Databases & Your Research

What, How, When/Why

Nate Derbinsky

Goals

By the end of this talk, you should…

1. have a big-picture understanding of core database system distinctions and topics
2. have a basic understanding of the relational data model, SQL, and query processing
3. have a practical sense of when and how the application of database software and/or techniques may benefit your research

Outline

What is a Database Management System?

– Core distinctions/topics

Relational Databases 101

– Relational model, SQL, indexing, tools/resources

Databases & Your Research

– Use cases

DataBase Management System (DBMS)

A system intended to organize, store, and retrieve large amounts of data easily

Allows users to…

- Specify a *schema* (logical structure)
- Store large amounts of data
- Query and modify data quickly and reliably
- Control access from many users at once

Core DBMS Distinctions

- Logical Data Model
- Query Interface
- Transaction Support
- Concurrency
- Performance

…

Logical Data Model

Constraints on logical data representation and structure

Dominant: relational

- XML (and other document stores)
- Object-oriented
- Graph
- Key-value
- Triplestore

…

Query Interface

How a user interacts with stored data

1. Logical/High-level Query Language
   - SQL (Structured Query Language) dialects
   - XQuery
   - SPARQL

2. Programmatic API
   - Native
   - Abstraction Library (ODBC)

Transaction Support

How the system handles a sequence of data read/write operations

Atomicity	Operations must be “all or nothing.”
Consistency	Data must shift between truthful states.
Isolation	Concurrent users shall not see dirty data.
Durability	Data changes survive system failure.

Concurrency

The ability of the DBMS to service multiple requests

Typical classes:

- File-based
- Client-server
- Distributed

CAP Theorem: impossible to simultaneously guarantee…

- Consistency (all nodes see same data)
- Availability (service survival despite node failures)
- Partition tolerance (service despite message loss)

Performance

- Memory management
- Disk management
- Locking
- Query execution
  - Indexing
  - Planning
- Optimization: OLTP, Analytics, Hashing, Desktop

…

A Representative Sample

	Embedded	Desktop	Online Transaction Processing (OLTP)	Cloud/Distributed
Proprietary		FileMaker	ORACLE Microsoft SQL Server IBM DB2	
Open	ORACLE BERKELEY DB 11g SQLite		MySQL PostgreSQL INGRES	HBASE Cassandra MEMCACHED

Example: SQLite

Logical Data Model	Relational
Query Interface	SQL(ish), API
Transaction Support	ACID
Concurrency	File-based
Performance	Limited scaling; limited query optimizer; in-process

Example: MySQL

Logical Data Model	Relational
Query Interface	SQL, API
Transaction Support	ACID
Concurrency	Client-server, Multi-server
Performance	Limited single-query parallelism

Example: Cassandra

Logical Data Model	Key-Value			
Query Interface	Limited API			
Transaction Support	Eventual Consistency			
Concurrency	Distributed			
Performance	50GB*			
	MySQL		Cassandra	
	Write	Read	Write	Read
	300ms	350ms	0.12ms	15ms

*The Cassandra Distributed Database (2010)
http://www.parleys.com/#st=5&id=1866

Relational Databases 101

- Data Model (informally)
- SQL Basics
- Indexing
- Tools/Resources

The Relational Data Model

- Database: set of tables
- Table: set of *n* columns, bag of rows
  – Column: name, [type]
  – Row: *n*-tuple (value for each column)

Example: SMem Tables

smem7_lti							
id	letter	num	child_ct	act_value	access_n	access_t	access_1
1	65	1	2	1	1	1	1

smem7_web				
parent_id	attr	val_const	val_lti	act_value
1	1	2	0	1
1	3	0	1	1

smem7_symbols_str	
id	sym_const
1	foo
2	bar
3	self

smem7_ascii	
ascii_num	ascii_chr
65	A
66	B
…	

```
smem --add {
     (<a1> ^foo bar
           ^self <a1>)
}
```

Data Model Notes

Normalization

- Data organization to minimize redundancy
- Forms: 3NF, BCNF, …

Key: column(s) useful in identifying rows

- Primary key: unique identification

Data meaning/use arises from queries that relate rows of distinct tables (usually via keys)

Structured Query Language (SQL)

Declarative language to describe *what* data to get/modify in a relational database

