Introduction to Databases & SQL

CSCE 156 - Introduction to Computer Science II

Christopher M. Bourke
`cbourke@cse.unl.edu`

Introduction to Databases & SQL

- Lifetime of a program is short-lived
- Applications perform small ephemeral operations
- Can crash and die
- Programs may be limited to sessions or even single requests
- Need a way to *persist* data or program state across program lives
- Databases provide such a means

Motivating Example I

Flat Files

Consider the following data, stored in a *flat file*:

Course	Course Name	Student	NUID	Email
		waits, tom	11223344	tomwaits@hotmail.com
CSCE 156	Intro to CSII	Lou Reed	11112222	reed@gmail.com
CSCE 156	Intro to CSII	Tom Waits	11223344	twaits@email.com
CSCE 230	Computer Hardware	Student, J.	12345678	jstudent@geocities.com
CSCE 156	Intro to CSII	Student, John	12345678	jstudent@geocities.com
CSCE 230	Computer Hardware	Student, J.	12345678	jstudent@geocities.com
CSCE 235	Discrete Math	Student, John	12345678	jstudent@geocities.com
CSCE 235	Discrete Math	Tom Waits	11223344	twaits@email.com
NONE	Null	Tom Waits	11223344	twaits@email.com

Table: Course enrollment data

Motivating Example II

Flat Files

Problems?

- Repetition of data
- Incomplete data
- Integrity of data
- Organizational problems: any aggregation requires processing all records
- Updating information is difficult (must enumerate all possible changes, side effects so that information is not lost)
- Formatting Issues
- Concurrency Issues

Relational Databases I

Key Aspects

Solution: Relational Database Systems (RDBS) or Relational Database Management System (RDMS)

- Stores data in *tables*
- Tables have a unique name and type description of its *fields* (integer, string)
- Each column stores a single piece of data (field)
- Each row represents a record (or object!)

Relational Databases II

Key Aspects

- Each row may have a unique *primary key* which may be
  - Automatically incremented
  - An external unique identifier: SSN, ISBN, NUID
  - Based on a combination of fields (Geographical data)
- Rows in different tables are related to each other through *foreign keys*
- Order of rows/columns is meaningless

Relational Databases III

Key Aspects

- Supports *Transactions*: an interaction or batch of interactions treated as one *unit*
- Constraints
  - Allowing or disallowing NULL
  - Disallowing "bad" values (ranges)
  - Enforcing formatting (capitalization, precision)
  - Limiting combinations of data fields

Relational Databases IV

Key Aspects

- ACID principles
  - **Atomicity** – Data modifictions must be an all-or-nothing process
  - Atomic operation: not divisible or decomposable
  - **Consistency** – Transactions will retain a state of *consistency*
  - All constraints, triggers, cascades preserve a valid state after the transaction has completed
  - **Isolation** – No transaction interferes with another
  - **Durability** – Once committed, a transaction remains so
  - Data is to be protected from catastrophic error (power loss/crash)

Commercial RDBMs

- MS Access ☹
- MySQL (owned by Oracle, released under GNU GPL)
- PostgreSQL (true FOSS!)
- Informix (IBM)
- DB2 (IBM)
- SQLServer (Microsoft)
- Oracle Database

Advantages I

- Data is *structured* instead of "just there"
- Better organization
- Duplication is minimized (with proper *normalization*)
- Updating information is easier
- Organization of data allows easy access
- Organization allows aggregation and more complex information

Advantages II

- Data integrity can be enforced (data types and user defined constraints)
- Faster
- Scalable
- Security
- Portability
- Concurrency

Structured Query Language

We interact with RDBMs using *Structured Query Language* (SQL)

- Common language/interface to most databases
- Developed by Chamberlin & Boyce, IBM 1974
- Implementations may violate standards: portability issues
- Comments: `--` or `#` (MySQL on cse)
- Create & manage tables: `CREATE`, `ALTER`, `DROP`
- Transactions: `START TRANSACTION`, `ROLLBACK`, `COMMIT`

Structured Query Language

CRUD

Basic SQL functionality: *CRUD*:

- Create – insert new records into existing tables
- Retrieve – get a (subset) of data from specific rows/columns
- Update – modify data in fields in specified rows
- Destroy – delete specific rows from table(s)

Misc RDMS Issues I

Important aspects that will be omitted (good advanced topics):

