Successful and Cost Effective Data Warehouse... The MySQL Way

Ivan Zoratti
Sales Consulting Manager EMEA (MySQL)
Agenda

- Intro
- The Collision of Two Worlds
- Data Warehousing Terminology
- OLAP Strategy and Implementation
- ROLAP Cookbook
- Benchmarks, really really?
Let me introduce...

The MySQL Sales Consulting Team
The Collision of Two Worlds

Credit: David A. Aguilar (Harvard-Smithsonian CfA)

Wednesday, 14 April 2010
Analytical and Business Analysis

- DBAs and Data Architects
- Large Servers
- Loads of Cache and Memory
- Parallel Queries
- Shared Storage

- DBAs and SW Developers
- Commodity Servers
- Limited memory resources
- Sharding and Replication
- Internal Storage
The “Enterprise Approach”

• Organized Systems
  – OLTPs, Operational Data Stage, Data Warehouses, Data Marts, Report Systems

• Organized Tools
  – ETL (Extraction, Transformation and Loading), Reporting, Business Intelligence, Analytical Functionality, Data Mining

• Organized Roles
  – Software Developers, Database Administrators, Data Architects, Data Analysts, Report Designers, Power Users, Standard Users

• Organized Data
  – Normalized Model, Denormalized Model, Star Schema, Snow Flake Schema
  – Raw data, Data Transformation, Data Consolidation, Data Certification, Usable Information
The “Web Approach” - 1

• Your Data Warehouse is a constant Beta Version

• New Systems Needs
  – OLTPs cannot be used for heavy reporting and analysis
  – Data needs to be transformed and multi-source information must be amalgamated
  – A single system is too big and it does not satisfy all requirements
  – Reporting and analysis are different

• New Tools Needs
  – Data need transformation, it cannot be simply copied from one server to another
  – Multiple sources and a workflow are required
  – SQL and ad-hoc reports are not enough
  – Users want to “slice and dice” data on the fly
  – Users do not know which information they will find
The “Web Approach” - 2

• New Roles Needs
  – Software Developers do not have enough information and knowledge to prepare complex reports
  – New data must be “created” to comply with business requirements
  – Technology and Business users must speak a common language
  – Not all the users have similar requirements

• New Data Needs
  – A normalized model does not satisfy complexity and performance requirements
  – Raw data does not provide enough valuable information for the user
  – Data cannot simply be modified on the same server
  – Data from multiple sources must be combined
  – Data must be certified. Checkpoints are required
The stages of data warehousing

1. No needs
   - Beta stage or focused on realising the core application asap

2. Reporting Needs
   - The core app requires some reporting
   - The business users want to know how things are going

3. Multiple Systems and Data consolidation
   - Reporting and analysis needs are becoming important
   - There are multiple servers and application, data must be consolidated to provide a uniform view

4. Real Business Intelligence
   - Business users require more in depth analysis. They need to slice and dice their data to understand market trends, set a business strategy etc.

5. Advanced Analysis
   - The business is becoming more sophisticated, so must be the view of the collected information.
Data Warehousing

• One term, different meanings

“A generic collection of systems and data designed to provide information for reporting, data analysis and decision support”

• OLAP - OnLine Analytical Processing

“OLAP transforms Data Warehouse data into strategic information”

– ROLAP - Relational Online Analytical Processing
Common Cases with MySQL

- **Data Mart** - Small, semi real-time data marts
- **Real-Time** - Continuous, real-time/query data warehousing
- **Reporting** - Traditional, standard reporting warehouse
- **Historical** - Massive historical, with ad-hoc queries warehouse
- **Analytical** - BI, analytic in OLTP applications
OLAP Strategy and Implementation
The MySQL Data Warehousing Ecosystem

ETL
BI/REPORTING TOOLS
INTEGRATION

RDBMS

STORAGE ENGINE

PLATFORM

Wednesday, 14 April 2010
Technologies for Data Warehousing

• Multiple Storage Engines
  – InnoDB, MyISAM, Archive, Federated, CSV
  – Column-based 3rd party engines

• Caching
  – Query Cache, Key Buffer, Buffer Pool, Memcached

• Partitioning

• Active-Active Servers with Shared Storage

• Replication and Sharding
Data Warehousing Cases and Technology Mapping

**Data Mart**
- Columnar Engines
- InnoDB, MyISAM, Federated, CSV, Archive
- Query Cache, Memcached
- Active-Active Replication, Sharding

**Real-Time**
- Columnar Engines
- InnoDB, MyISAM, Federated, CSV, Archive
- Query Cache, Memcached
- Active-Active Replication, Sharding

**Reporting**
- Columnar Engines
- InnoDB, MyISAM, Federated, CSV, Archive
- Query Cache, Memcached
- Active-Active Replication, Sharding

**Historical**
- Columnar Engines
- InnoDB, MyISAM, Federated, CSV, Archive
- Query Cache, Memcached
- Active-Active Replication, Sharding

**Analytical**
- Columnar Engines
- InnoDB, MyISAM, Federated, CSV, Archive
- Query Cache, Memcached
- Active-Active Replication, Sharding

