Testing SQL and NoSQL Databases

Javier Tuya, C. de la Riva, M.J. Suárez-Cabal and R. Blanco

University of Oviedo
Software Engineering Research Group
http://giis.uniovi.es/

Workshop on software testing, metrics, data mining and decision making. Iceberg project
Alcalá de Henares, March 25th 2015
AGENDA

1. Why measure test coverage for database applications?
2. A test coverage criterion or SQL
   - Application: manual creation of test databases
   - Application: automatic reduction of test databases
3. Different issues for NoSQL databases
   - Testing data storage
   - Testing data processing

Acknowledgements (2):
Test4SOA - Techniques for Functional Testing in Service Oriented Architectures (TIN2007-67843-C06-01)
Test4DBS - Test methods for applications with databases and services (TIN2010-20057-C03-01)

Acknowledgements (3):
PERTEST - Testing of data persistence and user perspective for new paradigms (TIN2013-46928-C3-1-R)
Do you know your test coverage?

A very simple Java program …

Salary is calculated as base salary plus the average bonus generated by projects in which an employee has participated.

```java
public float CalculateSalary(String idEmployee, Connection conn) throws Exception{
    ResultSet rsTrans;
    Statement stmt;
    String query;
    float basesalary;
    float averagebonus;

    stmt = conn.createStatement();
    query = "SELECT basesalary, avg(bonus) " +
             "FROM Employee E INNER JOIN Assignment A ON E.idEmp = A.idEmp " +
             "INNER JOIN Project P ON A.idProj = P.idProj " +
             "WHERE E.idEmp = " + idEmployee + " AND P.bonus > 0 " +
             "GROUP BY E.idEmp";
    rsTrans = stmt.executeQuery(query);
    rsTrans.next();
    basesalary = rsTrans.getFloat(1);
    averagebonus = rsTrans.getFloat(2);
    return basesalary + averagebonus;
}
```

Executing this function executes all branches 100% coverage. Really?
Do you know your test coverage?

... and your SQL test coverage?

```java
public float CalculateSalary(String idEmployee, Connection conn) throws Exception{
    ResultSet rsTrans;
    Statement stmt;
    String query;
    float basesalary;
    float averagebonus;

    stmt = conn.createStatement();

    query = "SELECT basesalary, avg(bonus) " +
            "FROM Employee E INNER JOIN Assignment A ON E.idEmp = A.idEmp " +
            "INNER JOIN Project P ON A.idProj = P.idProj " +
            "WHERE E.idEmp = " + idEmployee + " AND P.bonus > 0 " +
            "GROUP BY E.idEmp";

    basesalary = rsTrans.getFloat(1);
    averagebonus = rsTrans.getFloat(2);

    return basesalary + averagebonus;
}
```
... and your SQL test coverage?

```sql
query = "SELECT basesalary, avg(bonus) " + 
    "FROM Employee E INNER JOIN Assignment A ON E.idEmp = A.idEmp " + 
    "INNER JOIN Project P ON A.idProj = P.idProj " + 
    "WHERE E.idEmp = " + idEmployee + " AND P.bonus > 0 " + 
    "GROUP BY E.idEmp";
```

What if...

... All assigned projects for a given employee have bonus > 0?

... Some assigned Project has bonus = 0?

... All assigned projects have bonus = 0?

... Employee is not assigned to any project?
... and your SQL test coverage?

```
query = "SELECT basesalary, avg(bonus) " +
    "FROM Employee E INNER JOIN Assignment A ON E.idEmp = A.idEmp " +
    "INNER JOIN Project P ON A.idProj = P.idProj " +
    "WHERE E.idEmp = " + idEmployee + " AND P.bonus > 0 " +
    "GROUP BY E.idEmp";
```

What if…

... All assigned projects for a given employee have bonus > 0?

The average bonus returned by the SQL statement is **correct**
... and your SQL test coverage?

query = "SELECT basesalary, avg(bonus) " +
'"FROM Employee E INNER JOIN Assignment A ON E.idEmp = A.idEmp " +
"INNER JOIN Project P ON A.idProj = P.idProj " +
"WHERE E.idEmp = " + idEmployee + " AND P.bonus > 0 " +
"GROUP BY E.idEmp";

What if...

... Some assigned project has bonus = 0?

