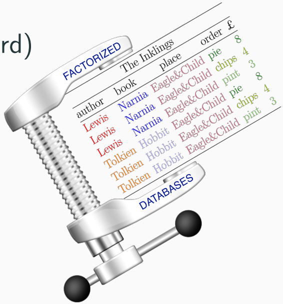
Factorised Databases

fdbresearch.github.io

Dan Olteanu & FDB Team (University of Oxford)

November 2017

Alan Turing Institute



Data Management Challenge

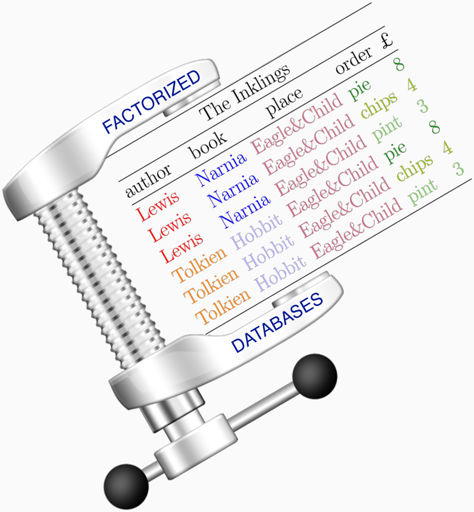
Data management is a defining challenge of our time

- Much cheaper to generate and process data
- Society is becoming increasingly more computational

Existing efforts on scalable relational data systems unsatisfactory

- Highly redundant data representation and processing
- Tractability map for queries and analytics mostly uncharted

Current Focus: Factorised Databases



Research Agenda

Investigate foundational and systems aspects of scalable data management at the confluence of

- **Compression**,
- **Distribution**, and
- **Approximation**

for mixed workloads of

- **Database Queries** and
- **Optimisation Problems**

over Relational Data.

Factorised Databases in Three Minutes!

Consider the natural join on the column Zip of the three relations:

House

Zip	Area	HPrice
OX1	80 m^2	300k
OX1	50 m^2	200k
OX2	60 m^2	249k
OX2	80 m^2	260k

Shop

Zip	SName	Hours
OX1	M&S	8
OX1	Tesco	24
OX1	CoOp	10
OX2	M&S	6
OX2	Zara	9

Restaurant

Zip	RName	RPrice
OX1	Ask	£
OX1	Zizzi	££
OX2	Eat	£
OX2	GBK	££

Factorised Databases in Three Minutes!

Consider the natural join on the column Zip of the three relations:

House

Zip	Area	HPrice
OX1	80 m^2	300k
OX1	50 m^2	200k
OX2	60 m^2	249k
OX2	80 m^2	260k

Shop

Zip	SName	Hours
OX1	M&S	8
OX1	Tesco	24
OX1	CoOp	10
OX2	M&S	6
OX2	Zara	9

Restaurant

Zip	RName	RPrice
OX1	Ask	£
OX1	Zizzi	££
OX2	Eat	£
OX2	GBK	££

The join lists the combinations of input tuples for each postcode:

House ⋈ Shop ⋈ Restaurant

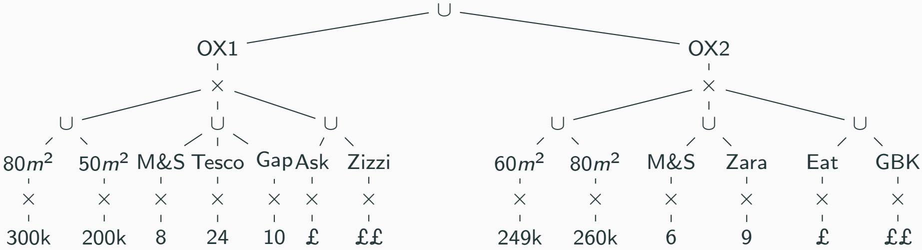
Zip	Area	HPrice	SName	Hours	RName	RPrice
OX1	80 m^2	300k	M&S	8	Ask	£
OX1	80 m^2	300k	M&S	8	Zizzi	££
OX1	80 m^2	300k	Tesco	24	Ask	£
OX1	80 m^2	300k	Tesco	24	Zizzi	££

. . . 8 more combinations for OX1 and 8 more for OX2 . . .

Factorised Databases in Three Minutes!

