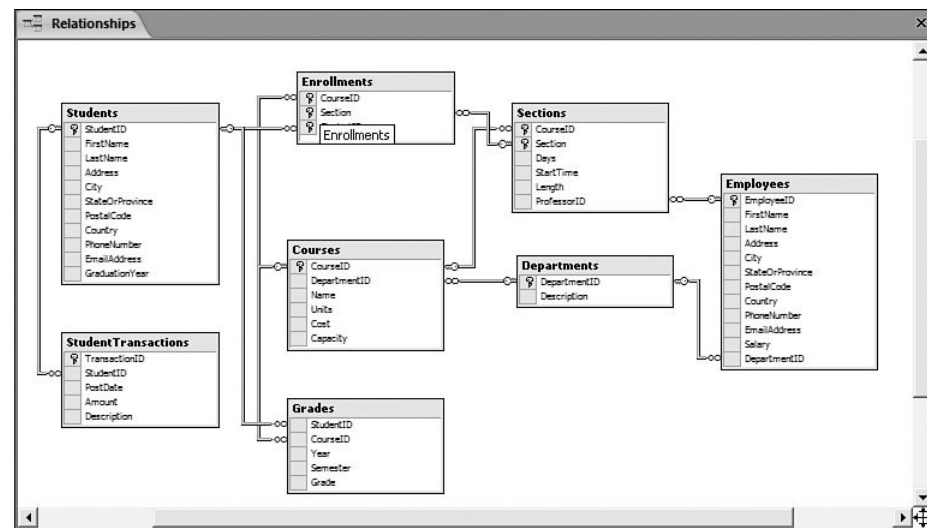


CS 193A

Databases and SQL

What is a database?

- **relational database:** A method of structuring data as **tables** associated to each other by shared attributes.
- a table **row** corresponds to a unit of data called a record; a **column** corresponds to an attribute of that record
- relational databases typically use Structured Query Language (**SQL**) to define, manage, and search data



Why use a database?

- **powerful**: can search, filter, combine data from many sources
- **fast**: can search/filter a database very quickly compared to a file
- **big**: scale well up to very large data sizes
- **safe**: built-in mechanisms for failure recovery (transactions)
- **multi-user**: concurrency features let many users view/edit data at same time
- **abstract**: layer of abstraction between stored data and app(s)
- **common syntax**: database programs use same SQL commands

Some database software

- **Oracle**
- **Microsoft**
 - **SQL Server** (powerful)
 - **Access** (simple)
- **PostgreSQL**
 - powerful/complex free open-source database system
- **SQLite**
 - transportable, lightweight free open-source database system
- **MySQL**
 - simple free open-source database system
 - many servers run "LAMP" (Linux, Apache, MySQL, and PHP)
 - Wikipedia is run on PHP and MySQL



Example database: school

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

students

id	name	teacher_id
10001	Computer Science 142	1234
10002	Computer Science 143	5678
10003	Computer Science 190M	9012
10004	Informatics 100	1234

courses

id	name
1234	Krabappel
5678	Hoover
9012	Stepp

teachers

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

grades

Example database: world

code	name	continent	independence_year	population	gnp	head_of_state	...
AFG	Afghanistan	Asia	1919	22720000	5976.0	Mohammad Omar	...
NLD	Netherlands	Europe	1581	15864000	371362.0	Beatrix	...
...

countries (Other columns: region, surface_area, life_expectancy, gnp_old, local_name, government_form, capital, code2)

id	name	country_code	district	population
3793	New York	USA	New York	8008278
1	Los Angeles	USA	California	3694820
...

cities

country_code	language	official	percentage
AFG	Pashto	T	52.4
NLD	Dutch	T	95.6
...

languages

Example database: imdb

id	first_name	last_name	gender
433259	William	Shatner	M
797926	Britney	Spears	F
831289	Sigourney	Weaver	F
...			

actors

movie_id	genre
209658	Comedy
313398	Action
313398	Sci-Fi
...	

movies_genres

id	name	year	rank
112290	Fight Club	1999	8.5
209658	Meet the Parents	2000	7
210511	Memento	2000	8.7
...			

movies

id	first_name	last_name
24758	David	Fincher
66965	Jay	Roach
72723	William	Shatner
...		

directors

director_id	movie_id
24758	112290
66965	209658
72723	313398
...	

