Data Definition Language (DDL), Views and Indexes

Instructor: Shel Finkelstein

Reference:
A First Course in Database Systems,
3rd edition, Chapter 2.3 and 8.1-8.4

Important Notices

- Lab3 assignment is due on **Sunday, May 19**, by 11:59pm.
  - 3 weeks to do Lab3 because of Midterm.
  - Lab2 was/will be discussed at Lab Sections.
  - Your solution should be submitted via Canvas as a zip file.
    - Canvas is used for both Lab submission and grading.
    - Late Lab Assignments will not be accepted.
    - Be sure that you post the correct file!
  - Load file for Lab3 has been posted to Piazza.
    - You must use load file to do Lab3.
    - Helps with testing, but we won’t post query solutions.
- The **third** Gradiance Assignment, "CMPS 182 Spring 2019 #3", was/will be posted on Monday, April 29, and is due **Monday, May 6**, by 11:59pm.

Important Notices

- **Reminder:** Midterm is on **Wednesday, May 8.**
  - **Includes material up to and including previous Lecture**, but not this Lecture.
  - No make-ups, no early Midterms, no late Midterms … and no devices.
  - You may bring a **single two-sided 8.5” x 11” sheet of paper** with as much info written (or printed) on it as you can fit and read unassisted.
    - No sharing of these sheets will be permitted.
  - “Practice Midterm” from Spring 2017 has beenposted on Piazza under Resources→Exams.
    - Solution will be posted there next week … but take it yourself first.
    - Some questions are on topics we haven’t covered yet.
  - Hope that all requests for DRC accommodation have been submitted.
  - Piazza announcement will describe required seating pattern for Midterm.
- See Small Group Tutoring website for LSS Tutoring with Chandler Hawkins.

Important Notices Midterm

- **Reminder:** Midterm is on **Monday, Feb 11.**
  - **Includes material up to and including Lecture 6**, but not this Lecture, Lecture 7.
  - No make-ups, no early Midterms, no late Midterms … and no devices.
  - You may bring a **single two-sided 8.5" x 11" sheet of paper** with as much info written (or printed) on it as you can fit and read unassisted.
    - No sharing of these sheets will be permitted.
  - "Practice Midterm" from Fall 2018 has been posted on Piazza.
    - Solution will be posted on Friday, February 8 … but take it yourself first.
  - Hope that requests for DRC accommodation have all been submitted.
    - *You should have already received notification of DRC room if you submitted your DRC request!*
  - Piazza announcement describes required seating pattern for Midterm.
    - You'll be assigned alternating seats with one space between them as you enter the room, and then gaps will be filled as assigned by the TAs.

SQL Language

- Data Manipulation Language (DML)
  - Access and modify data
  - SELECT, INSERT, DELETE, UPDATE
- Data Definition Language (DDL)
  - Modify structure of data
  - CREATE, DROP, ALTER
- Data Control Language (DCL)
  - Control access to the data (security)
  - GRANT, REVOKE
- Databases also have Utilities, such as Backup/Restore
  - Syntax not specified in the SQL standard

CREATE TABLE

```
CREATE TABLE MovieStar (
    name            CHAR(30) ,
    address         VARCHAR(255)  DEFAULT ‘Hollywood’,
    gender          CHAR(1),
    birthdate       DATE NOT NULL DEFAULT ‘2001-12-30’,
    PRIMARY KEY (name)
    );
```

- PRIMARY KEY
- DEFAULT
- NOT NULL

Reminder: Some Facts About Nulls

- Almost all comparisons with NULL will evaluate to unknown. If Salary is NULL, then the following will be UNKNOWN:
  - Salary = 10
  - Salary <> 10
  - 90 > Salary OR 90 <= Salary
  - Salary = NULL
  - Salary <> NULL

- Use of IS NULL and IS NOT NULL
  - Salary IS NULL will be true if Salary is NULL, false otherwise
  - Salary IS NOT NULL will be true if Salary isn't NULL, false otherwise

- ORDER BY works with attributes that can have NULL values
  - NULL will probably be smallest or largest value
  - Not specified by SQL standard, so it depends on the implementation

- GROUP BY also works with attributes that can have NULL values

DROP TABLE

- Dropping a table:

```
DROP TABLE MovieStar;
```

- Don’t assume that rolling back transaction will bring back the table!
  - Interaction of DDL and transactions may depend on implementation.