A factorised join avoids redundancy by exploiting

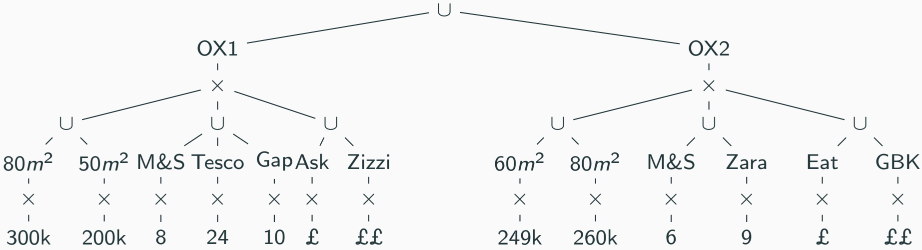
- the conditional independence in the join result and
- the distributivity of Cartesian product over union.



Factorised Databases in Three Minutes!

A factorised join avoids redundancy by exploiting

- the conditional independence in the join result and
- the distributivity of Cartesian product over union.

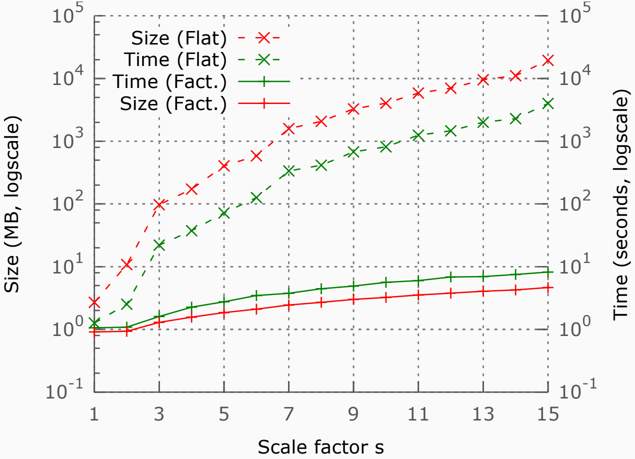


Which factorised joins have worst-case optimal size?

Can we compute factorised joins worst-case optimally?

Factorised Databases in Three Minutes!

- Assume 25K zipcodes and s records per zipcode and relation.
- The factorised join stays **linear** in the input size.
- The standard join becomes **cubic** in the input size.



Factorised In-Database Analytics

Scalable techniques for machine learning over databases that

- exploit the relational structure (schema, query, dependencies),
- push the learning task inside the database query engine, and
- factorise its computation.

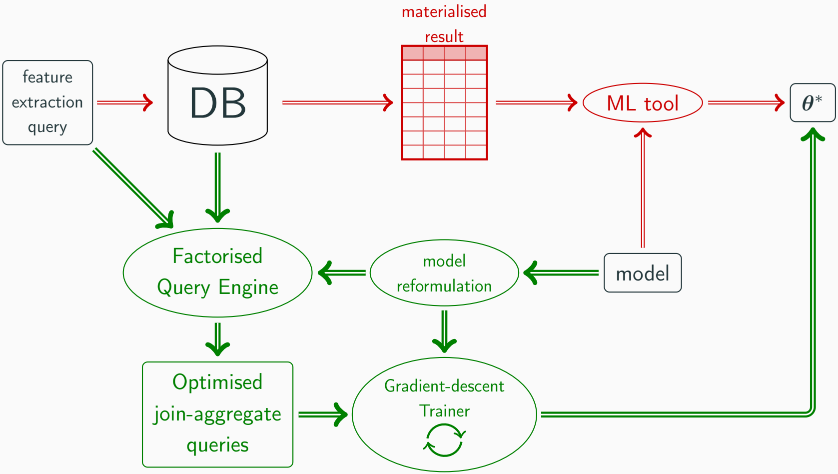
Prototypes @Oxford and @LogicBlox (now Infor) support:

- ridge linear regression, polynomial regression, factorisation machines; logistic regression, SVM; PCA.
- (on-going) decision trees, frequent itemset, ..

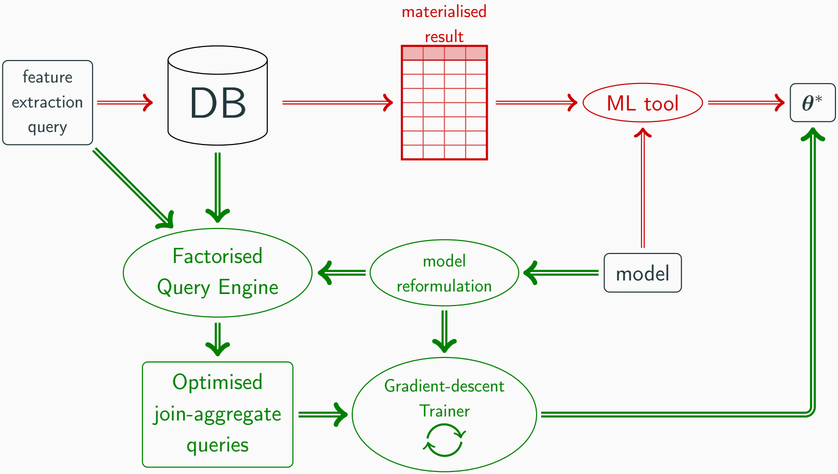
Why In-Database Analytics?