Views – RDBSs allow you to create view of data; predefined select statements that aggregate (or limit) data while appearing to be a separate table to the end user

Triggers – SQL routines that are executed upon predefined events (inserts/updates) in order to create side-effects on the database

Misc RDMS Issues II

Stored Procedures – SQL routines (scripts) that are available to the end user

Temp Tables – Temporary tables can be created to store intermediate values from a complex query

Nested Queries – SQL supports using subqueries to be used in other queries

MySQL

Getting Started

Useful MySQL commands to get you started:

- `USE dbdname;`
- `SHOW TABLES;`
- `DESCRIBE tablename;`

Warning: these are MySQL commands, not SQL!

Creating Tables

Syntax:

```
CREATE TABLE table_name (
  field_name fieldType [options],
  ...
  PRIMARY KEY (keys)
```

Options:

- `AUTO_INCREMENT` (for primary keys)
- `NOT NULL`
- `DEFAULT (value)`

Column Types

- `VARCHAR(n)` – variable character field (or `CHAR`, `NCHAR`, `NVCHAR` – fixed size character fields)
- `INTEGER` or `INT`
- `FLOAT` (`REAL`, `DOUBLE PRECISION`)
- `DECIMAL(n,m)`, `NUMERIC(n,m)`
- Date/Time functions: rarely portable
- MySQL: see `http://dev.mysql.com/doc/refman/5.0/en/date-and-time-functions.html`

Creating Tables

Example

```
CREATE TABLE book (
  id       INTEGER PRIMARY KEY AUTO_INCREMENT NOT
     NULL,
  title   VARCHAR(255) NOT NULL,
  author  VARCHAR(255),
  isbn    VARCHAR(255) NOT NULL DEFAULT '',
  dewey   FLOAT,
  num_copies INTEGER DEFAULT 0
);
```

Primary Keys

- Records (rows) need to be distinguishable
- A *primary key* allows us to give each record a unique identity
- At most one primary key per table
- Must be able to uniquely identify all records (not just those that exist)
- No two rows can have the same primary key value
- PKs can be one ore more columns–combination determines key
- Should not use/allow NULL values
- Can/should[1] be automatically generated

[1] How to handle the foreign key problem?

Keys

- Tables can have multiple keys
- May be a combination of columns
- NULL values are allowed
- Uniqueness is enforced (updates, inserts may fail)
- May be declared non-unique in which case it serves as an *index* (allows database lookup optimization)
- MySQL syntax:
  `KEY(column1, column2,...)`

Foreign Keys I

- Relations between records in different tables can be made with *foreign keys*
- A FK is a column that references a key (PK or regular key) in another table
- Inserts cannot occur if the referenced record does not exist
- Foreign Keys establish *relationships* between tables:
  - One-to-many relations
  - Many-to-one
  - Many-to-Many relations: requires a *Join Table*

Foreign Keys II

- Table with FK (referencing table) references table with PK (referenced table)
- Deleting rows in the referenced table can be made to *cascade* to the referencing records (which are deleted)
- Cascades can be evil
- MySQL Syntax:
  `FOREIGN KEY (column)REFERENCES table(column)`

Inserting Data

- Need a way to load data into a database
- Numerical literals
- String literals: use single quote characters
- Ordering of columns irrelevant
- MySQL Syntax:
  `INSERT INTO table_name (column1, column2, ...)VALUES (value1, value2);`
- Example:
  `INSERT INTO book (title, author, isbn)VALUES ('The Naked and the Dead', 'Normal Mailer', '978-0312265052');`

Updating Data

- Existing data can be changed using the `UPDATE` statement
- Should be used in conjunction with *clauses*
- Syntax:
  `UPDATE table SET column1 = value1, column2 = value2, ...`
  `WHERE (condition)`
- Example: `UPDATE book SET author = 'Norman Mailer'WHERE isbn = '978-0312265052';`

Deleting Data

- Data can be deleted using the `DELETE` statement
- Should be used in conjunction with *clauses*
- Unless you *really* want to delete *everything*
- Syntax:
  `DELETE FROM table WHERE (condition)`
- Example: `DELETE FROM book WHERE isbn = '978-0312265052';`

Querying Data

- Data can be retrieved using the `SELECT` statement
- Syntax:
  `SELECT column1, column2... FROM table WHERE (condition);`
- Example:
  `SELECT author, title FROM book WHERE isbn = '978-0312265052' ;`
- Can select *all* columns by using the * wildcard:
  `SELECT * FROM book`

