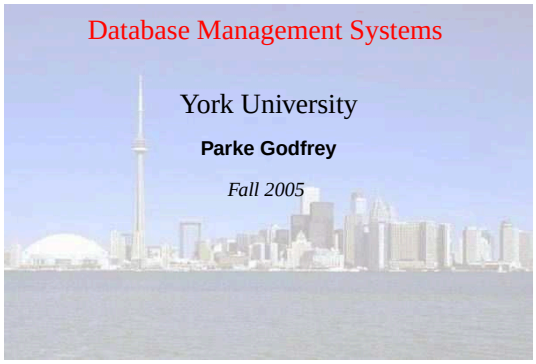
# CSE-4411



Database Management Systems

York University

**Parke Godfrey**

*Fall 2005*



# CSE-3421 vs CSE-4411

CSE-4411 is a continuation of CSE-3421, right?
More of the same, eh?

Ha! *No way.*

In this class, we focus on how to *build* a database system.
In CSE-3421, we focused on what functionality a database system provides, and how to *use* it.



# Data Independence

- Do not need to know how a *compiler* works to write a program.
- Do not need to know how an *operating system* is built to use one.
- Don't need to know how a *car* works to drive one.
- Don't need to know how a *database system* is built to use it.

- **physical data independence:** how the data is *logically* organized is independent of how it is *physically* organized. (There is also *logical data independence*. . .)
- **Codd's law**: Can only access and update the database via the "query language" (SQL).
- SQL is a *declarative* language.



# How to build a Database System?

Okay, more specifically, a *relational* database *management* system( RDBMS).

E.g., Oracle, IBM DB2, Microsoft SQL Server, Informix, MySQL, & Postgres.

In this class, we're going to build our own system!



# How to build a Database System?

## What is involved?

- What *functionality* do we need to support?
  - E.g., SQL
- What are our *design criteria*?
  - Should be fast. (At what?)
  - Must handle updates to the database and read-only queries efficiently.
    (Trade-offs involved!)
- What are our *design choices*? Our *design constraints*?
  - How will the available technology affect our design (*architecture*)?
    E.g., Main memory technologies (like CMOS) are volatile.



# I. The Physical Database

## Storage & Access

**Ensure that data is *permanent* and *safe*.**

**Goals:**

- permanence
- fast, random access
- fault tolerance (to support *crash recovery*)

**Design questions:**

- What devices / technology do we use?
- What data-structures do we use?
  How do we access given pieces of data quickly?

# II. The Query Processor

**How to evaluate (SQL) queries efficiently?** We need a
- query parser
- plan generator (and query optimizer)
  *Turns a valid SQL query into a "program" that answers the query.*
- query plan evaluator

**Problems:**
- SQL is reasonably complex.
- Not all (equivalent) queries are equal.
  *Some queries / query plans will evaluate inherently must faster.*

**Big issue:**
- How to "pick", or design, a good query plan for a query?



# A "Complex" Query

Supplier **S**: A (name), C (city)

Retailer **R**: B (name), C (city)

**Query:** *Which supplier has a location in every city of a retailer? Show such supplier (A) / retailer (B) pairs.*

$$\{\langle A, B\rangle \mid \forall \mathbf{C}(\langle B, C\rangle \in \mathbf{R} \rightarrow \langle A, C\rangle \in \mathbf{S})\}$$

$$\pi_{\mathsf{A,B}}(\mathbf{R} \times \mathbf{S}) - \pi_{\mathsf{A,B}}(\pi_{\mathsf{A,B,C}}(\pi_{\mathsf{A}}(\mathbf{S}) \times \mathbf{R}) - \mathbf{R} \bowtie \mathbf{S})$$



# A "Complex" Query
## in SQL

```
select A, B from R, S
except
select A, B from (
      select S.A, R.B, R.C from R, S
      except
      select S.A, R.B, R.C
            from R, S
            where R.C = S.C) as Z;
```

Any problems?



# A "Complex" Query
## Better?

```
select A, B
      from R, S
      where R.C = S.C
except
select A, B from (
      select S.A, R1.B, R2.C
            from R as R1, R as R2, S
            where R1.C = S.C and R1.B = R2.B
      except
      select S.A, R.B, R.C from R, S
            where R.C = S.C
) as Z;
```



# A "Complex" Query
## cleaned up

```
with
      J (A, B, C) as (
            select S.A, R.B, R.C
                  from R, S
                  where R.C = S.C)
select distinct A, B from J
except
select J.A, J.B
      from J, R
      where J.B = R.B and
            (J.A, J.B, R.C) not in
                  (select A, B, C from J);
```



# A "Complex" Query
## via COUNT

```
select J.A, J.B
      from (select S.A, R.B, count(*) as Cs
                  from R, S
                  where R.C = S.C
                  group by S.A, R.B) as J,
            (select B, count(*) as Cs
                  from R
                  group by B) as K
      where J.B = K.B and
            J.Cs = K.Cs;
```

# The Query Optimizer

- **Rewrite**
  - Rewrites the query into something "simpler", but means the same thing.
- **Cost-based**
  - Determine a "best" over-all query tree.
  - Pick the best *access path* for each table involved.
  - Assign the "best" algorithm to each operator ($\bowtie$, $\pi$, $\sigma$, ...).



# III. Database Management

- **transaction management**
  - *How do we ensure updates are made to the database correctly?*
- **concurrency control**
  - *How do we ensure that multiple X-act's occuring "simultaneously" are treated correctly?*
- **crash recovery**
  - *How do we recover from failures? (E.g., ARIES)*

**Properties:**

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability



# Buliding a Database System

## Anything we miss?

- host language support
  e.g., JDBC
- data definition language (DDL)
  e.g., CREATE TABLE ...
- administrative functions (for DBA's) & security
  e.g., GRANT ...
- ...

**What pieces / modules do we need to implement all this?**
**What's our architecture?**
Need a

- need a query optimizer
- a transaction manager
  - a lock manager for concurrency control
- a crash recovery mechanism
- ...



# Buliding a Database System

## Why study this?!

- It's fun!
- *Some* will get a job building RDBMSs.
  E.g., at IBM Toronto Laboratory (for DB2)
- Cannot be a *good* DB Administrator *without* understanding how the system works.
- Can be a better DB programmer when you understand how the system works.
- Lots of places are building database-like systems.
  *Can reuse the techniques and technologies from RDBMSs.*