- Move the analytics code, not the data
  - Avoid expensive data export/import
  - Exploit database technologies
  - Build better models using larger datasets
- Cast analytics code as join-aggregate queries
  - Many similar queries that massively share computation
  - Fixpoint computation needed for model convergence

In-database vs. Out-of-database Analytics



In-database vs. Out-of-database Analytics



Complexity gap for some models: $\mathcal{O}(|DB|^{fhtw})$ vs. $\mathcal{O}(|DB|^n)$, where n is the number of relations in the database and $fhtw \ll n$ is the fractional hypertree width of the join of all database relations.

Does It Pay Off in Practice?

Retailer dataset (records)		excerpt (17M)	full (86M)
Linear regression			
Features	(cont+categ)	33 + 55	33+3,653
Aggregates	(cont+categ)	595+2,418	595+145k
MadLib	Learn	1,898.35 sec	$> 24h$
R	Join (PSQL)	50.63 sec	–
	Export/Import	308.83 sec	–
	Learn	490.13 sec	–
Our approach	Aggregate+Join	25.51 sec	380.31 sec
(1core, commodity machine)	Converge (runs)	0.02 (343) sec	8.82 (366) sec
Polynomial regression degree 2			
Features	(cont+categ)	562+2,363	562+141k
Aggregates	(cont+categ)	158k+742k	158k+37M
MadLib	Learn	$> 24h$	–
Our approach	Aggregate+Join	132.43 sec	1,819.80 sec
(1core, commodity machine)	Converge (runs)	3.27 (321) sec	219.51 (180) sec

Real-Time In-Database Analytics

- Datasets continuously evolve over time
  - E.g.: data streams from sensors, social networks, apps
- Real-time analytics over streaming data
  - Users want fresh up-to-date data models

