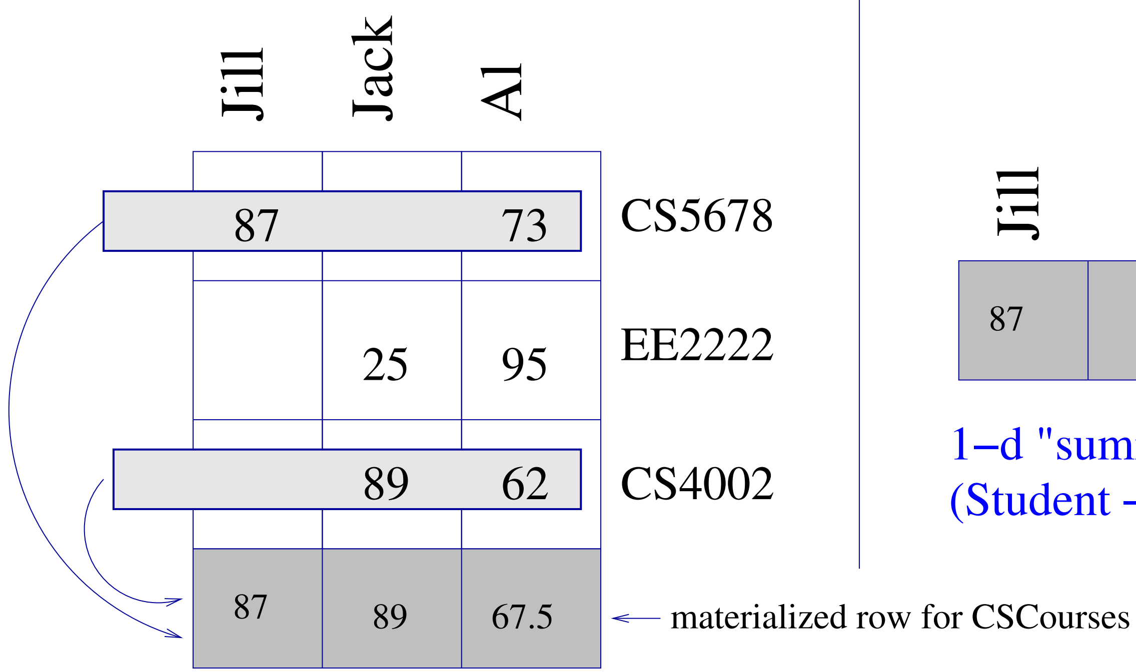
What about "CSCourses", we said it was a dimension value too!

But there are no facts *directly* about anybody's performance on CSCourses. They are views; compute them by averaging facts involving CS5678 and CS4002.

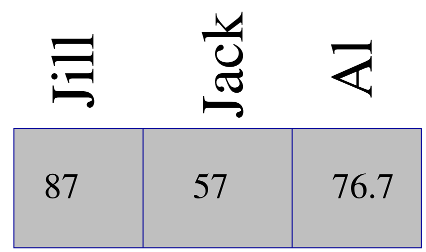
A materialized view (created by aggregation) is a *summary table*.

Summary tables usually have fewer dimensions than "base cube".

Base cube and a summary cube



2–d cube (Student x Course –> Grade)



1–d "summary table"
(Student –> Grade)

Aside: Materialized View Support

Major commercial DBMSs currently support materialized views.

Unfortunately, support has lagged in MySQL and PostgreSQL.

MySQL: no views at all till version 5. x4! new (&cheap) add-on product to support materialized views and rewrite queries to use them [EBM05].

PostgreSQL: had views a little earlier. “Do It Yourself” materialization is possible via the postgres scripting language and triggers (see Gardner [Gar04]). Gardner also project lead of *matview Project* for PostgreSQL [mP03]. Project appears inactive.

Basic Operations

An important part of any data model is “what operations are provided?”. Deferred to a later lecture.

Soon, we describe relational *storage*, and a class of SQL queries.

Other multidimensional models

Many different multidimensional models.

Some models, eg., Thomsen’s “LC” [Tho02] are not as rigid about separating “dimensions” from “measures”.

LC distinguishes valid data from “missing” and “meaningless”.

Storing cubes in a relational database

Relational technology is mature, robust, available.

But somehow, we must map cubes into world of tuples and tables.

- ▷ Star Schema (now)
- ▷ Snowflake Schema (later lecture)
- ▷ Constellation (now)

Star Schema

- One main fact table
- one dimension table per dimension.

Fact table stores measure value AND for each dimension, a *foreign key* into the corresponding dimension table.

Dimension table stores name of (lowest level) dimensional value *and* all higher-level dimension values that contain it.

Dimension table's key is a "dummy value" and might be the record-id number.

Star Schema Example

Name comes from the fact table in the centre, with all the dimension tables radiating out from the centre.

Students

Student	Kind	sID
Jill	grad	0
Jack	grad	1
Al	u/g	2

←

Facts

sID	Grade	cID
0	87	0
0	73	2
1	25	1
1	95	2
2	89	1
2	62	2

→

Courses

cID	Course	Discipline	Level
0	CS5678	CSCourses	g
1	EE2222	EECourses	u
2	CS4002	CSCourses	u

(no column necessary for $\top$ in dim tables.)

Constellation

A *fact constellation*: several fact tables sharing common dimensions.

Eg, Parking-office fact table

Student $\times$ Campus $\longrightarrow$ Number of Parking Violations

“Student” dimension, at least, would be same as with registrar’s star

DW Queries

Aggregation is the tool to “see the forest through the trees”.