The average bonus returned by the SQL statement is **NOT** correct
... and your SQL test coverage?

```
query = "SELECT basesalary, avg(bonus) " +
"FROM Employee E INNER JOIN Assignment A ON E.idEmp = A.idEmp " +
"INNER JOIN Project P ON A.idProj = P.idProj " +
"WHERE E.idEmp = " + idEmployee + " AND P.bonus > 0 " +
"GROUP BY E.idEmp";
```

What if…

… All assigned projects have bonus = 0?

… Employee is not assigned to any project?

SQL statement does not return any row. A runtime exception raises (0 rows)
... and your SQL test coverage?

```
query = "SELECT basesalary, avg(bonus) " +
    "FROM Employee E INNER JOIN Assignment A ON E.idEmp = A.idEmp " +
    "INNER JOIN Project P ON A.idProj = P.idProj " +
    "WHERE E.idEmp = " + idEmployee + " AND P.bonus > 0 " +
    "GROUP BY E.idEmp";
```

We should cover a number of test situations derived from the SQL statement.

According to ISO/IEC/IEEE 29119-4:
- Test Condition: the SQL statement
- Test Coverage Items: each test situation
What about this query?

```
SELECT o.AD_Client_ID, o.AD_Org_ID, o.C_BPartner_ID, o.C_Order_ID,
o.DocumentNo, o.DateOrdered, o.C_DocType_ID,
SUM((l.QtyOrdered-l.QtyInvoiced)*l.PriceActual) AS TotalLines
FROM C_Order o
  INNER JOIN C_OrderLine l ON (o.C_Order_ID=l.C_Order_ID)
  INNER JOIN C_BPartner bp ON (o.C_BPartner_ID=bp.C_BPartner_ID)
LEFT OUTER JOIN C_InvoiceSchedule si ON (bp.C_InvoiceSchedule_ID=si.C_InvoiceSchedule_ID)
WHERE o.DocStatus IN ('CO','CL','IP') -- Standard Orders are IP
  -- not Offers and open Walkin-Receipts
AND o.C_DocType_ID IN (SELECT C_DocType_ID FROM C_DocType
  WHERE DocBaseType='SOO' AND DocSubTypeSO NOT IN ('ON','OB','WR'))
  -- we need to invoice
AND l.QtyOrdered <> l.QtyInvoiced
  -- Immediate
AND o.InvoiceRule='I'
-- Order complete ** not supported **
  OR o.InvoiceRule='0'
-- Delivery
  OR (o.InvoiceRule='D' AND l.QtyInvoiced>l.QtyDelivered)
-- Order Schedule, but none defined on Business Partner level
  OR (o.InvoiceRule='S' AND bp.C_InvoiceSchedule_ID IS NULL)
-- Schedule defined at BP
  OR (o.InvoiceRule='S' AND bp.C_InvoiceSchedule_ID IS NOT NULL AND
    ( -- Daily or none
      (si.InvoiceFrequency IS NULL OR si.InvoiceFrequency='D')
      -- Weekly
    OR (si.InvoiceFrequency='W')
    -- Bi-Monthly
    OR (si.InvoiceFrequency='T'
      AND (TRUNC(o.DateOrdered) <= firstOf(getdate(),'MM')+si.InvoiceDayCutoff-1
        AND TRUNC(getdate()) >= firstOf(o.DateOrdered,'MM')+si.InvoiceDay-1)
      OR (TRUNC(o.DateOrdered) <= firstOf(getdate(),'MM')+si.InvoiceDayCutoff+14
        AND TRUNC(getdate()) >= firstOf(o.DateOrdered,'MM')+si.InvoiceDay+14)
    -- Monthly
    OR (si.InvoiceFrequency='M'
      AND TRUNC(o.DateOrdered) <= firstOf(getdate(),'MM')+si.InvoiceDayCutoff-1 -- after cutoff
      AND TRUNC(getdate()) >= firstOf(o.DateOrdered,'MM')+si.InvoiceDay-1) -- after invoice day
    )
  )
GROUP BY o.AD_Client_ID, o.AD_Org_ID, o.C_BPartner_ID, o.C_Order_ID, o.DocumentNo, o.DateOrdered, o.C_DocType_ID
```

From Compiere ERP V2.53b
View c_invoice_candidate_v

J. Tuya (2015) Testing SQL and NoSQL Databases 10
Context of the problem: Database application testing

Test Criterion

Test Case Design

Expected Outputs

Inputs

System Under Test

Actual Outputs

Result Comparison

Pass/Fail
Context of the problem: Database application testing

- Database is both input & output
- Complex structure
- Large: Difficult to check actual vs. expected output
- Declarative Query language (SQL)
- Three-valued logic (null values)
- Potentially a lot of functionality in a single statement
Our Goals