ORACLE
Optimized Schemas

**STAR SCHEMA**

**SNOW FLAKE SCHEMA**

Wednesday, 14 April 2010
Multidimensional Structure

- Customers
- Geography
- Time
- Products

Orders
CREATE TABLE LU_CUSTOMER (  
  CUSTOMER_ID int(10) NOT NULL DEFAULT '0',  
  CUST_LAST_NAME varchar(255) DEFAULT NULL,  
  CUST_FIRST_NAME varchar(255) DEFAULT NULL,  
  GENDER_ID int(10) DEFAULT NULL,  
  CUST_BIRTHDATE datetime DEFAULT NULL,  
  EMAIL varchar(255) DEFAULT NULL,  
  ADDRESS varchar(255) DEFAULT NULL,  
  ZIPCODE varchar(255) DEFAULT NULL,  
  INCOME_ID int(10) DEFAULT NULL,  
  CUST_CITY_ID int(10) DEFAULT NULL,  
  AGE_YEARS double(7,2) DEFAULT NULL,  
  AGERANGE_ID int(10) DEFAULT NULL,  
  MARITALSTATUS_ID int(10) DEFAULT NULL,  
  EDUCATION_ID int(10) DEFAULT NULL,  
  HOUSINGTYPE_ID int(10) DEFAULT NULL,  
  HOUSEHOLDCOUNT_ID int(10) DEFAULT NULL,  
  PLAN_ID int(10) DEFAULT NULL,  
  FIRST_ORDER datetime DEFAULT NULL,  
  LAST_ORDER datetime DEFAULT NULL,  
  TENURE double(7,2) DEFAULT NULL,  
  RECENTY double(7,2) DEFAULT NULL,  
  STATUS_ID int(10) DEFAULT NULL,  
  PRIMARY KEY (CUSTOMER_ID)  
  KEY ixCustomer01 (CUST_CITY_ID)  
) ENGINE=MyISAM
CREATE TABLE LU_CUSTOMER (
  CUSTOMER_ID int(10) NOT NULL DEFAULT '0',
  CUST_LAST_NAME varchar(255) DEFAULT NULL,
  CUST_FIRST_NAME varchar(255) DEFAULT NULL,
  GENDER_ID int(10) DEFAULT NULL,
  CUST_BIRTHDATE datetime DEFAULT NULL,
  EMAIL varchar(255) DEFAULT NULL,
  ADDRESS varchar(255) DEFAULT NULL,
  ZIPCODE varchar(255) DEFAULT NULL,
  INCOME_ID int(10) DEFAULT NULL,
  CUST_CITY_ID int(10) DEFAULT NULL,
  AGE_YEARS double(7,2) DEFAULT NULL,
  AGERANGE_ID int(10) DEFAULT NULL,
  MARITALSTATUS_ID int(10) DEFAULT NULL,
  EDUCATION_ID int(10) DEFAULT NULL,
  HOUSINGTYPE_ID int(10) DEFAULT NULL,
  HOUSEHOLDCOUNT_ID int(10) DEFAULT NULL,
  PLAN_ID int(10) DEFAULT NULL,
  FIRST_ORDER datetime DEFAULT NULL,
  LAST_ORDER datetime DEFAULT NULL,
  TENURE double(7,2) DEFAULT NULL,
  RECENCY double(7,2) DEFAULT NULL,
  STATUS_ID int(10) DEFAULT NULL,
  PRIMARY KEY (CUSTOMER_ID)
) ENGINE=MyISAM

CREATE TABLE LU_CUST_CITY (
  CUST_CITY_ID smallint(5) NOT NULL DEFAULT '0',
  CUST_CITY_NAME varchar(50) DEFAULT NULL,
  CUST_STATE_ID smallint(5) DEFAULT NULL,
  PRIMARY KEY (CUST_CITY_ID),
  KEY ixCustCity01 (CUST_STATE_ID)
) ENGINE=MyISAM

CREATE TABLE LU_CUST_STATE (
  CUST_STATE_ID smallint(5) NOT NULL DEFAULT '0',
  CUST_STATE_NAME varchar(50) DEFAULT NULL,
  CUST_REGION_ID smallint(5) DEFAULT NULL,
  PRIMARY KEY (CUST_STATE_ID),
  KEY ixCustState01 (CUST_REGION_ID)
) ENGINE=MyISAM
CREATE TABLE LU_CUSTOMER (  CUSTOMER_ID int(10) NOT NULL DEFAULT '0',  CUST_LAST_NAME varchar(255) DEFAULT NULL,  CUST_FIRST_NAME varchar(255) DEFAULT NULL,  GENDER_ID int(10) DEFAULT NULL,  CUST_BIRTHDATE datetime DEFAULT NULL,  EMAIL varchar(255) DEFAULT NULL,  ADDRESS varchar(255) DEFAULT NULL,  ZIPCODE varchar(255) DEFAULT NULL,  INCOME_ID int(10) DEFAULT NULL,  CUST_CITY_ID smallint(5) DEFAULT NULL,  AGE_YEARS double(7,2) DEFAULT NULL,  AGERANGE_ID int(10) DEFAULT NULL,  MARITALSTATUS_ID int(10) DEFAULT NULL,  EDUCATION_ID int(10) DEFAULT NULL,  HOUSINGTYPE_ID int(10) DEFAULT NULL,  HOUSEHOLDCOUNT_ID int(10) DEFAULT NULL,  PLAN_ID int(10) DEFAULT NULL,  FIRST_ORDER datetime DEFAULT NULL,  LAST_ORDER datetime DEFAULT NULL,  TENURE double(7,2) DEFAULT NULL,  RECENCY double(7,2) DEFAULT NULL,  STATUS_ID int(10) DEFAULT NULL,  PRIMARY KEY (CUSTOMER_ID)  INDEX ixCustomer01 (CUST_CITY_ID)  ENGINE=MyISAM  DEFAULT CHARACTER SET utf8  COLLATE utf8_general_ci ) ENGINE=MyISAM

CREATE TABLE LU_CUST_CITY (  CUST_CITY_ID smallint(5) NOT NULL DEFAULT '0',  CUST_CITY_NAME varchar(50) DEFAULT NULL,  CUST_STATE_ID smallint(5) DEFAULT NULL,  PRIMARY KEY (CUST_CITY_ID)  INDEX ixCustCity01 (CUST_STATE_ID)  ENGINE=MyISAM  DEFAULT CHARACTER SET utf8  COLLATE utf8_general_ci ) ENGINE=MyISAM

CREATE TABLE LU_CUST_STATE (  CUST_STATE_ID smallint(5) NOT NULL DEFAULT '0',  CUST_STATE_NAME varchar(50) DEFAULT NULL,  CUST_REGION_ID smallint(5) DEFAULT NULL,  PRIMARY KEY (CUST_STATE_ID)  INDEX ixCustState01 (CUST_REGION_ID)  ENGINE=MyISAM  DEFAULT CHARACTER SET utf8  COLLATE utf8_general_ci ) ENGINE=MyISAM

