III. The System

1. Database Management
   - schema management (CREATE, DROP, ALTER)
   - data management (INSERT, DELETE, UPDATE)

     scalable: handle billions or trillions of records
   - ◦ integrity protection
   - ◦ security and authorization (DBA)
2. Query Engine (SQL queries)
   - handle sophisticated (i.e., logically complex) queries
3. Application Program Support
   - how to talk with the rest of the world?
4. Transaction Management
   - *all-or-nothing*
5. Concurrency Control
   - handle thousands of requests concurrently
6. Crash Recovery
   - never lose data!
7. Tuning (DBA)
   - control over the physical database for efficiency

Database Designer's Waterfall for Integrity

1. **Structural and domain constraints (Design)**
   - normalized schema
   - primary key, foreign key, unique, and domain constraints
2. **Check Constraints**
   - row-level
   - with single-table sub-queries (aggregation)
   - with inter-table sub-queries (joins)
3. **Assertions (independent, inter-table constraints)**
4. **Triggers with rollback / abort**
5. **Integrity checking at the application level**

   (Aided by transaction management)

Design Considerations:

- Be as declarative as possible.

  Be as high up the waterfall as possible.
- Make good use of *views* (and other tools) for cleaner design.
- Must make trade-offs / concessions for sake of efficiency.

  (Sometimes nasty.)

Query Engine

- Process sophisticated (SQL) queries efficiently.
- Do so over databases with billions, or trillions, of tuples.

A key component is the *query optimizer*.

Application Program Support

Is SQL the only way to talk to the database (system)? *Yes!*

By Codd's dictate, SQL is the only way we have access to the data.

On the other hand, we need more than a SQL shell / window.

Programs need to be able to "talk" to the database, not just users.

And SQL is just a query language, not a programming language. So what about when we need programming?

Transaction Management

Certain sets of actions on the database we want to occur together.

Such a set of actions we call a *transaction*.

Properties:

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

Goes hand-in-hand with *concurrency control*. The RDBMS should be able to handle 100,000's transactions a minute.

Some of these will be in conflict.

So a transaction may

- commit or
- abort (a.k.a. rollback)

Concurrency Control

Handle 100,000's concurrent transactions. How?

- Locking protocols

  A transaction must lock resources so other transactions cannot have them.

- Serializability

  No matter how the transactions are processed, it looks like they were processed sequentially.

Crash Recovery

Never lose data. How?

After data is *committed*, it should never be lost (unless deleted).

- physical storage redundancy (e.g., RAID)
- write-ahead logs
- back-ups (DBA)

Physical Database Tuning

Control over the physical database for efficiency. (This is what DBA's do for a living.)

- Adjust parameters dictating resource allocation.
  - buffer pool management / main memory
- Dictate how tables are physically mapped to disk.
- Specify *indexes*
  - and materialized views, etc.
  - runstats (statistics used by the query optimizer)