ALTER TABLE

- Adding a column to a table:
  - ALTER TABLE MovieStar ADD phone CHAR(16) DEFAULT ‘unlisted’;

- Dropping a column from a table:
  - ALTER TABLE MovieStar DROP birthdate;
  - In some systems:
    ALTER TABLE MovieStar DROP **COLUMN** birthdate;
  - In some SQL systems, dropping a column isn’t allowed.

- Changing the type of a column:
  - Some implementations let you change type of column in limited ways.

What Can You CREATE/DROP in SQL DDL?

- TABLE
- **VIEW**
- **INDEX**
- ASSERTION
- TRIGGER
- SCHEMA
- PROCEDURE/FUNCTION/TYPE
  - SQL2003 standard, but there are significant variations in implementations in different systems
- …

VIEWS: Motivation for Views

- Views help with logical data independence, allowing you to retrieve data if it matches the description in the view.

```
CREATE VIEW < view-name> AS < view-definition> ;
```

```
CREATE VIEW ParamountMovies AS
        SELECT title, year
        FROM Movies
        WHERE studioName = ‘Paramount ' ;
```

- You may now ask queries on ParamountMovies as if it were a table:

```
SELECT title FROM ParamountMovies WHERE year=1976;
```

  - Composition in SQL is powerful: Tables, Queries, Views

Some Advantages of Views

- Short-hand/encapsulation: You can treat a view as a table (for queries), which allows you to define a short-hand for a concept that might involved a complicated query.
- Re-use: You can re-use a view as often as you like; other people who can access the view may also re-use it.
- Authorization: People may be granted access to a view, even though they don't have access to the underlying tables.
  - They may not even know that the view isn't a table.
  - Even if they know that it's a view, they may not know which tables underlie it.
- Logical data independence: Even if the tables underlying a view change, the view may still be used in queries (or applications), without re-writing the queries (or applications).

More Views

Movies (title , year , length , genre , studioName , producerC#)

MovieExec (name , address , cert# , netWorth)

```
CREATE VIEW MovieProd AS
       SELECT m.title, e.name, m.genre
       FROM Movies m, MovieExec e
        WHERE m.producerC# = e.cert# ;

      SELECT DISTINCT genre
       FROM MovieProd
       WHERE name = ‘George Lucas’;
```

Renaming Attributes in CREATE VIEW

Movies (title , year , length , genre , studioName , producerC#)

MovieExec (name , address , cert# , netWorth)

```
CREATE VIEW MovieProd(movie_title, prod_name, movie_genre) AS
       SELECT m.title, e.name, m.genre
       FROM Movies m , MovieExec e
        WHERE m.producerC# = e.cert# ;

      SELECT DISTINCT movie_genre
       FROM MovieProd
       WHERE prod_name = ‘George Lucas’;
```

What is a View?

- A view can include any SQL SELECT statement
  - Including UNION, Aggregates, GROUP BY, HAVING, ORDER BY, etc.
- A view is not stored as a table
  - The tables underlying the view are stored in the database, but only the description of the view is in the database
  - … although some systems support MATERIALIZED VIEWS
- But a view can be used in many (not all) of the same ways as tables
  - Views can be queried
  - Views can be defined on views, as well as on tables!

Queries on Views and Tables

```
CREATE VIEW ParamountMovies AS
        SELECT title , year
        FROM Movies
        WHERE studioName = ’Paramount ' ;

SELECT DISTINCT s.starName
FROM ParamountMovies p , StarsIn s
WHERE p.title = s.movieTitle AND p.year = s.movieYear ;

CREATE VIEW ParamountStars AS
       SELECT DISTINCT starName
       FROM ParamountMovies , StarsIn
       WHERE title = movieTitle AND year = movieYear ;
```

DROP VIEW

```
CREATE VIEW ParamountMovies AS
        SELECT title , year
        FROM Movies
        WHERE studioName = ’Paramount ' ;

DROP View ParamountMovies;
```

- What happens if you execute the following after dropping that view?
  - SELECT * FROM ParamountMovies;
  - SELECT * FROM Movies;

View Updates

- Some modification operations on views work, but others do not, generally failing either because:
  - Constraint on underlying table would be violated, or
  - The effects of the View modification is not well-defined on the underlying tables.

