Relational DBMSs

What is an RDBMS?

A relational database management system (RDBMS) is

- software designed to support large-scale data-intensive applications
- allowing high-level description of data (domains, constraints)
- with high-level access to the data (relational model, SQL)

- providing efficient storage and retrieval (disk/memory management)
- supporting multiple simultaneous users (privilege, protection)
- doing multiple simultaneous operations (transactions, concurrency)
- maintaining reliable access to the stored data (backup, recovery)

Note: databases provide persistent storage of information

Description of Data

RDBMSs implement ≅ the relational model.

Provide facilities to define:

- domains, attributes, tuples, tables
- constraints (domain, key, reference, ...)

Variations from the relational model:

- no strict requirement for tables to have keys
- bag semantics, rather than set semantics

- no direct support for multi-table constraints

RDBMS Operations

RDBMSs typically provide at least the following operations:

- create/remove a database
- create/remove/alter table schemas within a database
- insert/delete/update tuples within a table
- queries on data, define named queries (views)

Most also provide mechanisms for

- defining new data types
- implementing complex constraints (triggers)
- defining/storing procedural code to manipulate data
- creating/managing users of the database
- describing transactional behaviour (see later)

Access to Data

All modern RDBMSs provide access to the data via SQL.

Each RDBMS has its own dialect ≅ the SQL standard.

Most provide SQL via one or more of

- an interactive "shell" (e.g. psql, sqlite3, SQL*Plus, ...)
- programming language APIs (e.g. Java/JDBC, Python, C, ...)

Some operations are also implemented as utility commands

e.g. PostgreSQL's **`createdb`**, **`dropdb`**, **`createuser`**, ...

Access to Data (cont)

RDBMSs also provide access to meta-data (catalog).

Meta-data typically presented as collection of tables.

A standard **`INFORMATION_SCHEMA`** exists for meta-data.

DB users interact with catalog via meta-commands:

- PostgreSQL's **`\d`**, **`\d`** *Table*
- SQLite's **`.schema`**, **`.schema`** *Table*
- Oracle's **`select * from tab`**

DB admin typically also has SQL access to catalog.

Schema/Data Import/Export

RDBMSs typically provide mechanisms for

- loading schemas (typically from a file)
- bulk upload of data into tables (from files)
- saving entire databases (data + meta-data)

- clearing tables (e.g. **`delete from Staff`**)
- clearing schema (not in PostgreSQL)

Available via command line or interactive SQL shell.

Query Engine

RDBMS query engines implement RA operations

- projection (tuple manipulation; generalised projection)
- selection (tuple filtering; indexes, hashing)
- join (nested–loop, hash–join, indexed–join)
- set operations (union, intersection, difference)
- aggregation (e.g. count, sum, avg, ...)
- grouping (group–by, group filtering (via having))

Privilege/Protection

RDBMSs typically provide role-based user management:

- individual users have username/password
- users may be associated with multiple roles
- roles are assigned privileges (e.g. create tables, view table R)

Authentication via username/password gives access to DBs.

Roles determine what can be done within a DB.

Transactions/Concurrency

Often in application programming

- a single application-level operation (transaction)
- involves mutliple DBMS-level operations (e.g. insert, update)

To faithfully represent the application-level operation:

- either all DBMS-level operations must complete

- or all DBMS–level operations must fail

If the transaction fails partway

- any completed DBMS operations must be undone
- the DBMS should enforce this automatically

Transactions treat a group of DBMS operations as atomic.

Transactions/Concurrency (cont)