Querying Data

Aliasing

- Names of the columns are part of the database
- SQL alllows us to "rename" them in result of our query using *aliasing*
- Syntax: `column_name AS column_alias`
- Sometimes necessary if column has no name (aggregates)

```
SELECT title      AS bookTitle,
       num_copies AS numberOfCopies
FROM book;
```

WHERE Clause

- Queries can be quantified using the `WHERE` clause
- Only records matching the condition will be affected (updated, deleted, selected)
- Compound conditions can be composed using parentheses and:
  - `AND`
  - `OR`

```
SELECT * FROM book WHERE num_copies > 10 AND (title
    != 'The Naked and the Dead' OR author = 'Dr.
    Seuss');
```

To check nullity: `WHERE dewey IS NULL`, `WHERE dewey IS NOT NULL`

LIKE Clause

- VARCHAR values can be searched/partially matched using the `LIKE` clause
- Used in conjunction with the string wildcard, `%`
- Example:
  `SELECT * FROM book WHERE isbn LIKE '123%';`
- Example:
  `SELECT * FROM book WHERE author LIKE '%Mailer%';`

IN Clause

- The `IN` clause allows you to do conditionals on a *set* of values
- Example:
```
SELECT * FROM book WHERE isbn in ('978-0312265052', '
789-65486548', '681-0654895052');
```

ORDER BY Clause

- In general, the order of the results of a `SELECT` clause is irrelevant
- Nondeterministic, not necessarily in any order
- To impose an order, you can use `ORDER BY`
- Can order along multiple columns
- Can order descending or ascending (`DESC`, `ASC`)
- Example:
```
SELECT * FROM book ORDER BY title;
```
- Example:
```
SELECT * FROM book ORDER BY author DESC, title ASC
```

Aggregate Functions

- Aggregate functions allow us to compute data on the database without processing all the data in code
- `COUNT` provides a mechanism to count the number of records
- Example:
```
SELECT COUNT(*)AS numberOfTitles FROM book;
```
- Aggregate functions: `MAX`, `MIN`, `AVG`, `SUM`
- Example:
```
SELECT MAX(num_copies)FROM book;
```
- NULL values are ignored/treated as zero

GROUP BY clause

- The `GROUP BY` clause allows you to project data with common values into a smaller set of rows
- Used in conjunction with aggregate functions to do more complicated aggregates
- Example: find total copies of all books by author:
```
SELECT author, SUM(num_copies)AS totalCopies FROM book GROUP
 BY author;
```
- The projected data can be further filtered using the `HAVING` clause:
```
SELECT author, SUM(num_copies)AS totalCopies FROM book GROUP
 BY author HAVING totalCopies > 5;
```
- `HAVING` clause evaluated *after* `GROUP BY` which is evaluated *after* any `WHERE` clause

JOINs

A *join* is a clause that combines records from two or more tables.

- Result is a set of columns/rows (a "table")
- Tables are joined by shared values in specified columns
- Common to join via Foreign Keys
- Table names can be aliased for convenience
- Types of joins we'll look at:
  - (INNER) JOIN
  - LEFT (OUTER) JOIN
- Other types of joins: Self-join, cross join (cartesian product), right outer joins, full outer joins

INNER JOIN I

- Most common type of join
- Combines rows of table A with rows of table B for all records that satisfy some predicate
- Predicate provided by the `ON` clause
- May omit `INNER`
- May provide join predicate in a `WHERE` clause

INNER JOIN II

```
SELECT * FROM book b
    INNER JOIN person p ON b.author = p.name

SELECT * FROM book b
    JOIN person p ON b.author = p.name

SELECT *
FROM book b, person p
WHERE b.author = p.name

SELECT * FROM student s
    JOIN email e ON s.student_id = e.student_id;
```

LEFT OUTER JOIN I

- Left Outer Join joins table A to table B, preserving all records in table A
- For records in A with no matching records in B: `NULL` values used for columns in B
- `OUTER` may be omitted

LEFT OUTER JOIN II

```
SELECT * FROM book b
    LEFT OUTER JOIN person p ON b.author = p.name

SELECT * FROM book b
    LEFT JOIN person p ON b.author = p.name

SELECT * FROM student s
    LEFT JOIN email e ON s.student_id = e.
       student_id;
```