CREATE TABLE LU_CUST_REGION (  CUST_REGION_ID smallint(5) DEFAULT NULL,  CUST_REGION_NAME varchar(50) DEFAULT NULL,  CUST_COUNTRY_ID smallint(5) DEFAULT NULL,  CUST_REGION_NAME_DE varchar(50) DEFAULT NULL,  CUST_REGION_NAME_FR varchar(50) DEFAULT NULL,  CUST_REGION_NAME_ES varchar(50) DEFAULT NULL,  CUST_REGION_NAME_IT varchar(50) DEFAULT NULL,  CUST_REGION_NAME_PO varchar(50) DEFAULT NULL,  CUST_REGION_NAME_JA varchar(50) DEFAULT NULL,  CUST_REGION_NAME_SCH varchar(50) DEFAULT NULL,  CUST_REGION_NAME_KO varchar(50) DEFAULT NULL,  PRIMARY KEY (CUST_REGION_ID)  INDEX ixCustRegion01 (CUST_COUNTRY_ID)  ENGINE=MyISAM  DEFAULT CHARACTER SET utf8  COLLATE utf8_general_ci ) ENGINE=MyISAM
CREATE TABLE LU_CUSTOMER (
  CUSTOMER_ID int(10) NOT NULL DEFAULT '0',
  CUST_LAST_NAME varchar(255) DEFAULT NULL,
  CUST_FIRST_NAME varchar(255) DEFAULT NULL,
  GENDER_ID int(10) DEFAULT NULL,
  CUST_BIRTHDATE datetime DEFAULT NULL,
  EMAIL varchar(255) DEFAULT NULL,
  ADDRESS varchar(255) DEFAULT NULL,
  ZIPCODE varchar(255) DEFAULT NULL,
  INCOME_ID int(10) DEFAULT NULL,
  CUST_CITY_ID int(10) DEFAULT NULL,
  AGE_YEARS double(7,2) DEFAULT NULL,
  AGERANGE_ID int(10) DEFAULT NULL,
  MARITALSTATUS_ID int(10) DEFAULT NULL,
  EDUCATION_ID int(10) DEFAULT NULL,
  HOUSINGTYPE_ID int(10) DEFAULT NULL,
  HOUSEHOLDCOUNT_ID int(10) DEFAULT NULL,
  PLAN_ID int(10) DEFAULT NULL,
  FIRST_ORDER datetime DEFAULT NULL,
  LAST_ORDER datetime DEFAULT NULL,
  TENURE double(7,2) DEFAULT NULL,
  RECNENCY double(7,2) DEFAULT NULL,
  STATUS_ID int(10) DEFAULT NULL,
  PRIMARY KEY (CUSTOMER_ID),
  KEY ixCustomer01 (CUST_CITY_ID)
) ENGINE=MyISAM

CREATE TABLE LU_CUST_CITY (
  CUST_CITY_ID smallint(5) NOT NULL DEFAULT '0',
  CUST_CITY_NAME varchar(50) DEFAULT NULL,
  CUST_STATE_ID smallint(5) DEFAULT NULL,
  PRIMARY KEY (CUST_CITY_ID),
  KEY ixCustCity01 (CUST_STATE_ID)
) ENGINE=MyISAM

CREATE TABLE LU_CUST_STATE (
  CUST_STATE_ID smallint(5) NOT NULL DEFAULT '0',
  CUST_STATE_NAME varchar(50) DEFAULT NULL,
  CUST_REGION_ID smallint(5) DEFAULT NULL,
  PRIMARY KEY (CUST_STATE_ID),
  KEY ixCustState01 (CUST_REGION_ID)
) ENGINE=MyISAM

CREATE TABLE LU_CUST_REGION (
  CUST_REGION_ID smallint(5) DEFAULT NULL,
  CUST_REGION_NAME varchar(50) DEFAULT NULL,
  CUST_COUNTRY_ID smallint(5) DEFAULT NULL,
  CUST_REGION_NAME_DE varchar(50) DEFAULT NULL,
  CUST_REGION_NAME_FR varchar(50) DEFAULT NULL,
  CUST_REGION_NAME_ES varchar(50) DEFAULT NULL,
  CUST_REGION_NAME_IT varchar(50) DEFAULT NULL,
  CUST_REGION_NAME_PO varchar(50) DEFAULT NULL,
  CUST_REGION_NAME_JA varchar(50) DEFAULT NULL,
  CUST_REGION_NAME_SCH varchar(50) DEFAULT NULL,
  CUST_REGION_NAME_KO varchar(50) DEFAULT NULL,
  PRIMARY KEY (CUST_REGION_ID),
  KEY ixCustRegion01 (CUST_COUNTRY_ID)
) ENGINE=MyISAM