1. To define a test criterion for SQL statements executed against a database
   - Assessing coverage of test data related to the query
   - Helping the tester in the task of populating test databases

2. To generate a suitable test database
   - Either Starting from an empty database
   - Or Starting from a large database (e.g. Production)
     - To produce a small test database
     - Keep the same test coverage/maximize test coverage

3. Features
   - Automation and efficiency
   - Able to manage a large set of SQL queries
(1) SQLFpc Coverage Criterion

- SQL Full Predicate Coverage criterion (SQLFpc)
  - What test situations/test coverage items/test requirements should be covered: Taylorize MCDC to tackle with specific SQL and database features
  - How to evaluate the coverage: Each test requirement is embodied in a SQL statement, called coverage rule. If coverage rule returns any row, then the corresponding test requirement is covered
  - How to create coverage rules: Elaborate a set of transformations of the original query to generate the coverage rules
  - Other features: Automated and efficient for large databases
  - Example of application: Compiere ERP queries
(1) How to measure test coverage (I)

How to measure test coverage of this simple query?

- SELECT * FROM departments D LEFT JOIN job_history H ON h.dep_id=d.dep_id WHERE dep_name LIKE 'IT%' AND loc_id<>1700

MCDC principle:

- Each single condition determines the output
- Example: a AND b:
  - 3 test requirements:  a AND b,  NOT a AND b,  a AND NOT b
- We should derive a test database with rows that satisfy each of the above test requirements:

<table>
<thead>
<tr>
<th>Situation to be covered / Test Requirement</th>
<th>dep_name</th>
<th>loc_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>a AND b</td>
<td>dep_name LIKE 'IT%' AND loc_id&lt;&gt;1700</td>
<td>IT Support</td>
</tr>
<tr>
<td>NOT a AND b</td>
<td>NOT dep_name LIKE 'IT%' AND loc_id&lt;&gt;1700</td>
<td>Marketing</td>
</tr>
<tr>
<td>a AND NOT b</td>
<td>dep_name LIKE 'IT%' AND NOT loc_id&lt;&gt;1700</td>
<td>IT Helpdesk</td>
</tr>
</tbody>
</table>
(1) How to measure test coverage (II)

- We create a coverage rule for each test requirement:
  - SELECT * FROM departments D LEFT JOIN job_history H ON h.dep_id=d.dep_id WHERE dep_name LIKE 'IT%' AND loc_id<>1700
  - SELECT * FROM departments IT% LEFT JOIN job_history H ON h.dep_id=d.dep_id WHERE NOT dep_name LIKE 'IT%' AND loc_id<>1700
  - SELECT * FROM departments D LEFT JOIN job_history H ON h.dep_id=d.dep_id WHERE dep_name LIKE 'IT%' AND NOT loc_id<>1700

- Each coverage rule retrieves only the rows that satisfy its test requirement
  - A coverage rule is covered if returns any row

- Approach:
  - To devise what test requirements must be met
  - To transform the original query into coverage rules (Δ) to check these requirements
(1) Transformations for select predicates (WHERE clauses)

- Codd 1990 notation (RM/v2):  \[ Z \leftarrow R[(a='x' \lor nl(a)) \land b>c] \]

  SELECT * FROM R WHERE (a='x' OR a IS NULL) AND b>c

1. \(a='x'\), OR \(\lor\) \(nl(a)\) false
2. \(a='x' \lor nl(a)\): AND \(\land\) \(b>c\) true

- \(a='x'\) determines the output if:
  - \(a='x' \land \neg nl(a) \land b>c\)
  - \(\neg a='x' \land \neg nl(a) \land b>c\)

### Full set of coverage rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Delta_T(q, a='x'))</td>
<td>(R[a='x' \land \neg nl(a) \land b&gt;c])</td>
</tr>
<tr>
<td>(\Delta_F(q, a='x'))</td>
<td>(R[\neg a='x' \land \neg nl(a) \land b&gt;c])</td>
</tr>
<tr>
<td>(\Delta_N(q, a='x', a))</td>
<td>(R[nl(a) \land b&gt;c])</td>
</tr>
<tr>
<td>(\Delta_N(q, nl(a), a))</td>
<td>(R[nl(a) \land b&gt;c])</td>
</tr>
<tr>
<td>(\Delta_{CB+}(q, b&gt;c))</td>
<td>(R[b=c+1 \land (a='x' \lor nl(a))])</td>
</tr>
<tr>
<td>(\Delta_{CB=}(q, b&gt;c))</td>
<td>(R[b=c \land (a='x' \lor nl(a))])</td>
</tr>
<tr>
<td>(\Delta_{CB-}(q, b&gt;c))</td>
<td>(R[b=c-1 \land (a='x' \lor nl(a))])</td>
</tr>
<tr>
<td>(\Delta_N(q, b&gt;c, b))</td>
<td>(R[nl(b) \land (a='x' \lor nl(a))])</td>
</tr>
<tr>
<td>(\Delta_N(q, b&gt;c, c))</td>
<td>(R[nl(c) \land (a='x' \lor nl(a))])</td>
</tr>
</tbody>
</table>
(1) Transformations for join predicates (JOIN clause)