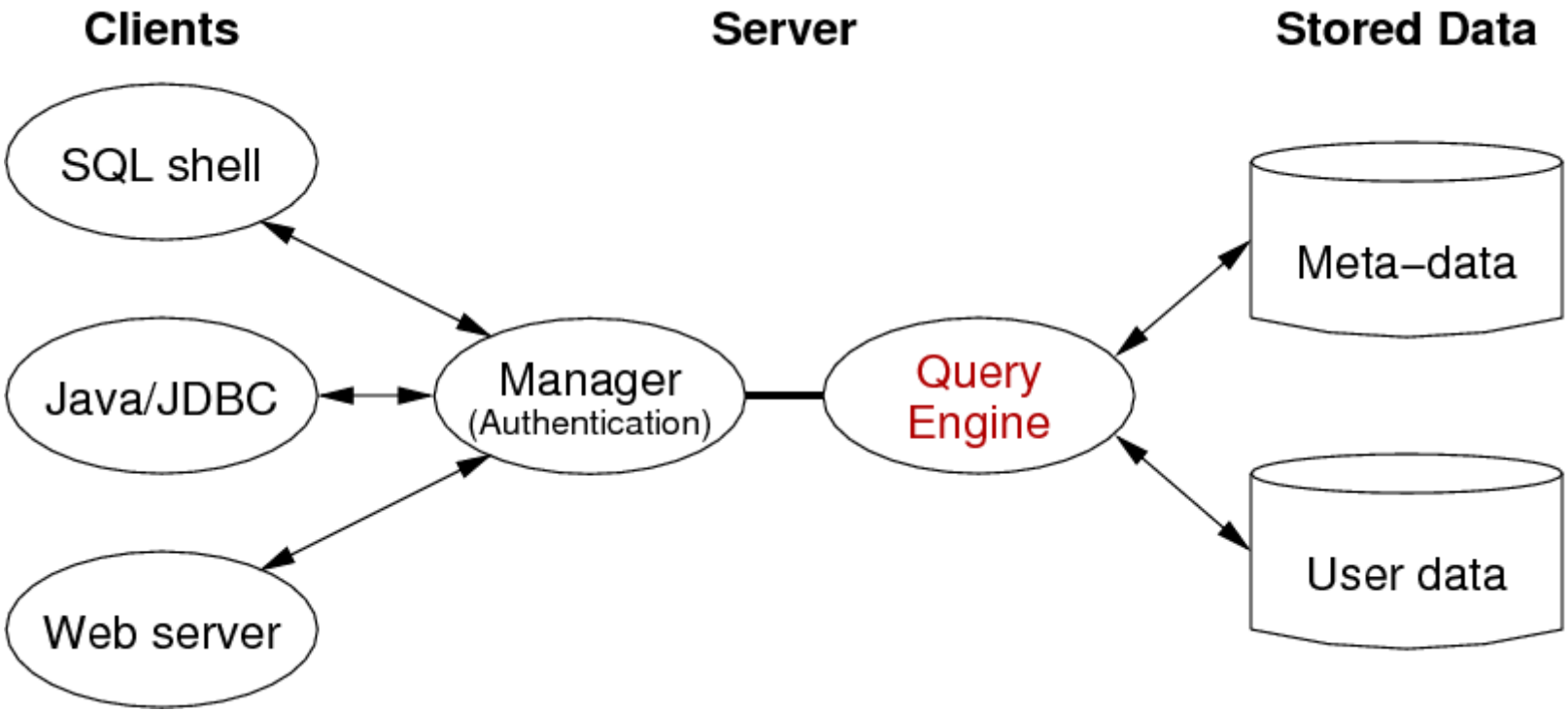
For serious applications, the RDBMS must be ACID ...

Atomicity	Either all operations of transaction are reflected in database or none are.
Consistency	Execution of a transaction in isolation preserves data consistency (i.e. maps a valid DB to a valid DB).
Isolation	Transactions are "unaware" of other transactions executing concurrently.
Durability	After successful transaction, changes persist even if system later fails.

DBMSs with these properties provide a strong guarantee that any update operation will result in a valid database (no corruption).

RDBMS Architecture

Typical client–server architecture for modern RDBMS:



Core of RDBMS = a relational algebra engine.

Examples of RDBMSs

Examples of ACID (serious) database management systems:

- Oracle (most widely-used commercial RDBMS)
- IBM's DB2 (maybe the best relational engine + research backing)
- Microsoft's SQL Server (solid relational engine + research backing)
- PostgreSQL (open-source, reliable, serious functionality)
- MySQL (open-source, now has sufficient functionality)
- SQLite (open-source, serverless, DB is a single file)
- HyperSQL (open-source, 100% Java, serious functionality)
- Informix, Sybase, Ingres, ... have all faded or are fading

Examples of RDBMSs (cont)

Most serious RDBMS's

- support all of the SQL:92 and SQL:99 standards
- support most of the SQL:2003 and SQL:2008 standards

Typical variations:

- mechanism for creating unique ID values (e.g. **`SEQUENCE`**)
- mechanism to return part of result set (e.g. **`LIMIT`**)
- extra data types (not defined in standard)
- many additional functions on data values
- don't support all SQL transaction levels
- **`DROP`** *Object* **`IF EXISTS`**

DB/Application Interaction

Database applications typically involve:

- code in an application language (e.g. Java, Python)
- definitions and queries in SQL (stored in DB)

- procedural code/transactions (stored in DB)

Code in multiple languages, stored in different places.

Interaction between app.code and DBMS

- is via a "thin pipe" (SQL in, tuples out)
- following old-fashioned file access pattern:

```
open; while (more input) { get next; process; } close
```

DB/Application Interaction (cont)

A typical DB application has several types of code:

- UI (user interface) code, e.g. Python/HTML, Java/Swing
- application/business logic, e.g. Python, Java, PL/SQL
- data access, e.g. PL/SQL, SQL

Multiple code sources ⇒ software management problems:

- maintaining consistency across code–bases
- minimising cross–talk (separation of concerns)

Patterns such as Model–View–Controller aim to manage this.

DB/Application Interaction (cont)

Perfomance issues with data–intensive applications

- minimise traffic between app.code and DB
  (connection/communication costs are expensive)
- "impedance mismatch" between app.code and DB
  (DB = tuple–set–at–a–time, Code = object–at–a–time)

These suggest that

- as much as possible, data manipulation is pushed into DBMS
- but this requires a tight interaction between layers

Software engineering suggests minimal interaction between layers.

This conflict is not yet well–resolved.

DB/Application Interaction (cont)

Standard paradigm for accessing DB from app.code

```
--  establish connection to DBMS
db = dbConnect("dbname=X user=Y passwd=Z");
query = "select a,b from R,S where ... ";
--  invoke query and get handle to result set
results = dbQuery(db, query);
--  for each tuple in result set
while (tuple = dbNext(results)) {
    --  process next tuple
    process(val(tuple,'a'), val(tuple,'b'));
}
dbClose(results);
```

Produced: 13 Sep 2020