CREATE TABLE LU_CUST_COUNTRY (
  CUST_COUNTRY_ID smallint(5) DEFAULT NULL,
  CUST_COUNTRY_NAME varchar(50) DEFAULT NULL,
  CUST_COUNTRY_NAME_DE varchar(50) DEFAULT NULL,
  CUST_COUNTRY_NAME_FR varchar(50) DEFAULT NULL,
  CUST_COUNTRY_NAME_ES varchar(50) DEFAULT NULL,
  CUST_COUNTRY_NAME_IT varchar(50) DEFAULT NULL,
  CUST_COUNTRY_NAME_PO varchar(50) DEFAULT NULL,
  CUST_COUNTRY_NAME_JA varchar(50) DEFAULT NULL,
  CUST_COUNTRY_NAME_SCH varchar(50) DEFAULT NULL,
  CUST_COUNTRY_NAME_KO varchar(50) DEFAULT NULL,
  PRIMARY KEY (COUNTRY_ID)
) ENGINE=MyISAM
select d.MONTH_ID,
od.ITEM_ID,
cs.CUST_REGION_ID,
sum((od.QTY_SOLD * (od.UNIT_PRICE - od.DISCOUNT))),
sum(od.QTY_SOLD),
MOD(ROUND(((sum((od.QTY_SOLD * 
(od.UNIT_PRICE - od.DISCOUNT)))- 0.0)*RAND()+0.0)), 2.0),
(0.95 + MOD(sum((od.QTY_SOLD * 
(od.UNIT_PRICE - od.DISCOUNT))), 19.0) / 100.0)),
(0.95 - MOD(sum((od.QTY_SOLD * 
(od.UNIT_PRICE - od.DISCOUNT))), 19.0) / 100.0))
from ORDER_DETAIL od
join LU_CUSTOMER c
on (od.CUSTOMER_ID = c.CUSTOMER_ID)
join LU_CUST_CITY cc
on (c.CUST_CITY_ID = cc.CUST_CITY_ID)
join LU_CUST_STATE cs
on (cc.CUST_STATE_ID = cs.CUST_STATE_ID)
join LU_DAY d
on (od.ORDER_DATE = d.DAY_DATE)
where d.QUARTER_ID = 20094
group by d.MONTH_ID,
od.ITEM_ID,
cs.CUST_REGION_ID
select d.MONTH_ID, 
   od.ITEM_ID, 
   c.CUST_REGION_ID, 
   sum((od.QTY_SOLD * (od.UNIT_PRICE - od.DISCOUNT))), 
   sum(od.QTY_SOLD), 
   MOD(ROUND((sum((od.QTY_SOLD * 
   (od.UNIT_PRICE - od.DISCOUNT))) - 0.0)*RAND()+0.0), 2.0), 
   (0.95 + MOD(sum((od.QTY_SOLD * 
   (od.UNIT_PRICE - od.DISCOUNT))), 19.0) / 100.0)), 
   (0.95 - MOD(sum((od.QTY_SOLD * 
   (od.UNIT_PRICE - od.DISCOUNT))), 19.0) / 100.0))
from ORDER_DETAIL od 
join LU_CUSTOMER c 
on (od.CUSTOMER_ID = c.CUSTOMER_ID) 
join LU_DAY d 
on (od.ORDER_DATE = d.DAY_DATE) 
where d.QUARTER_ID = 20094 
group by d.MONTH_ID, 
   od.ITEM_ID, 
   c.CUST_REGION_ID
Avoid Complex Queries

```
explain
select pa11.BRAND_ID,
a13.BRAND_DESC,
pa11.CATEGORY_ID,
a14.CATEGORY_DESC,
pa11.CUST_REGION_ID,
a15.CUST_REGION_NAME,
pa11.Revenue,
pa11.WJXBFS1,
pa12.WJXBFS1
from  (  
    select a16.CUST_REGION_ID,
a13.CATEGORY_ID,
a12.BRAND_ID,
    sum((a11.QTY_SOLD * (a11.UNIT_PRICE - a11.DISCOUNT))) Revenue,
    sum((a11.QTY_SOLD * ((a11.UNIT_PRICE - a11.DISCOUNT) - a11.UNIT_COST))) WJXBFS1
    from ORDER_DETAIL a11
    join LU_ITEM a12 on (a11.ITEM_ID = a12.ITEM_ID)
    join LU_SUBCATEG a13 on (a12.SUBCAT_ID = a13.SUBCAT_ID)
    join LU_CUSTOMER a14 on (a11.CUSTOMER_ID = a14.CUSTOMER_ID)
    join LU_CUST_CITY a15 on (a14.CUST_CITY_ID = a15.CUST_CITY_ID)
    join LU_CUST_STATE a16 on (a15.CUST_STATE_ID = a16.CUST_STATE_ID)
    join LU_DAY a17 on (a11.ORDER_DATE = a17.DAY_DATE)
    where a17.YEAR_ID in (2009)
    group by a16.CUST_REGION_ID,
a13.CATEGORY_ID,
a12.BRAND_ID
) pa11
join  (  
    select a18.CUST_REGION_ID,
a15.CATEGORY_ID,
a14.BRAND_ID,
    sum((a11.QTY_SOLD * (a11.UNIT_PRICE - a11.DISCOUNT)))  WJXBFS1
    from  ORDER_DETAIL a11
    join  LU_DAY a12 on (a11.ORDER_DATE = a12.DAY_DATE)
    join  LU_YEAR a13 on (a12.YEAR_ID = a13.PREV_YEAR_ID)
    join  LU_ITEM a14 on (a11.ITEM_ID = a14.ITEM_ID)
    join  LU_SUBCATEG a15 on (a14.SUBCAT_ID = a15.SUBCAT_ID)
    join  LU_CUSTOMER a16 on (a11.CUSTOMER_ID = a16.CUSTOMER_ID)
    join  LU_CUST_CITY a17 on (a16.CUST_CITY_ID = a17.CUST_CITY_ID)
    join  LU_CUST_STATE a18 on (a17.CUST_STATE_ID = a18.CUST_STATE_ID)
    where  a13.YEAR_ID in (2009)
    group by a18.CUST_REGION_ID,
a15.CATEGORY_ID,
a14.BRAND_ID
) pa12
    on  (pa11.BRAND_ID = pa12.BRAND_ID and
    pa11.CATEGORY_ID = pa12.CATEGORY_ID and
    pa11.CUST_REGION_ID = pa12.CUST_REGION_ID)
join  LU_BRAND a13 on (pa11.BRAND_ID = a13.BRAND_ID)
join  LU_CATEGORY a14 on (pa11.CATEGORY_ID = a14.CATEGORY_ID)
join  LU_CUST_REGION a15 on (pa11.CUST_REGION_ID = a15.CUST_REGION_ID)
```
Avoid Complex Queries

```
explain
select pa11.BRAND_ID,
a13.BRAND_DESC,
pa11.CATEGORY_ID,
a14.CATEGORY_DESC,
pa11.CUST_REGION_ID,
a15.CUST_REGION_NAME,
pa11.Revenue,
pa11.WJXBFS1,
pa12.WJXBFS1
from  (select a16.CUST_REGION_ID,
a13.CATEGORY_ID,
a12.BRAND_ID,
sum((a11.QTY_SOLD * (a11.UNIT_PRICE - a11.DISCOUNT))) Revenue,
sum((a11.QTY_SOLD * ((a11.UNIT_PRICE - a11.DISCOUNT) - a11.UNIT_COST))) WJXBFS1
    from ORDER_DETAIL  a11
    join LU_ITEM       a12 on (a11.ITEM_ID = a12.ITEM_ID)
    join LU_SUBCATEG   a13 on (a12.SUBCAT_ID = a13.SUBCAT_ID)
    join LU_CUSTOMER   a14 on (a11.CUSTOMER_ID = a14.CUSTOMER_ID)
    join LU_CUST_CITY  a15 on (a14.CUST_CITY_ID = a15.CUST_CITY_ID)
    join LU_CUST_STATE a16 on (a15.CUST_STATE_ID = a16.CUST_STATE_ID)
    join LU_DAY        a17 on (a11.ORDER_DATE = a17.DAY_DATE)
    where a17.YEAR_ID in (2009)
group by a16.CUST_REGION_ID,
a13.CATEGORY_ID,
a12.BRAND_ID)
pa11
join  (select a18.CUST_REGION_ID,
a15.CATEGORY_ID,
a14.BRAND_ID,
sum((a11.QTY_SOLD * (a11.UNIT_PRICE - a11.DISCOUNT)))  WJXBFS1
    from  ORDER_DETAIL     a11
    join  LU_DAY           a12 on (a11.ORDER_DATE = a12.DAY_DATE)
    join  LU_YEAR          a13 on (a12.YEAR_ID = a13.PREV_YEAR_ID)
    join  LU_ITEM          a14 on (a11.ITEM_ID = a14.ITEM_ID)
    join  LU_SUBCATEG      a15 on (a14.SUBCAT_ID = a15.SUBCAT_ID)
    join  LU_CUSTOMER      a16 on (a11.CUSTOMER_ID = a16.CUSTOMER_ID)
    join  LU_CUST_CITY     a17 on (a16.CUST_CITY_ID = a17.CUST_CITY_ID)
    join  LU_CUST_STATE    a18 on (a17.CUST_STATE_ID = a18.CUST_STATE_ID)
    where  a13.YEAR_ID in (2009)
group by a18.CUST_REGION_ID,
a15.CATEGORY_ID,
a14.BRAND_ID)
pa12
on  (pa11.BRAND_ID = pa12.BRAND_ID and
    pa11.CATEGORY_ID = pa12.CATEGORY_ID and
    pa11.CUST_REGION_ID = pa12.CUST_REGION_ID)
join  LU_BRAND       a13 on (pa11.BRAND_ID = a13.BRAND_ID)
join  LU_CATEGORY    a14 on (pa11.CATEGORY_ID = a14.CATEGORY_ID)
join  LU_CUST_REGION a15 on (pa11.CUST_REGION_ID = a15.CUST_REGION_ID)
```
Avoid Complex Queries