- **Z \leftarrow R[A_1=B_1]^FS**
  
  ```
  SELECT * FROM R FULL OUTER JOIN S ON A1=B1
  ```

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A₀</td>
<td>A₁</td>
</tr>
<tr>
<td>Inner join</td>
<td>11</td>
<td>x</td>
</tr>
<tr>
<td>Left Outer Increment (LOI)</td>
<td>12</td>
<td>y</td>
</tr>
<tr>
<td>Right Outer Increment (ROI)</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

- **Coverage rules**

  \[\Delta_l(R[A_1=B_1]^FS) := R[A_1=B_1]^S\]
  \[\Delta_l(R[A_1=B_1]^FS) := (R[A_1=B_1]^S)[\neg \text{nl}(A_1) \land \text{nl}(B_1)]\]
  \[\Delta_R(R[A_1=B_1]^FS) := (R[A_1=B_1]^RS)[\text{nl}(A_1) \land \neg \text{nl}(B_1)]\]

  Example for 2nd rule:  
  ```
  SELECT * FROM R LEFT JOIN S ON A1=B1
  WHERE A1 IS NOT NULL AND B1 IS NULL
  ```
(1) Other features

- Example a general query:
  - `SELECT A,B,C,count(*) FROM R INNER JOIN S ON A0=B0 INNER JOIN T ON B1=C1 WHERE A2=B2 AND C2 IS NULL GROUP BY A0,A1 HAVING A3=10 AND sum(B3)>3`

- Other transformations to generate rules for:
  - Nested joins
  - Combinations of joins and WHERE clauses
  - GROUP BY clauses
  - Selection after grouping (HAVING)
  - Aggregate functions (count, min, max…)
  - CASE expressions (in SELECT and/or WHERE)
  - Subqueries (in SELECT and/or WHERE)
  - UNION and derived tables
(1) Results

- Case study: ERP Open Source: **Compiere**
- Set of views
  - 107 queries
  - 136 tables
  - 23 columns per table (average), 84 (maximum)
  - WHERE up to 24 conditions (C_Invoice_Candidate_v)
  - JOIN up to 15 different tables (C_Order_Header_v)
  - CASE, up to 19 expressions (C_Dunning_Line_vt)
  - UNION up to 15 queries (RV_UnPosted)
- Databases with 4, 10, 100, 1000 rows per table
  - Generated with DBMonster

http://in2test.lsi.uniovi.es/sqlfpc/

J. Tuya (2015)
## Results

### Coverage evaluation

- **Evolution of coverage per rule type**

<table>
<thead>
<tr>
<th>category</th>
<th>type</th>
<th>Num Rules</th>
<th>SQLFpc Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Select Operators (WHERE)</td>
<td>$\Delta_B$</td>
<td>261</td>
<td>0.38</td>
</tr>
<tr>
<td></td>
<td>$\Delta_N$</td>
<td>64</td>
<td>1.56</td>
</tr>
<tr>
<td></td>
<td>$\Delta_T$, $\Delta_F$</td>
<td>513</td>
<td>29.82</td>
</tr>
<tr>
<td></td>
<td><strong>Sub-total:</strong></td>
<td><strong>838</strong></td>
<td>18.50</td>
</tr>
<tr>
<td>Join Operators</td>
<td>$\Delta_I$</td>
<td>80</td>
<td>65.00</td>
</tr>
<tr>
<td></td>
<td>$\Delta_L$, $\Delta_{NL}$</td>
<td>291</td>
<td>22.68</td>
</tr>
<tr>
<td></td>
<td>$\Delta_R$, $\Delta_{NR}$</td>
<td>397</td>
<td>86.65</td>
</tr>
<tr>
<td></td>
<td><strong>Sub-total:</strong></td>
<td><strong>768</strong></td>
<td>60.16</td>
</tr>
<tr>
<td>Grouping and Aggregate</td>
<td>$\Delta_A$, $\Delta_{AN}$</td>
<td>152</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>$\Delta_G$, $\Delta_{GA}$</td>
<td>144</td>
<td>2.78</td>
</tr>
<tr>
<td></td>
<td><strong>Sub-total:</strong></td>
<td><strong>296</strong></td>
<td>1.35</td>
</tr>
<tr>
<td><strong>TOTAL:</strong></td>
<td></td>
<td><strong>1,902</strong></td>
<td>32.65</td>
</tr>
</tbody>
</table>
(1) Results
Effectiveness and efficiency

