Introduction to Database Systems
CSE 444
Lecture #9
Jan 29 2001

Announcements
- Mid Term on Monday (in class)
  - Material in lectures
  - Textbook
    - Chapter 1.1, Chapter 2 (except 2.1 and ODL), Chapter 3 (except 3.2, 3.8), Chapter 4.1, 4.5, 4.6, Chapter 5 (except 5.10), Chapter 6.1, 6.2, 7.1, 7.3
- Mid Term will be in class closed book exam
- Extra Office Hours
  - Surajit (Today) 4.50-5.50
  - Yana Thu 4.30-5.30
- Solution to HW#1 available

Decomposition: Schema Design using FD

Review: Closure, Key, Superkey

Given a set of attributes $M$ over $R(A)$, and a set of Fds on $R$, closure($M$) is the set of all attributes $L$ such that $M \rightarrow L$

If Closure($M$) = $A$, then $M$ is a superkey

$M$ is also a key if no proper subset $M'$ of $M$ satisfies closure($M'$)=$A$

Superkey: A set of attributes containing key

Review: BCNF

A relation $R(A)$ is in BCNF if for every nontrivial dependency $X \rightarrow Y$ on the relation $R$, $X$ is a superkey

Every 2-column relation is in BCNF. Why?

Relation in BCNF does not have update or deletion anomalies

If relation $R(A)$ violates BCNF, decomposition is needed

How to find a FD that violates BCNF?

Check Closure($X$) of every FD $X \rightarrow Y$ in the given set of dependency

Decomposition Requires Care

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera</td>
<td>Gadget</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>DoubleClick</td>
<td>Camera</td>
<td>29.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

When we put it back:

Cannot recover information
Decomposition Strategy for BCNF

Find a FD that violates the BCNF condition (RHS = all nontrivial attributes functionality determined by LHS):

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

\[ R1 \hspace{0.5cm} \text{Others} \hspace{0.5cm} R2 \]

Example

\( R(A, B, C, D, E) \) decomposed into \( S(A, B, C) \) and ..
\( \) FD on \( R \): \( A \rightarrow B, B \rightarrow E, DE \rightarrow C \)
\( \) Closure(\( A \)) = ?
\( \) Closure(\( B \)) = ?
\( \) Closure(\( C \)) = ?
\( \) Closure(\( \{A, B\} \)) = ?

Projecting FD

\( \) Given \( F \) over \( R \), what is the FD that must hold over \( R' \), where \( R' \) is obtained by decomposition?
\( \) Compute closure(\( X \)) for each subset \( X \) of \( R' \)
\( \) \( X \rightarrow B \) holds in \( S \) if
\( \) \( B \) in \( R' \)
\( \) \( B \) in closure(\( X \))
\( \) \( B \) not in \( X \)
\( \) See Examples 3.39 and 3.40 in text

Example: Projecting FD

\( \) \( R(A, B, C, D, E) \) decomposed into \( S(A, B, C) \) and ..
\( \) FD on \( R \): \( A \rightarrow B, B \rightarrow E, DE \rightarrow C \)
\( \) Closure(\( A \)) = ?
\( \) Closure(\( B \)) = ?
\( \) Closure(\( C \)) = ?
\( \) Closure(\( \{A, B\} \)) = ?

Decomposition Based on BCNF is Information Preserving

Attributes \( A, B, C \)
FD: \( A \rightarrow C \)
Relations \( R1[A, B] \hspace{0.5cm} R2[A, C] \)
Tuples in \( R1 \): \( (a, b), (a, b') \)
Tuples in \( R2 \): \( (a, c), (a, c') \)
Tuples in the join of \( R1, R2 \): \( (a, b, c), (a, b', c), (a, b', c') \)
Can \( (a, b, c') \) be a bogus tuple? What about \( (a, b', c') \)?

Decomposition into BCNF is Not Dependency Preserving

\( \) \( \text{Street, city} \rightarrow \text{zip}, \text{zip} \rightarrow \text{city} \)
\( \) \( \text{Key:} \ (\text{street, city}), (\text{street, zip}) \)
\( \) Consider \( (\text{street, zip}) \) and \( (\text{zip, city}) \)
\( \) How to check \( \text{street, city} \rightarrow \text{zip} \)?
\( \) Not dependency preserving!
\( \) 3NF
\( \) Allow FD if LHS is part of a key (prime)
Problems with Decompositions

- Some queries become more expensive. (e.g., find employee and department names)
- Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
- Checking some dependencies may require joining the instances of the decomposed relations.

