Referential Integrity and Relational Database Design

Outline

Correlated Updates - examples of UPDATEs only for SELECTed rows

Review: The Relational Model

- All data are represented as tables (relations)
- Tables are comprised of rows and columns (tuples)
- Rows are (officially) unordered (i.e., the order in which rows are referenced does not matter)
- A proper relational table contains **no duplicate rows.**
- Each table has a **primary key,** a unique identifier constructed from one or more columns
- Most primary keys are a single column (e.g., OWNERNUM for OWNERS)
- A table is linked to another by including the other table's primary key. Such an included column is called a **foreign key**

Review: Qualities of a Good Database Design

- Reflects real-world structure of the problem
- Can represent all expected data over time
- Avoids redundant storage of data items
- Provides efficient access to data
- Supports the maintenance of data integrity over time

- Clean, consistent, and easy to understand
- *Note: These objectives are sometimes contradictory!*

Review: Introduction to Entity-Relationship Modeling

- Entity-Relationship (E-R) Modeling: A method for designing databases
- A simplified version is presented here
- Represents the data by **entities** that have **attributes**.
- An entity is a class of distinct identifiable objects or concepts
- Entities have **relationships** with one another
- Result of the process is a **normalized** database that facilitates access and avoids duplicate data

Review: E-R Modeling Process

- Identify the entities that your database must represent
- Determine the cardinality relationships among the entities and classify them as one of
    - One-to-one (e.g., a parcel has one address)
    - One-to-many (e.g., a parcel may be involved in many fires)
    - Many-to-many (e.g., parcel sales: a parcel may be sold by many owners, and an individual owner may sell many parcels)
- Draw the entity-relationship diagram
- Determine the attributes of each entity
- Define the (unique) primary key of each entity

Review: From E-R Model to Database Design

- Entities with one-to-one relationships should be merged into a single entity
- Each remaining entity is modeled by a table with a primary key and attributes, some of which may be foreign keys
- One-to-many relationships are modeled by a foreign key attribute in the table representing entity on the "many" side of the relationship (e.g., the FIRES table has a foreign key that refers to the PARCELS table)
- Many-to-many relationships among two entities are modeled by a third table that has foreign keys that refer to the entities. These foreign keys should be included in the relationship table's primary key, if appropriate
- Commercially available tools can automate the process of converting a E-R model to a database schema

Database Design Example (from old exercise in 11.208)

Cambridge Fire Department Database

- Problem Statement*
- Proposed Solution*

Database Normalization

Much of formal database design is focused on **normalizing** the database and ensuring that design conforms to a **level of normalization** (e.g., first normal form, second normal form, etc.). Although there are higher normal forms, **Third Normal Form** is generally considered good enough for typical applications. Normalization generally involves taking a design with fewer tables and fewer columns and transforming it into a design with more tables with more columns -- after conducting some tests and applying some rules. A good reference on first through third normal forms, with nice examples, can be found here:

- David Faour's "Database Normalization" at swynk.com

Other Database Normalization References:

- Web Developer's Virtual Library
- Stuart Graduate School of Business
- Microsoft Product Support Services
- PHPBuilder.com
- About.com
- SQL Server Magazine
- Databasics at geekgirls.com

Review: Database Design Rules of Thumb

- Keep data items atomic (e.g., first and last names are separate). Concatenating columns together later on-the-fly is generally easy, but separating them is not. (First Normal Form)
    - *What is an example of where parsing subfields from a column may go awry?*
    - *When might you want to include the combined fields in a column anyway?*
- Define the primary key first. Use a descriptive name (PARCELID, not ID)
- In fact, use descriptive names that give a new user a decent chance of guessing what they mean for *all* your columns! (E.g., use PARCEL_COUNT rather than PACT)
- Use a single column for the primary key whenever possible; multi-column primary keys are appropriate for many-to-many relationships
- Use lookup tables rather than storing long values
- Use numeric keys whenever possible *(What about ZIP codes?)*
- Avoid intelligent keys (exception: lookup tables)

---

* Kindly refer to the Lecture Notes section

- Avoid using multiple columns to represent a one-to-many relationship (e.g., columns such as CHILD1, CHILD2 in a table called PARENT rather than putting the children in a separate table. (First Normal Form)
- For readability, use the primary key name for foreign keys unless the same foreign key is used multiple times in the same table (e.g., state of work and state of residence for a person might both be foreign keys that reference a table of states)
- Do not include two columns whose values are linked together (e.g., county name and county ID) unless one of the columns is the primary key of the table (Third Normal Form)
- Avoid allowing NULL values in columns that have a discrete range of possible values (e.g., integers between 1 and 10, inclusive)
    - *Not applicable to DBF files, which do not support NULLs*
- Avoid using multiple tables with similar structures that represent minor variants on the same entity (e.g., putting Boston parcels and Cambridge parcels in separate tables).
    - *Why is this rule often hard to practice with GIS?*
- Plan ahead for transferring data to a different database. For example, you may want to move data from Oracle to DBF, or Microsoft Access to Oracle.
    - Avoid column names with characters with other than UPPER CASE letters (A-Z), digits (0-9), and the underscore (_). Other characters may not be accepted by a database. Some database systems may be case sensitive with regard to column names, while others are not.
    - Keep your column names relatively short. Different databases support different numbers of characters in column names (e.g., 30 for Oracle, 64 for Microsoft Access, 10 for DBF). Try to make column names differ in the first few characters rather than at the end to avoid column name duplication if the names are truncated during the conversion process (e.g., use COL1 and COL2, not LONG_COLUMN_NAME_1 and LONG_COLUMN_NAME_2).
    - *Note that keeping column names short may be at odds with keeping your column names meaningful for neophytes. Be aware that you are making a tradeoff!*
- Remember that these are rules of thumb, *not* absolute laws! Bend the rules if you must but have a justification for your decision. The limitations of a GIS software package often provide a good reason.

Enforcing Referential Integrity in Oracle

Example: The Parcels Database*

- **Tables and Primary Keys**

* Kindly refer to the Lecture Notes section

Table	Primary Key
PARCELS	PARCELID
OWNERS	OWNERNUM
FIRES	PARCELID, FDATE
TAX	PARCELID

- **Cardinality Relationships**

Primary Table Columns	Foreign Table Columns	Cardinality
OWNERS.OWNERNUM	PARCELS.ONUM	One-to-many
PARCELS.PARCELID	FIRES.PID, FIRES.WPB	One-to-many
PARCELS.PARCELID	TAX.PARCELID	One-to-one

Parcels Database Enhancements

- Define a single-column primary key for the FIRES table (FIRES.FIREID)
- Merge the TAX and PARCEL tables *or* add the year to the tax table to keep track of taxes over time (changes the relationship to one-to-many)
- Rename PARCELS foreign key ONUM to be consistent with the OWNERS table
- Improve column names