Web Analytics

Sensor Networks

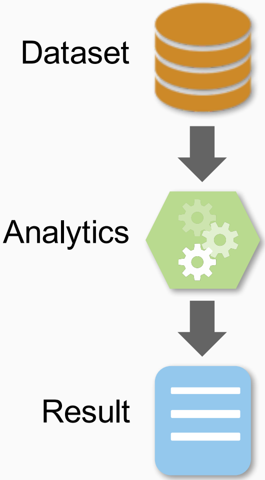
Cloud Monitoring

EVENTS → RUNTIME ENGINE → DECISION SUPPORT → ACTIONS

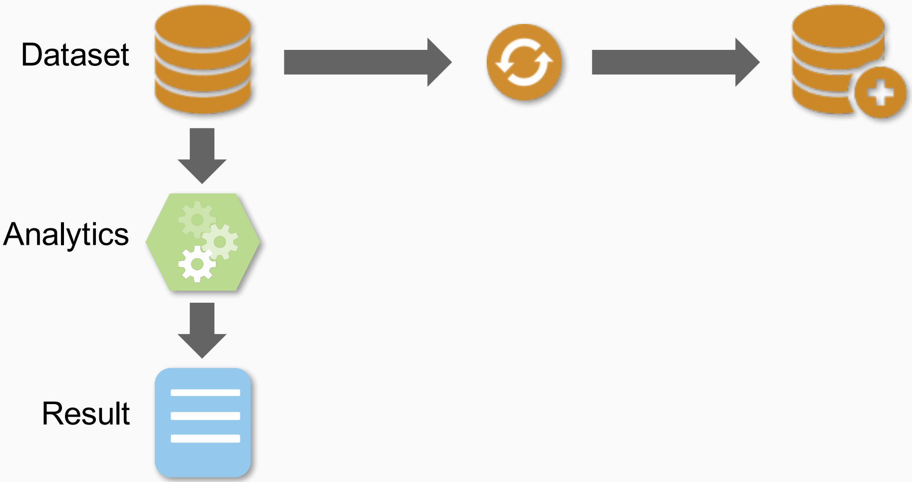
Continuously arriving data

Continuously evaluated views

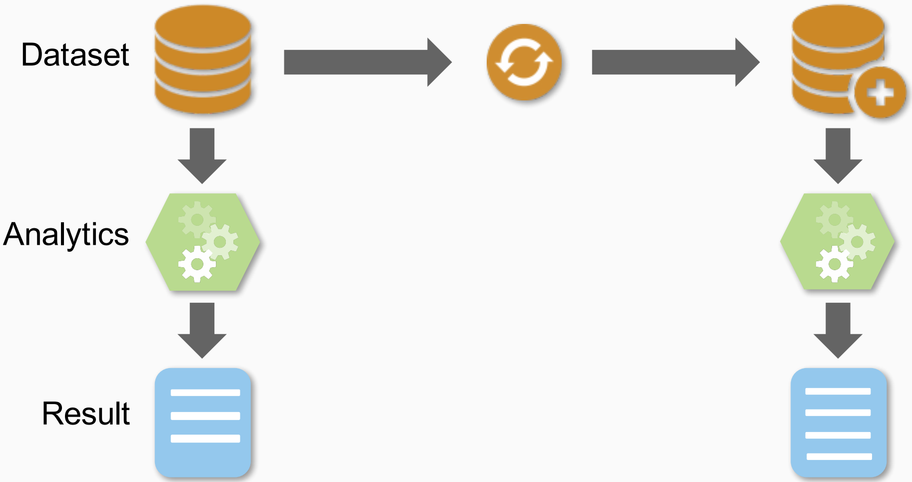
Real-Time Processing via Incremental Maintenance



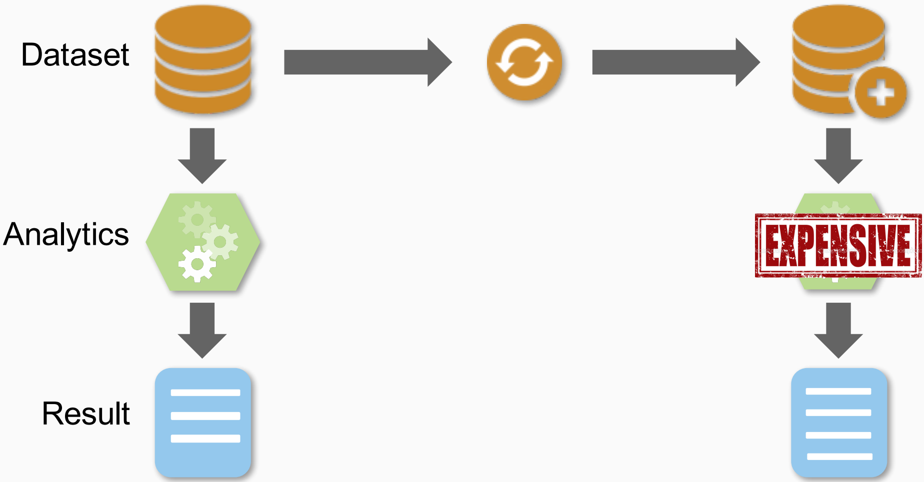
Real-Time Processing via Incremental Maintenance



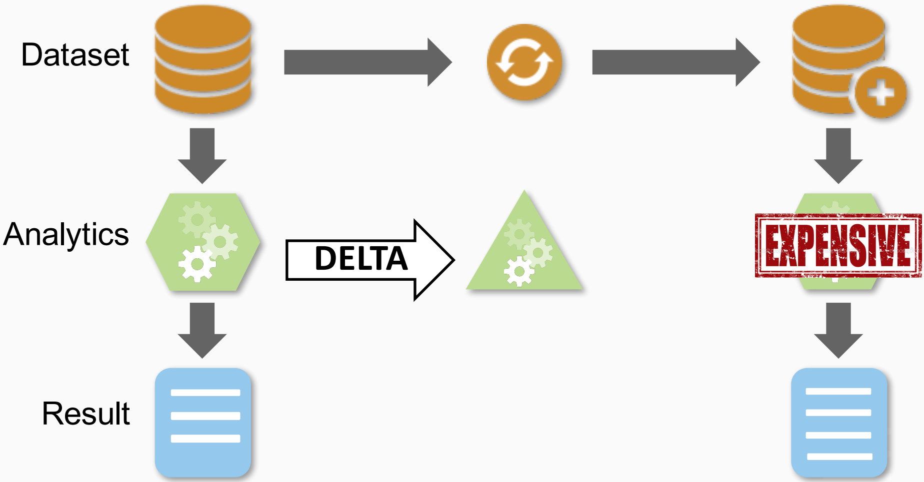
Real-Time Processing via Incremental Maintenance