Three separate EXPLAINed queries take 1.7 ms + 1.5 ms + 0.6 ms = 3.8 ms

- Also, simple queries:
  - Are more likely to be found in the query cache
  - Can be implemented in as key/value with memcached
  - Can use Key Buffer and Buffer Pool more efficiently
  - Are Sharding and Replication friendly

Example SQL:

```sql
explain
select pa11.BRAND_ID,
a13.BRAND_DESC,
pa11.CATEGORY_ID,
a14.CATEGORY_DESC,
pa11.CUST_REGION_ID,
a15.CUST_REGION_NAME,
pa11.Revenue,
pa11.WJXBFS1,
pa12.WJXBFS1
from  
(select a16.CUST_REGION_ID,
a13.CATEGORY_ID,
a12.BRAND_ID,
sum((a11.QTY_SOLD * (a11.UNIT_PRICE - a11.DISCOUNT))) Revenue,
sum((a11.QTY_SOLD * ((a11.UNIT_PRICE - a11.DISCOUNT) - a11.UNIT_COST))) WJXBFS1
from ORDER_DETAIL a11
join LU_ITEM a12 on (a11.ITEM_ID = a12.ITEM_ID)
join LU_SUBCATEG a13 on (a12.SUBCAT_ID = a13.SUBCAT_ID)
jion LU_CUSTOMER a14 on (a11.CUSTOMER_ID = a14.CUSTOMER_ID)
jion LU_CUST_CITY a15 on (a14.CUST_CITY_ID = a15.CUST_CITY_ID)
jion LU_CUST_STATE a16 on (a15.CUST_STATE_ID = a16.CUST_STATE_ID)
jion LU_DAY a17 on (a11.ORDER_DATE = a17.DAY_DATE)
where a17.YEAR_ID in (2009)
group by a16.CUST_REGION_ID,
a13.CATEGORY_ID,
a12.BRAND_ID
) pa11
join  
(select a18.CUST_REGION_ID,
a15.CATEGORY_ID,
a14.BRAND_ID,
sum((a11.QTY_SOLD * (a11.UNIT_PRICE - a11.DISCOUNT))) WJXBFS1
from ORDER_DETAIL a11
join LU_DAY a12 on (a11.ORDER_DATE = a12.DAY_DATE)
join LU_YEAR a13 on (a12.YEAR_ID = a13.PREV_YEAR_ID)
join LU_ITEM a14 on (a11.ITEM_ID = a14.ITEM_ID)
join LU_SUBCATEG a15 on (a14.SUBCAT_ID = a15.SUBCAT_ID)
jion LU_CUSTOMER a16 on (a11.CUSTOMER_ID = a16.CUSTOMER_ID)
jion LU_CUST_CITY a17 on (a16.CUST_CITY_ID = a17.CUST_CITY_ID)
jion LU_CUST_STATE a18 on (a17.CUST_STATE_ID = a18.CUST_STATE_ID)
where a13.YEAR_ID in (2009)
group by a18.CUST_REGION_ID,
a15.CATEGORY_ID,
a14.BRAND_ID
) pa12
  on (pa11.BRAND_ID = pa12.BRAND_ID and
  pa11.CATEGORY_ID = pa12.CATEGORY_ID and
  pa11.CUST_REGION_ID = pa12.CUST_REGION_ID)
join LU_BRAND a13 on (pa11.BRAND_ID = a13.BRAND_ID)
join LU_CATEGORY a14 on (pa11.CATEGORY_ID = a14.CATEGORY_ID)
join LU_CUST_REGION a15 on (pa11.CUST_REGION_ID = a15.CUST_REGION_ID)
```

20 rows in set, (27.63 sec)
Multi-Pass SQL

- Less joins: better query optimization, faster queries
- More efficient use of caching
  - Query cache and Memcached
- Can be spread on shards and replicas
- Bypass some limitations specifically with subqueries

- Client interface and coding is more complicated
- Use of temporary structures that must be cleaned/removed
Database Partitioning