Tradeoff: Must consider these issues vs. redundancy.

Summary of Schema Refinement

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations:
  - Lossless-join decomposition into BCNF is always possible
  - Lossless-join, dependency preserving decomposition into BCNF is not always possible
  - Lossless-join, dependency preserving decomposition into 3NF is always possible
- Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.
- Various decompositions of a single schema are possible.

Constraints and Triggers

Reading: Section 6
(MidTerm: 6.1 and 6.2 only)

Constraints

- A constraint = an assertion about the database that must be true at all times
- Part of the database schema
- Correspond to invariants in programming languages

SQL for Keys and Reference Keys

```
CREATE TABLE Books (
    isbn CHAR(11),
    title CHAR(20),
    pubname CHAR(25),
    pubdate DATE,
    PRIMARY KEY (isbn),
    FOREIGN KEY (pubname) REFERENCES Publishers (name))
```
### Declaring Keys and Foreign Keys

- **Composite Key Syntax**
  - Primary Key (col1, col2)
  - Unique (col3)
- **Foreign Key Syntax**
  - Foreign Key <attributes> REFERENCES <table> (<attributes>)
  - Non-NULL value in Foreign Key must be present in the reference table

### Enforcing Constraints

- **Key constraint**
  - Check on update/insert
  - Use indexes for efficient validation
- **Referential constraint**
  - Default: Reject modifications that violate constraint
  - Cascade: Delete referencing rows
  - Delete movie => movie_stars deleted
  - Set Null: Set referencing column value to NULL

### Example

```
CREATE TABLE Studio (...
  presC# INT REFERENCES MovieExec(cert#)  
  ON DELETE SET NULL
  ON UPDATE CASCADE)
```

### CHECK Constraint

- **CHECK(search-condition)**
- Like Where clause in Selection queries
- Value-based check
  - CHECK (movie_type IN ('Horror', 'Thriller',...))
- Simple check
  - CHECK (cost < 100 and cost > 0)
- Use to verify min/max/set of intervals
- Complex
  - CHECK (cost < (select max(price) from Walmart_Store))

### Assertions

- Not attached to table declaration
- Specifies a multi-table constraint
- CREATE ASSERTION max_inventory
  - CHECK ((SELECT SUM(movie_cost) From Movies) + (SELECT SUM(music_cost) From Music) < 1000))
- Database must satisfy assertions at all times
  - Tuple constraint enforced only when table is not empty

### Deferrable Constraints

- By default, constraints are checked at the end of each SQL statement
- A DEFERRABLE constraint is checked only when the transaction is committed
**TRIGGERS**

★ Tells what followup actions to take after execution of a SQL.
★ CREATE TRIGGER NetWorthTrigger
  ▢ AFTER UPDATE of networth ON MovieExec
  ▢ REFERENCING OLD AS ot NEW AS nt
  ▢ WHEN (ot.NetWorth > nt.NetWorth)
  ▢ UPDATE MovieExec
  ▢ SET NetWorth = ot. Networth
  ▢ WHERE ...
  ▢ FOR EACH ROW .. Tuple vs. statement granularity

---

**Privileges, Users, Security**

Reading: Chapter 7.4

---

**Granularity of AC**

★ GRANT privilege_list
★ ON object
★ TO user_list [WITH GRANT OPTION]
★ Privilege_list
  □ Select, Insert, Delete, Update, References, Usage
★ Object
  □ Table, Columns, Views, Domains, Transactions..