Common form of SQL query is

```
SELECT attribs, aggr-expr...
FROM fact,dim1,dim2,...WHERE ...
GROUP BY attribs
```

SELECT kind, avg(Grade) FROM Facts, Students

WHERE Facts.sID = Students.sID

GROUP BY kind

DW Query Example, Cont.

SELECT kind, avg(Grade) FROM Facts,Students

WHERE Fact.sID = Students.sID GROUP BY kind

is probably implemented by a star join between Facts and Students:

Student	kind	sID	Grade	cID
Jill	grad	0	87	0
Jill	grad	0	73	2
Jack	grad	1	25	1
Jack	grad	1	95	2
Al	u/g	2	89	1
Al	u/g	2	62	2

then

kind	avg(Grade)
grad	70
u/g	75.5

DW Query Example: Using Summary Tables

SELECT kind, avg(Grade) FROM Facts,Students
WHERE Fact.sID = Students.sID GROUP BY kind

But suppose that we already had a summary table giving the average grade for each student:

Student	kind	sID	Avg(Grade)
Jill	grad	0	80
Jack	grad	1	60
Al	u/g	2	75.5

Then, we might try to average Jack's avg grade with Jill's, to get an average grad grade. *Might* be faster.

But avg() is not distributive! (eg Jack took 1 course, Jill took 10).

Summary Tables: When faster for queries?

Summary table typically much smaller than the fact table.

When you *can* correctly answer a query from a summary table, speed *might* be helped by smaller data.

. . . But the main fact tables may have fast index built for it, which the summary table lacks.

Moral: query optimization is tricky.

Aside: Performance benchmarks for DW systems

The hardware, OS, and database all interact and are crucial for overall DW/OLAP performance. Cannot easily separate issues.

"Transaction Processing Council": neutral party that evaluates (commercial) DB systems. See `http://www.tpc.org`

- Their TPC-H benchmark simulates an ad-hoc (OLAP-style) environment.

  Their TPC-R benchmark simulated "canned" queries typical of report generators. Declared obsolete in 2005.

-

Aside: Performance for DW systems, 2

TPC-H computes “Composite Queries-per-hour” at various database sizes.

As of 26 Feb 2006, for a 10TiB warehouse, champion performance is Oracle database on a SunFire platform running Solaris 10.

For a 100GiB warehouse, champ is SQL Server 2005 on a HP Proliant running Windows Server 2003.

Source: `http://www.tpc.org/tpch/results/tpch_perf_results.asp`

Microsoft/Intel strong on lower end, commercial databases on Linux in the midsize, large multiprocessors running commercial UNIX and Oracle/DB2 at upper end.

Aside: Factoring price into TPC-H

TPC-H also rates systems by Price/Performance. (System cost/compsite queries per hour). Again, affected by database size.

Source: `http://www.tpc.org/tpch/results/tpch_price_perf_results.asp`

10TiB: typically $6M USD, large multiprocessor. Saw one $14M.

100GiB: typically $100k 4-cpu system

Winners:

10TiB, 100GiB champs are the performance champs.

1TiB, champ is SQL Server 2005 on Bull NovaScale with 16 Itanium-2 CPUs, running Windows 2003. Cost: abt $400k.

Choosing summary tables in DW design

DW designer (or database system) needs to decide which summary tables to use. Issues:

- ▷ identifying candidates
- ▷ potential size of each
- ▷ potential query benefit of each
- ▷ potential cost of keeping each updated

Do we know the queries that will be asked?

Identifying Candidate Summary Tables

Common approach: consider summaries that are

▷ group-bys

▷ of some subset

▷ of fact-table columns

Candidates belong to the "data cube" [GBLP97].

Data-Cube Aggregates

Eg, for Facts(sID, cID, Grade) the possible candidates are

1. Select sID, cID, avg(Grade) from Facts group by sID, cID (= Facts!)
2. Select sID, avg(Grade) from Facts group by sID
3. Select cID, avg(Grade) from Facts group by cID
4. Select avg(Grade) from Facts

Note that if we choose to materialize 2 and 4, we can first compute 2.

Then 4 from 2.

- (Likewise, 4 can be computed from 3)

Sizes of Summary Tables

Various sophisticated techniques can try to predict the size of a summary table. Deferred till next week.

Benefit of Summary Table

Basically, depends on smallness

. . . and how often it can be used to answer queries.

Deferred till next week.

References

[C+01] Surajit Chaudhuri et al. Database technology for decision support systems. *IEEE Computer*, pages 48–55, December 2001.

[EBM05] EBM Software BV. X4! online at http://x4.olap4all.com/, 2005. checked 25 February 2006.

[Gar04] Jonathan Gardner. Materialized views in PostgreSQL. online, http://www.jonathangardner.net/PostgreSQL/materialized_views/matviews.html, 2004. checked 25 February 2006.

[GBLP97] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: a relational aggregation operator generalizing group-by, cross-tabs and subtotals. Number 1, pages 29–53, 1997.

[Joh02] Theodore Johnson. Data warehousing. In J. Abello et al., editors, *Handbook of Massive Data Sets*, chapter 19, pages 661–710. Kluwer, 2002.

[Kot02] Yannis Kotidis. Aggregate view management in data warehouses. In J. Abello et al., editors, *Handbook of Massive Data Sets*, chapter 20, pages 711–741. Kluwer, 2002.

[mP03] matview Project. Implementation of materialized views for PostgreSQL.

online: http://gborg.postgresql.org/project/matview/projdisplay.php, 2003. checked 25 February 2006.

[PJ01] Torben Bach Pedersen and Christian S. Jensen. Multidimensional database technology. *IEEE Computer*, pages 40–46, December 2001.

[Tho02] Erik Thomsen. *OLAP Solutions*. Wiley, 2nd edition, 2002.