- **Mutation Coverage (effectiveness)**

<table>
<thead>
<tr>
<th>Category</th>
<th>Num. Mutants</th>
<th>Mutation Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SQL Clause mutation (SC)</td>
<td>3,293</td>
<td>40.84</td>
</tr>
<tr>
<td>Null Mutation Operators (NL)</td>
<td>1,754</td>
<td>25.09</td>
</tr>
<tr>
<td>Operator Replacement (OR)</td>
<td>24,861</td>
<td>53.15</td>
</tr>
<tr>
<td>Identifier Replacement (IR)</td>
<td>162,295</td>
<td>65.51</td>
</tr>
<tr>
<td><strong>TOTAL:</strong></td>
<td>192,203</td>
<td>63.12</td>
</tr>
</tbody>
</table>

- **Performance: Pentium IV 3 Ghz, Local Oracle DB**
  - Generate all rules (1902): 45.9 seconds
  - Evaluate all rules: from 128.5 to 170.5 seconds
(1) Results
Using SQLFpc to develop a test DB

- Iterative process (starting from an empty or previously populated database)
  - Execute all uncovered rules
  - Select a non covered rule
  - Insert rows into DB to cover this rule

C_Invoice_Candidate_v

C_Invoice_LineTax_vt
(2) Test database reduction vs Test database generation

Test Engineer

Test Database Generation

Test Criterion (SQLFpc)

Executed SQL Queries

Database Application

Test Database (small)

(generated rows)
(2) Test database reduction vs Test database generation

Test Database Generation

Test Engineer

Test Database Reduction

Test Criterion (SQLFpc)

Test Database (small)

Production Database (large)

Database Application

Executed SQL Queries

Executed SQL Queries

(created rows)

(selected rows)
(2) Test database reduction (II)

- Query-Aware Test Database Reduction
  - What test situations/test requirements should be met: We use SQLFpc coverage rules
  - How to select the minimum set of rows that satisfy each coverage rule
    - Transforming each coverage rule to get only the rows that fulfil the rule
    - Devising strategies to select the smallest number of rows to fulfil the rule
  - Features:
    - Automated and efficient for large databases
    - Very small reduced database which preserves the initial coverage
  - Example of application to Compiere and TPC-H benchmark
(2) Test database reduction (III)

- **Approach:**
  - **To transform** each coverage rule ($\Delta$) into a reduction rule ($\delta$) which reveals the primary keys of all rows that cover $\Delta$
  - **Execute** the reduction rule against the original database
  - **Select** the minimum set of rows that are needed to preserve coverage (reduction procedure)
    - Minimizing the number of rows to be inserted into the reduced database
    - Reusing rows previously inserted to cover previous rules
    - Minimizing number of rows in groups while satisfying HAVING
  - **Insert** these rows into the destination (reduced) database
(2) Results

http://in2test.lsi.uniovi.es/qashrink/

Case study (I): ERP Open Source: Compiere
Case study (II): Standard benchmark: TPC-H

- 8 tables
- 11 out of 22 very complex queries (all 22 currently supported)
- 159 coverage rules
- 6 databases
  - Size from 1 Mb to 100Gb
  - No. of rows from 2,488 to 865,860,820
  - Generated using the standard TPC-H procedure
(2) Results - Effectiveness and degree of reduction

- Coverage after reduction
  - Without optimizations: Coverage increases!!
  - Compiere: 51.53%-51.75%
  - TPC-H: 57.86%-62.26%

- Mutation Score decreases
  - Compiere 86.62%-79.84%
  - TPCH: 81.16%-77.54%
  - Normal s/ Rothermel (7.3%)

- Size of DB after reduction
  - Compiere: 1,293 rows
  - TPC-H: 32 to 341 rows
  - From 1,5 to 2,5 rows per rule
(2) Results - Efficiency

- Time spent into reduction
  - Linear increase related to the DB size
  - Smallest time by using optimization