• Partitioning in 5.1
  – Up to 1024 partitions
  – Partitions by range, list, hash, key > based on integer values
  – Sub-partitions by hash, key
  – Partition pruning
  – All partitions have the same storage engine

• Partitioning in 5.5
  – Partitioning by non-integer, list of values and expressions is accepted

Vertical Partitioning

Horizontal Partitioning
Database Partitioning

- Provides a stable and fast INSERT mechanism specifically when partitioning is time-based.
- Database administration is significantly improved.
  - Partitions can be instantly created and dropped
- With partition pruning, SELECT performance is improved

- A high number of partitions may affect performance with sparse INSERTS
  - LOCK TABLE may alleviate the impact on performance for a large number of INSERTS
- No foreign key spatial and fulltext index support
- No parallel SELECTs on multiple partitions.
Application Partitioning

Partition Mapping Table

Partition Key | Partition Base Table
--------------|---------------------

getPTable

Partition Tables

<table>
<thead>
<tr>
<th>PK</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
</tr>
<tr>
<td>PK</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
</tr>
<tr>
<td>PK</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
<td>Met</td>
</tr>
</tbody>
</table>
Application Partitioning

Partition Mapping Table

<table>
<thead>
<tr>
<th>Partition Key</th>
<th>Partition Base Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

getPTable

CALL getPTable('TABLE', 'P1')

Partition Tables

<table>
<thead>
<tr>
<th>PK</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Application Partitioning

Partition Mapping Table

<table>
<thead>
<tr>
<th>Partition Key</th>
<th>Partition Base Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

getPTable

CALL getPTable('TABLE', 'P1')

@pTable='TABLE_NAME'

Partition Tables
Application Partitioning

Partition Mapping Table

<table>
<thead>
<tr>
<th>Partition Key</th>
<th>Partition Base Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

getPTable

CALL getPTable('TABLE', 'P1')

@pTable='TABLE_NAME'

SET @myStmt=CONCAT('SELECT...FROM ', @pTable, ' WHERE...');
PREPARE myPrep FROM myStmt;
EXECUTE myPrep;

Partition Tables

<table>
<thead>
<tr>
<th>PK</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Wednesday, 14 April 2010
Application Partitioning

Partition Mapping Table

<table>
<thead>
<tr>
<th>Partition Key</th>
<th>Partition Base Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partition Tables

<table>
<thead>
<tr>
<th>PK</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

getPTable

CALL getPTable('TABLE', 'P1')

@pTable='TABLE_NAME'

SET @myStmt=CONCAT('SELECT...FROM ', @pTable, ' WHERE...');
PREPARE myPrep FROM myStmt;
EXECUTE myPrep;

Result Set

<table>
<thead>
<tr>
<th>Col 1</th>
<th>Col 2</th>
<th>Col 3</th>
<th>Col 4</th>
<th>...</th>
<th>Col N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Application Partitioning

PMT_SALES

<table>
<thead>
<tr>
<th>QUARTER_ID</th>
<th>PBT_SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>201001</td>
<td>SALES_201001</td>
</tr>
<tr>
<td>201002</td>
<td>SALES_201002</td>
</tr>
<tr>
<td>201003</td>
<td>SALES_201003</td>
</tr>
<tr>
<td>201004</td>
<td>SALES_201004</td>
</tr>
</tbody>
</table>

getPTable

SALES Tables

<table>
<thead>
<tr>
<th>PK</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Application Partitioning

**PMT_SALES**

<table>
<thead>
<tr>
<th>QUARTER_ID</th>
<th>PBT_SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>201001</td>
<td>SALES_201001</td>
</tr>
<tr>
<td>201002</td>
<td>SALES_201002</td>
</tr>
<tr>
<td>201003</td>
<td>SALES_201003</td>
</tr>
<tr>
<td>201004</td>
<td>SALES_201004</td>
</tr>
</tbody>
</table>

**CALL getPTable('SALES', '201001')**

**SALES Tables**

<table>
<thead>
<tr>
<th>PK</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
<th>Met</th>
</tr>
</thead>
</table>

**Wednesday, 14 April 2010**
**Application Partitioning**

**PMT_SALES**

<table>
<thead>
<tr>
<th>QUARTER_ID</th>
<th>PBT_SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>201001</td>
<td>SALES_201001</td>
</tr>
<tr>
<td>201002</td>
<td>SALES_201002</td>
</tr>
<tr>
<td>201003</td>
<td>SALES_201003</td>
</tr>
<tr>
<td>201004</td>
<td>SALES_201004</td>
</tr>
</tbody>
</table>

**CALL getPTable('SALES', '201001')**

@pTable='SALES_201001'

**SALES Tables**
Application Partitioning

**PMT_SALES**

<table>
<thead>
<tr>
<th>QUARTER_ID</th>
<th>PBT_SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>201001</td>
<td>SALES_201001</td>
</tr>
<tr>
<td>201002</td>
<td>SALES_201002</td>
</tr>
<tr>
<td>201003</td>
<td>SALES_201003</td>
</tr>
<tr>
<td>201004</td>
<td>SALES_201004</td>
</tr>
</tbody>
</table>

**getPTable**

CALL getPTable('SALES', '201001')

@pTable='SALES_201001'

SET @myStmt=CONCAT('SELECT...FROM ', @pTable, ' WHERE...');
PREPARE myPrep FROM myStmt;
EXECUTE myPrep;

**SALES Tables**

PK Met Met Met Met Met Met
Application Partitioning

PMT_SALES

<table>
<thead>
<tr>
<th>QUARTER_ID</th>
<th>PBT_SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>201001</td>
<td>SALES_201001</td>
</tr>
<tr>
<td>201002</td>
<td>SALES_201002</td>
</tr>
<tr>
<td>201003</td>
<td>SALES_201003</td>
</tr>
<tr>
<td>201004</td>
<td>SALES_201004</td>
</tr>
</tbody>
</table>

getPTable

CALL getPTable('SALES', '201001')

@pTable='SALES_201001'

SET @myStmt=CONCAT('SELECT...FROM ', @pTable, ' WHERE...');
PREPARE myPrep FROM myStmt;
EXECUTE myPrep;