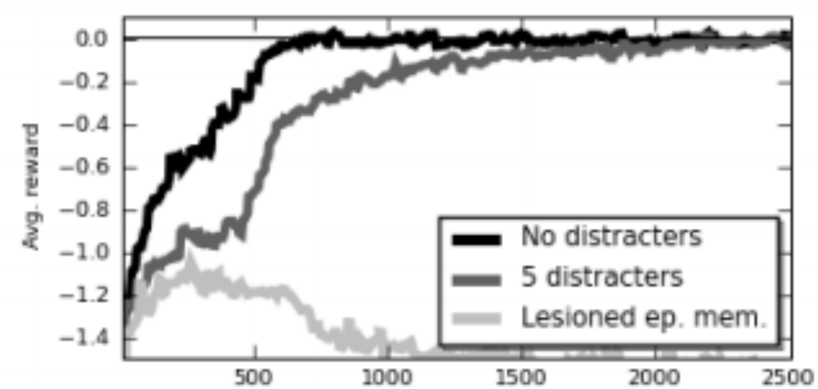
Core commands

- CREATE
- INSERT
- UPDATE
- DELETE
- SELECT

Running Example: RL

Experimental Data

- Multiple conditions
- Multiple trials
- Multiple episodes
- Metric: avg. reward



CREATE Table

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
…
);
```

CREATE Table: Example

```
CREATE TABLE my_experiment
(
exp_condition VARCHAR(100),
trial_num INT,
episode_num INT,
reward DOUBLE
);
```

CREATE Table: Result

my_experiment			
exp_condition	trial_num	episode_num	reward

INSERT Data

```
INSERT INTO table_name
(column1, column2, …)
VALUES
(value1, value2, …);
```

INSERT Data: Example

```
INSERT INTO my_experiment
(exp_condition, trial_num,
episode_num, reward)
VALUES
('simple', '1', '1', '-10');
```

INSERT Data: Result

my_experiment			
exp_condition	**trial_num**	**episode_num**	**reward**
simple	1	1	-10

A Few INSERTs Later…

my_experiment			
exp_condition	trial_num	episode_num	reward
simple	1	1	-10
simple	1	2	-9
simple	1	3	-10
complex	1	1	-10
complex	1	2	-5
complex	1	3	-1

UPDATE Data

```
UPDATE table_name
SET column1=value1,
column2=value2, …
WHERE [condition(s)];
```

UPDATE Data: Example

```
UPDATE my_experiment
SET exp_condition='stupid'
WHERE exp_condition='simple';
```

UPDATE Data: Result

my_experiment			
exp_condition	**trial_num**	**episode_num**	**reward**
stupid	1	1	-10
stupid	1	2	-9
stupid	1	3	-10
complex	1	1	-10
complex	1	2	-5
complex	1	3	-1

DELETE Data

```
DELETE FROM table_name
WHERE [condition(s)];
```

SELECT Data

```
SELECT
column1, column2, …
FROM
table1, table2, …
WHERE
[condition(s)]
GROUP BY
column1, column2, …
ORDER BY
column1 [ASC/DESC], …
```

<u>Aggregation Functions</u>
Count, Average, Min, Max
StdDev, Variance
…

<u>Comparison Functions</u>
=, <>, >, <, …
IN, BETWEEN, STRCMP
LIKE (regex)
…

<u>Output</u>
A "result set" containing
>=0 rows

A "cursor" to the first row

SELECT Data: Example

```
SELECT
exp_condition, trial_num, episode_num,
AVG(reward) AS avg_reward
FROM
my_experiment
WHERE
exp_condition <> 'old'
GROUP BY
exp_condition, trial_num, episode_num
ORDER BY
trial_num ASC, episode_num ASC,
exp_condition DESC
```

SELECT Data: Result

exp_condition ↑	trial_num ↓	episode_num ↓	avg_reward
stupid	1	1	-10.00
complex	1	1	-10.00
stupid	1	2	-9.00
complex	1	2	-5.00
stupid	1	3	-10.00
complex	1	3	-1.00

SELECT 201: Joins

```
SELECT id, ascii_chr, num
FROM smem7_lti, smem7_ascii
WHERE
smem7_lti.letter=
smem7_ascii.ascii_num
```

smem7_ascii	
ascii_num	ascii_chr
65	A
66	B
…	

smem7_lti							
id	letter	num	child_ct	act_value	access_n	access_t	access_1
1	65	1	2	1	1	1	1

SELECT 201: Joins

```
SELECT id, ascii_chr, num
FROM smem7_lti, smem7_ascii
WHERE
smem7_lti.letter=
smem7_ascii.ascii_num
```

id	ascii_chr	num
1	A	1

SELECT 201: Sub-Queries

```
SELECT id
FROM smem7_symbols_str
WHERE sym_const='foo'
```

smem7_web				
parent_id	attr	val_const	val_lti	act_value
1	1	2	0	1
1	3	0	1	1

smem7_symbols_str	
id	sym_const
1	foo
2	bar
3	self

SELECT 201: Sub-Queries