- This is a complex topic, which we’ll only discuss briefly.
  - See Textbook Section 8.2 for more info.

View Update Problems

Movies(title, year, length, genre, studioName , producerC#)

```
CREATE VIEW ParamountMovies AS
    SELECT title , year
    FROM Movies
    WHERE studioName = ‘Paramount ' ;
```

```
INSERT INTO ParamountMovies VALUES (‘StarTrek’, 1979);
```

The INSERT will fail if the other columns of Movies (besides title and year) don’t have defaults, and also don’t allow NULL values.

View Update Problems (continued)

Ambiguous View Update example with Employees and Departments

<< We’ll draw this on the board >>

INDEXES: Motivation for Indexes

- Searching an entire table may take a long time:

```
SELECT *
FROM Movies
WHERE studioName = ‘Disney’ AND year = 1990;
```

If there were 100 Million movies, searching them might take a while. An index (e.g., a B-Tree) would allow faster access to matching movies.

If a table is updated, all Indexes on that table are immediately automatically updated within the same transaction.

- Which indexes do you need to change on INSERT and DELETE?
- What about UPDATE?

CREATE INDEX

```
SELECT *
FROM Movies
WHERE studioName = ‘Disney’ AND year = 1990;
```

How much would each of these indexes help?

```
CREATE INDEX YearIndex ON Movies(year);

CREATE INDEX StudioIndex ON Movies(studioName);

CREATE INDEX YSIndex ON Movies(year,studioName);

CREATE INDEX SYIndex ON Movies(studioName,year);
```

How much would each of the indexes help if the WHERE clause was just year = 1990?

Indexes and Ordering

```
SELECT *
FROM Movies
WHERE studioName = ‘Disney’ AND year < 1990;
```

How much would each of these indexes help?

```
CREATE INDEX YearIndex ON Movies(year);

CREATE INDEX StudioIndex ON Movies(studioName);

CREATE INDEX YSIndex ON Movies(year,studioName);

CREATE INDEX SYIndex ON Movies(studioName,year);
```

How much would each of the indexes help if the WHERE clause was just year < 1990?

Indexes and Physical Independence

- SQL statements can be executed regardless of which indexes (if any) exist in the database.
  - Applications don’t have to be modified when indexes are created or dropped!
- What gets impacted when indexes are created or dropped?
  - Performance of SQL statements
  - Some may run faster, some may run slower

Disadvantages of Indexes?

- Why not put indexes on every attributes, or even on every combination of attributes that you might query on?
  - Huge number of indexes
  - Space for indexes
  - Cache impact of searching indexes
  - Update time for indexes when table is modified

Index Design

- Most Database Administrators (DBAs) pick a set of indexes that work well on expected workload, and there are tools that help pick good indexes
  - But workloads change, so choice of indexes may need to change
    - DROP INDEX YearIndex;

- Keys are indexed (automatically in many database systems) to:
  - Help maintain uniqueness (primary key, unique)
  - Check Foreign Key references to Primary Keys (Referential Integrity)

Index Utilization

- SQL statements don’t specify use of indexes, so they don’t have to be modified when you change what’s indexed!
  - Database Optimizer tries to figure out ”the best”/”a good” way to execute each SQL query.
  - All the tuples in a Relation can be scanned directly, without using indexes, so indexes aren’t necessary … except for performance.
  - Some systems have ways that you can tell the Optimizer what to do. This has advantages and disadvantages. (What are they?)

- Many SQL systems (including PostgreSQL) have an EXPLAIN PLAN statement, so that you can see what plan the optimizer chooses for a SQL statement.