Result Set

<table>
<thead>
<tr>
<th>Col 1</th>
<th>Col 2</th>
<th>Col 3</th>
<th>Col 4</th>
<th>...</th>
<th>Col N</th>
</tr>
</thead>
</table>
Vertical Partitioning

- “Handle with care”
- Useful when:
  - 90% of the queries...
    - ...require a join only with the primary fact
    - ...the result set is relatively small and it can be stored in a temp table, then it can be joined with the secondary fact
- Advantages:
  - Caching optimization
    - Queries on the primary fact are in cache
    - Queries on the secondary fact are not in cache
Vertical Partitioning

- "Handle with care"

- Useful when:
  - 90% of the queries...
    - ...require a join only with the primary fact
    - ...the result set is relatively small and it can be stored in a temp table, then it can be joined with the secondary fact

- Advantages:
  - Caching optimization
    - Queries on the primary fact are in cache
    - Queries on the secondary fact are not in cache
Query Cache

- Good for lookup queries and small reports
- Slow moving query cache can be warmed up
- Examples:
  - Cached reports of the top $X$ daily queries
  - With multi-pass SQL, some result sets may be immediately available
Memcached

- Great for all the queries
- It can be integrated with non-standard BI products
- But...
  - It must be manually invalidated every time that some data is loaded
  - It can be difficult to integrate with standard BI tools

1. Query requested and found in Memcached
2. Query requested but not found in Memcached
   a. Data is retrieved from the DW Server
   b. The query is stored in Memcached
3. The ETL process invalidates Memcached
Memcached

- Great for all the queries
- It can be integrated with non-standard BI products
- But...
  - It must be manually invalidated every time that some data is loaded
  - It can be difficult to integrate with standard BI tools

1. Query requested and found in Memcached

```sql
md5('select d.MONTH_ID, od.ITEM_ID, c.CUST_REGION_ID,
    sum((od.QTY_SOLD * (od.UNIT_PRICE - od.DISCOUNT))),
    sum(od.QTY_SOLD),
    MOD(ROUND(((sum((od.QTY_SOLD * (od.UNIT_PRICE - od.DISCOUNT)))- 0.0)*RAND()+0.0)), 2.0),
    (0.95 + (MOD(sum((od.QTY_SOLD * (od.UNIT_PRICE - od.DISCOUNT))), 19.0) / 100.0)),
    (0.95 - (MOD(sum((od.QTY_SOLD * (od.UNIT_PRICE - od.DISCOUNT))), 19.0) / 100.0))
from ORDERDETAIL od
join LU_CUSTOMER c
    on(od.CUSTOMER_ID = c.CUSTOMER_ID)
join LU_DAY d
    on(od.ORDER_DATE = d.DAY_DATE)
where d.QUARTER_ID = 20094
    group by d.MONTH_ID,
    od.ITEM_ID,
    c.CUST_REGION_ID');
```
Query Cache or Memcached?

**Query Cache**
- MySQL only
- Limited to 1 mysqld
  - Data cached may be replicated on multiple mysqld servers
- Transparent to the applications
  - Can be switched on/off/on demand
- Automatic invalidation

**Memcached**
- Any application
- Distributed and available for multiple mysqld servers
- Application specific
- Must be manually invalidated
Replication

- Helpful for near real time Data Warehousing and for DW Servers that must be always ON
- Basic Sharding through the `--replicate-do-table/db` and `--replicated-ignore-table/db` parameters
- But... Replication may not be the quickest way to perform a bulk update
- Scenario 1 - Distributed load balancing on all the slaves
Replication

- Scenario 2 - Continuous analysis, update and validation on a DW or DM
- Scenario 3 - Rotating, delayed snapshots

Wednesday, 14 April 2010
Sharding

- The sharding logic must be implemented at application level
  - The same approach of the Partition Mapping can be used
  - Sharding can be difficult to be used with some BI tools
- Lookup tables are replicated
- Fact tables are partitioned (in shards) on the servers
  - The sharding strategy is the key factor to improve performance
  - The storage is optimized
- Shards can use replication

Read

Write

Dimensions Master

Fact 00-03
Fact 04-07
Fact 08-11
Fact 12-15
Fact 16-19
Fact 20-23

BI/Report Servers

Shards

Wednesday, 14 April 2010
Sharding

- The sharding logic must be implemented at application level
  - The same approach of the Partition Mapping can be used
  - Sharding can be difficult to be used with some BI tools
- Lookup tables are replicated
- Fact tables are partitioned (in shards) on the servers
  - The sharding strategy is the key factor to improve performance
  - The storage is optimized
- Shards can use replication
Active-Active Servers & Shared Storage

- ETL
  - Parallel loading - by table

- Data Analysis and Reporting
  - Queries can be load balanced on all servers

- Configuration
  - OCFS2 or GFS/GFS2
  - MyISAM tables
  - mysqld with
    --external-locking
    --query-cache-size=0
Benchmarks, Really Really?
OLAP Benchmarks

• OLAP Benchmarks are designed to measure the performance of the HW and the DB SW on basic operations, but they do not reflect the complexity of an advanced BI and Analytical system.

• For example...
  – TCP-H
    • The most important DW benchmark based on 7 tables, with a OLTP design, not really multidimensional
  – SSC
    • Star Schema benchmark based on 5 tables, 4 dimensions, 24 attributes and 10 facts
    – 13 queries
What is your real need?

“Find the aggregate profit by year, nation and product category in 1997 and 1998 for manufacturers X and Y”

```sql
SELECT d_year, s_nation, p_category,
       sum(lo_revenue - lo_supplycost) as profit
  FROM date, customer, supplier, part, lineorder
WHERE lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_partkey = p_partkey
  AND lo_orderdate = d_datekey
  AND c_region = 'AMERICA'
  AND s_region = 'AMERICA'
  AND (d_year = 1997 OR d_year = 1998)
  AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
GROUP BY d_year, s_nation, p_category
ORDER BY d_year, s_nation, p_category
```
What is your real need?