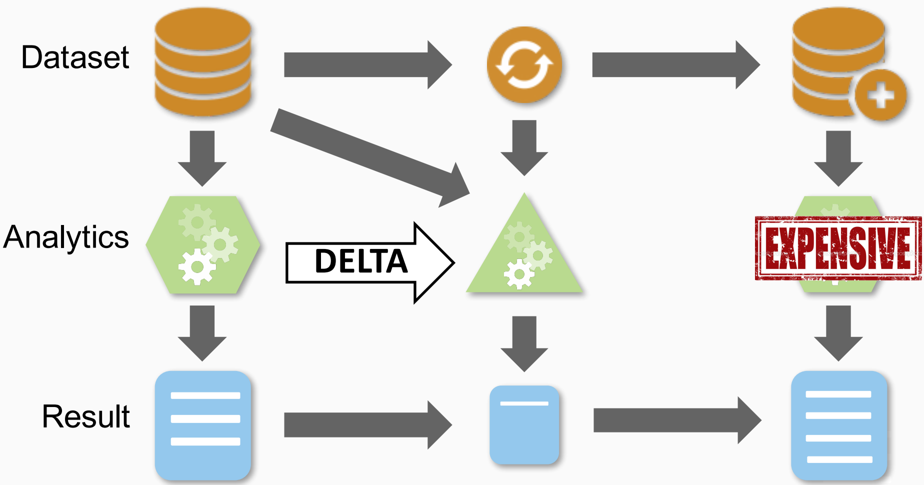
Real-Time Processing via Incremental Maintenance



Real-Time Processing via Incremental Maintenance



Real-Time Processing via Incremental Maintenance



Unified Framework for Real-Time In-Database Analytics

Unified framework F-IVM for a host of tasks, e.g.,

- database join-aggregate queries
- gradient computation for least-squares regression models
- matrix chain multiplication

Unified Framework for Real-Time In-Database Analytics

Unified framework F-IVM for a host of tasks, e.g.,

- database join-aggregate queries
- gradient computation for least-squares regression models
- matrix chain multiplication

Key to unified computation:

- same in-database computation, coupled with
- task-specific rings $(\mathcal{D}, +, *, \mathbf{0}, \mathbf{1})$

Unified Framework for Real-Time In-Database Analytics

Unified framework F-IVM for a host of tasks, e.g.,

- database join-aggregate queries
- gradient computation for least-squares regression models
- matrix chain multiplication

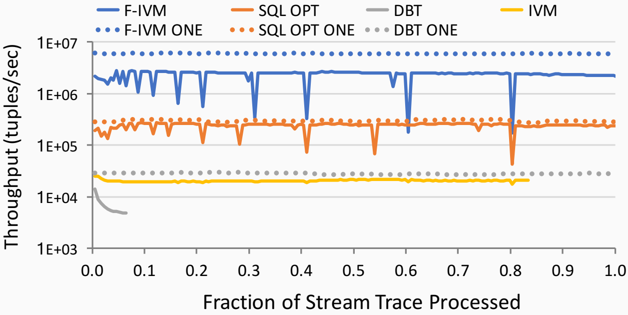
Key to unified computation:

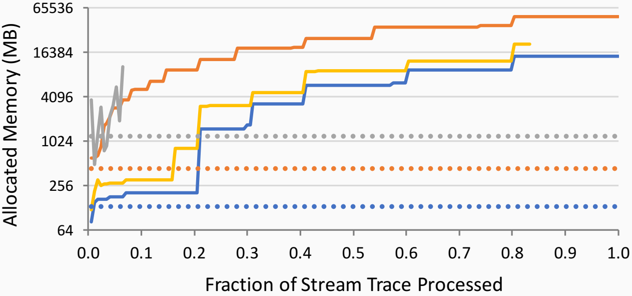
- same in-database computation, coupled with
- task-specific rings $(\mathcal{D}, +, *, \mathbf{0}, \mathbf{1})$

Key to performance: Triple-lock factorisation for

1. delta processing, compiled to optimised C++ code
2. representation of the result
3. bulk updates via tensor decomposition techniques

Performance for Learning a Linear Regression Model





Want To Find Out More?

- Dec 1, 2017: Talk on in-database learning, Turing Logic seminar series
- Jan 29, 2018: Advanced 3-hour Turing Data Science class

`fdbresearch.github.io`

Thank you!