```
SELECT id
FROM smem7_symbols_str
WHERE sym_const='foo'
```

id
1

SELECT 201: Sub-Queries

```
SELECT *
FROM smem7_web
WHERE attr=
(SELECT id FROM smem7_symbols_str WHERE sym_const='foo')
```

smem7_web				
parent_id	attr	val_const	val_lti	act_value
1	1	2	0	1
1	3	0	1	1

smem7_symbols_str	
id	sym_const
1	foo
2	bar
3	self

SELECT 201: Sub-Queries

```
SELECT *
FROM smem7_web
WHERE attr=
(SELECT id FROM smem7_symbols_str WHERE sym_const='foo')
```

parent_id	attr	val_const	val_lti	act_value
1	1	2	0	1

Additional SQL Topics

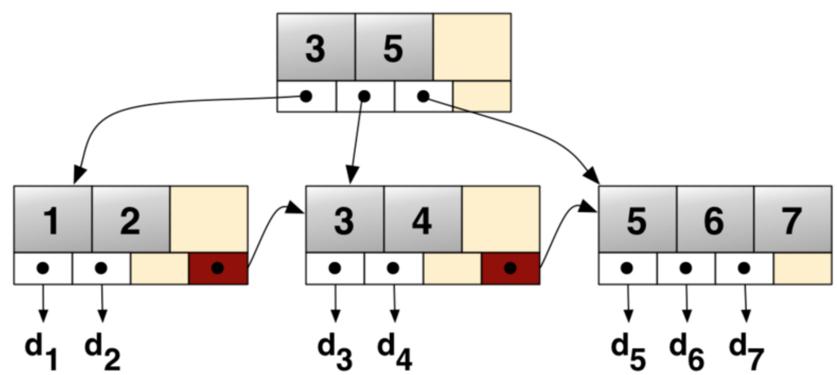
- Transactions
- Constraints
- Triggers
- Views
- Procedural SQL Extensions

…

Indexes: B+-Trees

Most common index

- Balanced tree
- Bounded out-degree
- Sorted, linked list of content pointers at leaves
- Usually keep top k-levels in RAM for fast lookups
- O(log) multi-key lookup
  - O(c) subsequent read



Indexes: Application

```
CREATE INDEX index_name ON
table_name (column1, column2, …)
```

– Ordering of columns is important!

Subsequent queries that test value of (column1) or (column1, column2) or … will automatically use index

- O(table size) -> O(log(table size))

Tools/Resources

- W3Schools SQL Reference
  - http://www.w3schools.com/sql/sql_quickref.asp
- phpMyAdmin: Web interface to MySQL
  - http://www.phpmyadmin.net
- SQLiteMan: Platform-independent GUI
  - http://sqliteman.com

Databases & Your Research

Draw on DBMS strengths, balanced with overhead (learning, computation, data conversion)

Use Cases

- Algorithmic Component (EpMem/SMem)
- Data Set Conversion (WordNet)
- Experimental Domain (Dice Game)
- Experimental Data Store (Speedy)

Use Case: EpMem/SMem

Approach	Nuggets	Coal
SQLite as easy B+-tree library	Efficient, well-tested/ supported Focus on interesting problems	Questionable scaling Library reliance -> bad surprises (optimizer, buffer management, typos) Limited inspect-ability of library (w/o becoming an expert) “Real” DBMS overhead is too great
File-based debugging/ analysis of memories	Existing, reliable tools on multiple platforms	

Use Case: WordNet

Approach	Nuggets	Coal
Conversion of existing data sets (WordNet, SemCor, Senseval-2/3) to SQL	Fast, easy, reliable queries using arbitrary tools/ languages (for analysis and experiments)	Time/computation overhead vs. existing tools and more basic data representation

Use Case: Dice Game

Approach	Nuggets	Coal
Conversion of existing database-backed application to SML I/O	Quick to get up and running (I=GET, O=POST) Uniform interface for Soar and humans Offload concurrency and multi-machine experimentation	Far too slow for mass RL experimentation

Use Case: Speedy

Approach	Nuggets	Coal
Database as experimental data store: a common format with benefits	Fast, reliable, reproducible data analysis	Overkill for small experiments
HTTP as easy intermediary between experimental data producer and SQL	Minimal effort to compute arbitrarily complex aggregations that scale	
	Easy to support web-accessibility and custom reporting	
	Robust to human/system failure	

Use Case Summary

Databases are not magic, but can serve useful roles in research

- Reliable, scalable, dynamic data analysis
- With caution, efficient data structures that scale and support inspection with powerful tools

Thoughts for future exploration…

- Incremental index statistic updating
- Graph DB algorithms
- Spatial DB algorithms
- Probabilistic DBs

Thanks :)

Questions?