---

**Examples**

★ GRANT SELECT ON movie_titles TO PUBLIC
★ GRANT REFERENCES (title) ON movie_titles TO USER1
★ GRANT SELECT ON movie to kirk
★ WITH GRANT OPTION
★ GRANT SELECT ON movie to Rob

---

**REVOKE**

★ REVOKE <privilege list> ON <database element> FROM <user list>
  □ CASCADE: All privileges granted based on revoked privileges are withdrawn
  □ RESTRICT: Allows execution of REVOKE only if there is no implied CASCADE
★ REVOKE GRANT OPTION FOR ....
★ Follow examples 7.24-7.26

---

**Concurrency Control I:**

Transactions, Schedules, Anomalies
**Why Have Concurrent Processes?**

- Better throughput, response time
- Done via better utilization of resources:
  - While one process is doing a disk read, another can be using the CPU or reading another disk.
- DANGER DANGER! Concurrency could lead to incorrectness!
  - Must carefully manage concurrent data access.
  - There's (much!) more here than the usual OS tricks!

**Transactions**

- Basic concurrency/recovery concept: a transaction (Xact).
  - A sequence of many actions which are considered to be one atomic unit of work.
- DBMS “actions”:
  - (disk) reads, (disk) writes
  - Special actions: commit, abort

**The ACID Properties**

- Atomicity: All actions in the Xact happen, or none happen.
- Consistency: If each Xact is consistent, and the DB starts consistent, it ends up consistent.
- Isolation: Execution of one Xact is isolated from that of other Xacts.
- Durability: If a Xact commits, its effects persist.

**Passing the ACID Test**

- Concurrency Control
  - Guarantees Consistency and Isolation, given Atomicity.
- Logging and Recovery
  - Guarantees Atomicity and Durability.
- We'll do C. C. first:
  - What problems could arise?
  - What is acceptable behavior?
  - How do we guarantee acceptable behavior?

**Schedules**

- Schedule: An interleaving of actions from a set of Xacts, where the actions of any 1 Xact are in the original order.
  - Represents some actual sequence of database actions.
  - Example: R(A), W(A), R(B), W(B), R(C), W(C)
  - In a complete schedule, each Xact ends in commit or abort.
  - Initial State + Schedule → Final State

**Acceptable Schedules**

- One sensible “isolated, consistent” schedule:
  - Run Xacts one at a time, in a series.
  - This is called a serial schedule.
  - NOTE: Different serial schedules can have different final states; all are “OK” -- DBMS makes no guarantees about the order in which concurrently submitted Xacts are executed.
- Serializable schedules:
  - Final state is what some serial schedule would have produced.
  - Aborted Xacts are not part of schedule; ignore them for now (they are made to ‘disappear’ by using logging).
Transactions: Serializability

Serializability Violations

Two actions may conflict when 2 xacts access the same item:
- W-R conflict: T2 reads something
  T1 wrote; T1 still active
- R-W and W-W conflicts: Similar.

WR conflict (dirty read):
- Result is not equal to any serial execution!
- T2 reads what T1 wrote, but it shouldn't have!!

More Conflicts

RW Conflicts (Unrepeatable Read)
- T2 overwrites what T1 read.
  T1: R(A), R(A), C
  T2: R(A), W(A), C
  Again, not equivalent to a serial execution.

WW Conflicts (Lost Update)
- T2 overwrites what T1 wrote.
  T1: W(A), W(A), C
  T2: W(A), W(B), C
  Usually occurs with RW or WR anomalies.
  Unless you have "blind writes" (as here).

Now, Aborted Transactions

Serializable schedule: Equivalent to a serial schedule of committed xacts.
As if aborted xacts never happened.

Two Issues:
- How does one undo the effects of a xact?
- We'll cover this in logging/recovery
- What if another Xact sees these effects??
- Must undo that Xact as well?

Cascading Aborts

Abort of T1 requires abort of T2!
- Cascading Abort

What about WW conflicts & aborts?
- T2 overwrites a value that T1 writes.
- T1 aborts: its "remembered" values are restored.
- Lose T2's write! We will see how to solve this, too.

An ACA (avoids cascading abort)
schedule is one in which cascading abort cannot arise:
- A Xact only reads data from committed Xacts.

Recoverable Schedules

Abort of T1 requires abort of T2!
- But T2 has already committed!

A recoverable schedule is one in which this cannot happen.
- i.e., a Xact commits only after all the Xacts it reads from commit.
- ACA implies Recoverable (but not vice-versa!).
- Real systems typically ensure that only recoverable schedules arise (through locking).
**COMMIT** and **ROLLBACK**

Can end a database operation in two ways:
- `EXEC SQL COMMIT;`
- `EXEC SQL ROLLBACK;`