movies_directors

actor_id	movie_id	role
433259	313398	Capt. James T. Kirk
433259	407323	Sgt. T.J. Hooker
797926	342189	Herself
...		

roles

SQL ([link](#))

```
SELECT name FROM cities WHERE id = 17;
```

```
INSERT INTO countries VALUES ('SLD', 'ENG', 'T', 100.0);
```

- **Structured Query Language (SQL):** a language for searching and updating a database
 - a standard syntax that is used by all database software (*with minor incompatibilities*)
 - generally case-insensitive
- a **declarative language:** describes what data you are seeking, not exactly how to find it

The SELECT statement

```
SELECT column(s) FROM table WHERE condition;
```

```
SELECT name, population FROM cities  
        WHERE country_code = "FSM";
```

- searches a database and returns a set of results
 - column name(s) after SELECT filter which parts of rows are returned
 - table and column names are **case-sensitive**
 - SELECT DISTINCT removes any duplicates
 - SELECT * keeps all columns
- WHERE clause filters out rows based on columns' data values
 - in large databases, WHERE clause is critical to reduce result set size

WHERE clauses

```
SELECT name, gnp FROM countries WHERE gnp > 2000000;
```

```
SELECT * FROM cities WHERE code = 'USA'  
AND population >= 2000000;
```

```
SELECT code, name, population FROM countries  
WHERE name LIKE 'United%';
```

-
- WHERE clause can use the following operators:

=, >, >=, <, <=

<> : not equal (some systems support !=)

BETWEEN *min* AND *max*

LIKE *pattern* (put % on ends to search for prefix/suffix/substring)

IN (*value*, *value*, ..., *value*)

condition1 AND *condition2* ; *condition1* OR *condition2*

ORDER BY, LIMIT

```
SELECT code, name, population FROM countries  
WHERE name LIKE 'United%' ORDER BY population;
```

```
SELECT * FROM countries ORDER BY population DESC, gnp;
```

```
SELECT name FROM cities WHERE name LIKE 'K%' LIMIT 5;
```

- ORDER BY sorts in ascending (default) or descending order
 - can specify multiple orderings in decreasing order of significance
- LIMIT gets first N results of the query
 - useful as a sanity check to make sure query doesn't return 10^7 rows

Related tables

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

students

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

grades

id	name	teacher_id
10001	Computer Science 142	1234
10002	Computer Science 143	5678
10003	Computer Science 190M	9012
10004	Informatics 100	1234

courses

id	name
1234	Krabappel
5678	Hoover
9012	Stepp

teachers

- **primary key:** column guaranteed to be unique for each row (ID)
- **normalizing:** splitting tables to improve structure / redundancy

JOIN

```
SELECT column(s) FROM table1 name1  
        JOIN table2 name2 ON condition(s)  
        ...  
        JOIN tableN nameN ON condition(s)  
WHERE condition;
```

```
SELECT name, course_id, grade  
FROM students s  
JOIN grades g ON s.id = g.student_id  
WHERE s.name = 'Bart';
```

-
- JOIN combines related records from two or more tables
 - ON clause specifies which records from each table are matched
 - rows are often linked by their key columns ('id')
 - joins can be tricky to understand; out of scope of this course

Create/delete a database; CRUD

```
CREATE DATABASE name;
```

```
DROP DATABASE name;
```

```
CREATE DATABASE warcraft;
```

- Must first create a database and add one or more tables to it.
- Most apps/sites do four general tasks with data in a database:
 - Create new rows
 - Read existing data
 - Update / modify values in existing rows
 - Delete rows

Creating tables

```
CREATE TABLE IF NOT EXISTS name (  
    columnName type constraints,  
    ...  
    columnName type constraints  
);  
DROP TABLE name;
```

```
CREATE TABLE students (  
    id INTEGER,  
    name VARCHAR(20),  
    email VARCHAR(32),  
    password VARCHAR(16)  
);
```

BOOLEAN	either TRUE or FALSE
INTEGER	32-bit integer
DOUBLE	real number
VARCHAR(<i>length</i>)	string up to given length
ENUM(<i>val</i> , ..., <i>val</i>)	a fixed set of values
DATE, TIME, DATETIME	timestamps (common value: NOW())
BLOB	binary data

-
- all columns' names and types must be listed (*see table above*)