- Normalized reduction time
  - Does not depends on the size of original database
  - Exception: when secondary storage is used for TEMPDB
  - Significant time decreasing when using optimizations
Automation: Architecture

test4data.com  http://giis.uniovi.es/tools/

- Web Browser
  - SQL Query + Database Schema
  - Coverage Rules / SQL Mutants

- SQLRules
  - SQL Query + Database Connection Info
  - Coverage Rules / SQL Mutants + Actual Coverage
  - Reduced Database

- QAShrink
  - Initial Database

- QAGrow
  - Populated Database

- Core
  - SQLFpcWeb
  - SQLMutationWeb
  - SQLFpcWS (Web Service)
  - SQLMutationWS (Web Service)
(3) Using data for decision making

Classical approach

OLTP Database

Relational Model

Star Model

OLAP Database (Data Warehouse)

Business Intelligence

Online Applic.

Visualization, Manual Analysis

J. Tuya (2015)
(3) Using data for decision making
Big Data world

- Online Applic.
- Other Data Sources

NoSQL Database(s)
- No Relational Model
- No Transactional (eventual consistency)
- No Normalized Replication/Distribution

Complex Massive Data Processing
- e.g. Hadoop with MapReduce

Data Analytics
- Visualization, Manual Analysis

Business Intelligence

J. Tuya (2015) Testing SQL and NoSQL Databases
(3) Data Storage: NoSQL Databases

- NoSQL (Not Only SQL): Non Relational SGBD, generally don’t use SQL to manipulate data. ACID properties (Atomicity, Consistency, Isolation, Durability) aren’t granted
  - Column oriented (e.g. HBase, Cassandra)
  - Key-Value (e.g. Riak)
  - Document oriented (e.g. MongoDB)
  - Graph oriented (e.g. Neo4J)

- Very large set of different technologies
(3) Issues from the testing point of view (NoSQL)

- Where is the conceptual model?
  - In the mind of the developer?
  - Not explicit relations, No triggers
  - Program shall maintain consistency

- Create “conceptual model” to derive tests

- Test for changes in programs when num. of tables grow

- Test how duplicated información is hanled

- Test for data quality (many sources)

- Even for Big Data test database should be small (Oracle problem)

- White-box test highly dependent on technology
REDUCE subq2.school, subq2.meme, subq2.cnt
USING 'top10.py' AS (school.meme.cnt)
FROM (SELECT subq1.school, subq1.meme, COUNT(1) AS cnt
FROM (MAP b.school, a.status
USING 'meme-extractor.py' AS (school.meme)
FROM status_updates a JOIN profiles b
ON (a.pid = b.pid))
GROUP BY a.status)
INTO 'top10.py'
USING 'top10.py'
GROUP BY subq2.school, subq2.meme, subq2.cnt

db.zipcodes.aggregate{
  $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$population" } },
  $sort: { pop: 1 } },
$group: { _id : "$_id.state",
  biggestCity: { $last: "$_id.city" },
  biggestPop: { $last: "$pop" },
  smallestCity: { $first: "$_id.city" },
  smallestPop: { $first: "$pop" } } },
// the following $project is optional, and modifies the output format.
$project: { _id: 0,
  state: "$_id",
  biggestCity: { name: "biggestCity", pop: "biggestPop" },
  smallestCity: { name: "smallestCity", pop: "smallestPop" } } }

J. Tuya (2015)
(3) Massive Data Processing

- MapReduce
  - Transformation of Data in form of pairs key-value
  - Programmer develops some functions
  - Infrastructure (e.g. hadoop) decides how (data) and when (distribution) the functions and are executed

J. Tuya (2015)
(3) Issues from the testing point of view (MapReduce)

- MapReduce jobs fail
  - Studies: 3% or 1.4% - 33%
- Dependency on infrastructure
  - Non deterministic order of execution
  - Some functions may be called or omitted
  - Large number of parameters to tune
- Variety of jobs
  - single-job, long-duration; many jobs iterative, short-duration inner-loops; complex combinations & sequences
- MapReduce combination with others (e.g. Hive, Pig Latin)
- Data quality / oracle problem
Summary and Conclusions

- Test coverage criterion for SQL
  - Assess test coverage of a database in relation to a set of queries
  - Useful to guide or automate the task of populating test databases

- Test database reduction
  - Preserves coverage from very large to small database
  - Time proportional to the execution of each query
  - Fully Automated, Large set of SQL

- NoSQL
  - Many new open problems
  - Many are specific for each technology