"Find the aggregate profit by year, nation and product category in 1997 and 1998 for manufacturers X and Y"

```
SELECT d_year, s_nation, p_category,
       sum(lo_revenue - lo_supplycost) as profit
  FROM date, customer, supplier, part, lineorder
 WHERE lo_custkey = c_custkey
   AND lo_suppkey = s_suppkey
   AND lo_partkey = p_partkey
   AND lo_orderdate = d_datekey
   AND c_region = 'AMERICA'
   AND s_region = 'AMERICA'
   AND (d_year = 1997 OR d_year = 1998)
   AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
 GROUP BY d_year, s_nation, p_category
 ORDER BY d_year, s_nation, p_category
```

"Analyze revenue, growth, units sold, revenue per unit and revenue forecast by supplier, month, region and item"

```
CREATE TEMPORARY TABLE ZZTJT56LYTZMD000(
   MONTH_ID        INTEGER(4),
   ITEM_ID         SMALLINT(2),
   CUST_REGION_ID SMALLINT(2),
   Revenue         DOUBLE,
   WJXBFS1         DOUBLE,
   WJXBFS2         DOUBLE,
   WJXBFS3         DOUBLE,
   WJXBFS4         DOUBLE)
```
"What is your real need?"

"Find the aggregate profit by year, nation and product category in 1997 and 1998 for manufacturers X and Y"

```
SELECT d_year, s_nation, p_category,
       sum(lo_revenue - lo_supplycost) as profit
  FROM date, customer, supplier, part, lineorder
WHERE lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_partkey = p_partkey
  AND lo_orderdate = d_datekey
  AND c_region = 'AMERICA'
  AND s_region = 'AMERICA'
  AND (d_year = 1997 OR d_year = 1998)
  AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
GROUP BY d_year, s_nation, p_category
ORDER BY d_year, s_nation, p_category
```

"Analyze revenue, growth, units sold, revenue per unit and revenue forecast for by supplier, month, region and item"

```
CREATE TEMPORARY TABLE ZZTJT56LYTZMD000(
   MONTH_ID        INTEGER (4) ,  
   ITEM_ID         SMALLINT (2) ,  
   CUST_REGION_ID  SMALLINT (2) ,  
   Revenue         DOUBLE ,  
   W J X B F S 1         DOUBLE ,  
   W J X B F S 2         DOUBLE ,  
   W J X B F S 3         DOUBLE ,  
   W J X B F S 4         DOUBLE 
)

insert into ZZTJT56LYTZMD000
select     a15.MONTH_ID        MONTH_ID,
           a11.ITEM_ID         ITEM_ID,
        ... 4 . CUST_REGION_ID
create index ZZTJT56LYTZMD000_i on ZZTJT56LYTZMD000 (MONTH_ID, ITEM_ID, CUST_REGION_ID)
```
What is your real need?

“Find the aggregate profit by year, nation and product category in 1997 and 1998 for manufacturers X and Y”

“Analyze revenue, growth, units sold, revenue per unit and revenue forecast for by supplier, month, region and item”

```
SELECT d_year, s_nation, p_category, 
    sum(lo_revenue - lo_supplycost) as profit
FROM date, customer, supplier, part, lineorder
WHERE lo_custkey = c_custkey 
    AND lo_suppkey = s_suppkey 
    AND lo_partkey = p_partkey 
    AND lo_orderdate = d_datekey 
    AND c_region = 'AMERICA' 
    AND s_region = 'AMERICA' 
    AND (d_year = 1997 OR d_year = 1998) 
    AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
GROUP BY d_year, s_nation, p_category
ORDER BY d_year, s_nation, p_category
```
What is your real need?

"Find the aggregate profit by year, nation and product category in 1997 and 1998 for manufacturers X and Y"

"Analyze revenue, growth, units sold, revenue per unit and revenue forecast for by supplier, month, region and item"
What is your real need?

“Find the aggregate profit by year, nation and product category in 1997 and 1998 for manufacturers X and Y”

“Analyze revenue, growth, units sold, revenue per unit and revenue forecast for by supplier, month, region and item”

```sql
SELECT d_year, s_nation, p_category,
       sum(lo_revenue - lo_supplycost) as profit
  FROM date, customer, supplier, ...
WHERE p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2'
GROUP BY d_year, s_nation, p_category
 ORDER BY d_year, s_nation, p_category
```

```sql
CREATE TEMPORARY TABLE ZZTJT56LYTZMD000(
  MONTH_ID INTEGER(4),
  ITEM_ID SMALLINT(2),
  CUST_REGION_ID SMALLINT(2),
  WJXBFS1 DOUBLE
)
select a15.MONTH_ID, a11.ITEM_ID, a14.CUST_REGION_ID
from ZZTJT56LYTZMD000 pa11
join ZZT9G61ZYFMD001 pa12
join LU_ITEM a13 on (pa11.ITEM_ID = a13.ITEM_ID)
join LU_SUBCATEG a14 on (a13.SUBCAT_ID = a14.SUBCAT_ID)
join LU_CUST_REGION a15 on (pa11.CUST_REGION_ID = a15.CUST_REGION_ID)
join LU_MONTH a16 on (pa11.MONTH_ID = a16.MONTH_ID)
join LU_SUPPLIER a17 on (a13.SUPPLIER_ID = a17.SUPPLIER_ID)
join LU_CATEGORY a18 on (a14.CATEGORY_ID = a18.CATEGORY_ID)
```
The Real Needs

- Standard analysis
  - Cx/Lx and xTD analysis
  - Min/Max/Avg value by attribute

- Contributions
  - Total Value, % or value by item or category

- Market Basket Analysis
  - Affinity products

- Inventory
  - Begin on hand/End on hand

- Time-based analysis
  - as-is/as-is, as-is/as-was
Benchmarks from the MySQL Performance Blog

• http://www.mysqlperformanceblog.com/2010/01/07/star-schema-bechmark-infobright-infinidb-and-luciddb

• http://www.percona.com/docs/wiki/benchmark:ssb:start

• On going tests:
  – X4450 4x Xeon 6 core (24 cores) 32GB RAM
    • RH EL 5 Upd 4
  – ST2450 Array 12x300GB SAS 15krpm drives in FB, R10 (1.8TB)
    • ext3
Next Steps

• More generic info here:
  – izoratti.blogspot.com

• Benchmark results and more info here:
  – mysql4all.wordpress.com
The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.