Table column constraints

```
CREATE TABLE students (  
    id INTEGER UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(20) NOT NULL,  
    email VARCHAR(32),  
    password VARCHAR(16) NOT NULL DEFAULT "12345"  
);
```

-
- NOT NULL: empty value not allowed in any row for that column
 - PRIMARY KEY / UNIQUE: no two rows can have the same value
 - DEFAULT **value**: if no value is provided, use the given default
 - AUTO_INCREMENT: default value is the last row's value plus 1
 - (usually used for ID column)
 - UNSIGNED: don't allow negative numbers (INTEGER only)

INSERT and REPLACE

```
INSERT INTO table (columnName, ..., columnName)  
VALUES (value, value, ..., value);
```

```
REPLACE INTO table (columnName, ..., columnName)  
VALUES (value, value, ..., value);
```

```
INSERT INTO students (name, email)  
VALUES ("Lewis", "lewis@fox.com");
```

```
REPLACE INTO students (id, name, value)  
VALUES (789, "Martin", "prince@fox.com");
```

-
- some columns have default or automatic values (such as IDs)
 - omitting them from the INSERT statement uses the defaults
 - REPLACE is like INSERT but modifies an existing row

UPDATE

```
UPDATE table
SET column1 = value1,
    ...,
    columnN = valueN
WHERE condition;
```

```
UPDATE students
SET email = "lisasimpson@gmail.com"
WHERE id = 888;
```

-
- modifies an existing row(s) in a table
 - Be careful! If you omit WHERE clause, it modifies ALL rows

DELETE

```
DELETE FROM table  
WHERE condition;
```

```
DELETE FROM students  
WHERE id = 888;
```

-
- removes existing row(s) in a table
 - can be used with other syntax like LIMIT, LIKE, ORDER BY, etc.
 - Be careful! If you omit WHERE clause, it deletes ALL rows

Modifying existing tables

ALTER TABLE *name* RENAME TO *newName*;

ALTER TABLE *name*
ADD COLUMN *columnName* *type constraints*;

ALTER TABLE *name* DROP COLUMN *columnName*;

ALTER TABLE *name*
CHANGE COLUMN *oldColumnName* *newColumnName* *type constraints*;

- SQL has many commands for modifying existing data
 - the above is not a complete reference

Android SQLiteDatabase ([link](#))

```
SQLiteDatabase db = openOrCreateDatabase(  
    "name", MODE_PRIVATE, null);  
db.execSQL("SQL query");
```

- methods:
 - `db.beginTransaction(), db.endTransaction()`
 - `db.delete("table", "whereClause", args)`
 - `db.deleteDatabase(file)`
 - `db.insert("table", null, values)`
 - `db.query(...)`
 - `db.rawQuery("SQL query", args)`
 - `db.replace("table", null, values)`
 - `db.update("table", values, "whereClause", args)`

ContentValues ([link](#))

```
ContentValues cvalues = new ContentValues();  
cvalues.put("columnName1", value1);  
cvalues.put("columnName2", value2);  
...  
db.insert("tableName", null, cvalues);
```

-
- ContentValues can be optionally used as a level of abstraction for statements like INSERT, UPDATE, REPLACE
 - meant to allow you to use cleaner Java syntax rather than raw SQL syntax for some common operations. Contrast the above with:

```
db.execSQL("INSERT INTO tableName ("  
    + columnName1 + ", " + columnName2  
    + ") VALUES (" + value1 + ", " + value2 + ")");
```

Cursor ([link](#))

```
Cursor cursor = db.rawQuery("SELECT * FROM students");
cursor.moveToFirst();
do {
    int id = cursor.getInt(cursor.getColumnIndex("id"));
    String email = cursor.getString(
        cursor.getColumnIndex("email"));
    ...
} while (cursor.moveToNext());
cursor.close();
```

-
- Cursor lets you iterate through row results one at a time
 - `getBlob(index)`, `getColumnCount()`, `getColumnIndex(name)`,
`getColumnName(index)`, `getCount()`, `getDouble(index)`, `getFloat(index)`,
`getInt(index)`, `getLong(index)`, `getString(index)`, `moveToPrevious()`, ...

Dictionary app exercise

- Write an app that lets the user look up words in a dictionary.
 - The dictionary should be created as a SQLite **database**.
 - When the user types in a word, if that exact word exists in the dictionary, show its definition.
 - If the exact word does not exist in the dictionary, list all words of which the user's text is a substring.