DISTINCT Clause

- Many records may have the same column value
- May want to query only the unique values
- May only want to count up the number of unique values
- SQL keyword: `DISTINCT`

```
SELECT DISTINCT author FROM book;
SELECT COUNT(DISTINCT author) FROM book;
```

Good Practice Tip 1

Use consistent naming conventions

- Short, simple, descriptive names
- Avoid abbreviations, acronyms
- Use *consistent* styling
  - Table/field names: Lower case, underscores, singluar or
  - Camel case, pluralized
- Primary key field: `table_name_id`
- Use all upper-case for SQL commands
- Foreign key fields should match the fields they refer to
- End goal: unambiguous, consistent, self-documenting

Good Practice Tip 2

Ensure Good Data Integrity

Data can break code, code should not break data.

- Data/databases are a *service* to code
- Different code, different modules can access the same data
- The database does *not* use the code!
- Should do everything you can to prevent bad code from harming data (constraints, foreign & primary keys, etc).
- Database is your last line of defense against bad code

Good Practice Tip 3

Keep Business Logic Out!

- Databases offer "programming functionality"
- Triggers, cascades, stored procedures, etc.
- *Use them sparingly!!!*
- RDMSs are for the management and storage of data, not processing
- Demarcation of responsibility
- DBAs should not have to be Application Programmers, and vice versa

Current Trends I

- Additional Data-layer abstraction tools (JPA, ADO for .NET)
- RDBMs are usually tuned to either small, frequent read/writes or large batch read-transactions
- Nature and scale of newer applications does not fit well with traditional RDBMs
- Newer applications are data intensive:
  - Indexing a large number of documents
  - High-traffice websites
  - Large-scale delilvery of multimedia (streaming video, etc.)
  - Search applications, data mining
  - New tools generating HUGE amounts of data (biological, chemical, sensor networks, etc)

Current Trends II

- Newer (revived) trend: NoSQL
  - Non-relational data
  - Sacrivifes rigid ACID principles for performance
  - Eventual consistency
  - Limited transactions
  - Emphasis on read-performance
  - Simplified Key-Value data model
  - Simple interfaces (associative arrays)
- Example: Google's BigTable (key: two arbitrary string (keys) to row/column with a datetime, value: byte array)
- XML-based databases (using XQuery)

Designing A Database

Exercise

Exercise Design a database to support a course roster system. The database design should be able to model Students, Courses, and their relation (ability of students to *enroll* in courses). The system will also need to email students about updates in enrollment, so be sure your model is able to incorporate this functionality.

Designing A Database

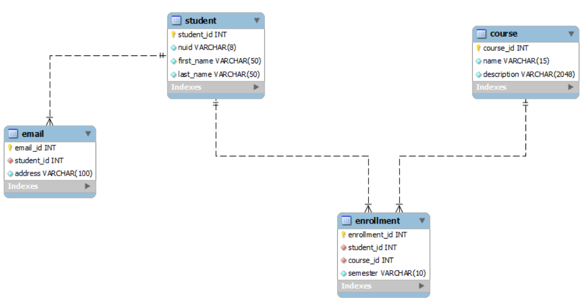

Figure: A normalized database design, ER diagram generated in MySQL Workbench

End Result

- Pieces of data are now organized and have a specific *type*
- No duplication of data
- Entities are represented by IDs, ensuring identity (Tom Waits is now the same as t. Waits)
- Data integrity is enforced (only one NUID per Student)
- Relations are well-defined
  - A student *has* email(s)
  - A course has student(s) and a student has course(s)

Data from flat file

student_id	first_name	last_name	nuid
1	Tom	Waits	11223344
2	Lou	Reed	11112222
5	John	Student	12345678

course_id	name	description
42123	CSCE 156	Computer Science II
12333	CSCE 230	Computer Hardware
11132	CSCE 235	Discrete Mathematics

email_id	student_id	address
8	2	reed@gmail.com
10	1	tomwaits@hotmail.com
13	1	twaits@email.com
14	5	jstudent@geocities.com

enrollment_id	student_id	course_id	semester
15	5	11132	Fall 2011
23	1	42123	Fall 2011
24	2	42123	Fall 2011
10	1	11132	Fall 2011
29	5	42123	Spring 2011
32	5	12333	Fall 2011