Lecture 7
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

- Lecture slides and other class information will be posted on Piazza (not Canvas).
  - Slides are posted under Resources→Lectures
  - Lecture Capture recordings are available to all students under Yuja.
    - That includes classes given over Zoom.
  - There's no Lecture Capture for Lab Sections.
- All Lab Sections (and Office Hours) normally meet In-Person, with rare exceptions announced on Piazza.
  - Some slides for Lab Sections have been posted on Piazza under Resources→Lab Section Notes
- Office Hours for TAs and me are posted in Syllabus and Lecture1, as well as in Piazza notice @7; that post also now includes hours for Group Tutors.
- Some suggestions about use (and non-use) of Generative AI systems such as ChatGPT were posted on Piazza in @41, with specific discussion about the Movies tables and the four Avatar queries in Lecture 4.

Important Notices

- Lab2 was due **Wednesday, February 7**, by 11:59pm.
  - Lab2 Solution was posted on Piazza under Resources→Lab2 on Thursday, February 8.
  - A correction to solution for query2 was posted on Monday, Feb 12 in @83, and our solution was fixed.

- Lab3 will be posted on Piazza on Wednesday, Feb 14, the day of the Midterm.
  - Lab3 is due on **Tuesday, February 27** by 11:59pm.
  - Lab3 has not been posted yet because:
    - a) Midterm is on Wednesday, Feb 14, and …
    - b) There are several Lab3 topics that we haven't discussed yet.

- **There will be Lab Sections before the Midterm, on topics including:**
  - Lectures, Lab2 solution, last quarter's Midterm, Gradiance assignments
  - **There will also be Lab Sections after the Midterm** on topics including Lab3.

Important Notices

- Second Gradiance Assignment , CSE 180 Winter 2024 #2, was due on Tuesday, February 6, by 11:59pm.
- Third Gradiance Assignment, CSE 180 Winter 2024 #3, was assigned on Tuesday, February 6.
    - It is due on Thursday, February 15, by 11:59pm, the day after the Midterm.
        - But it's a good idea to complete this Gradiance Assignment before the Midterm.
        - The day of the week on which Gradiance Assignments are due will vary.
    - There are 9 questions in this Gradiance Assignment.
        - Some of them may be difficult.
    - Website and Class Token for Gradiance are in the Syllabus and first Lecture.
        - If you forget your Gradiance password, you can recover it by providing your Gradiance login and email.
    - You may take each Gradiance Assignment as many times as you like.
        - Questions stay the same, but answers change.
        - Your Gradiance Assignment score is the score on your **Final** Gradiance submission before the deadline.
        - Gradiance does not allow late submissions, so no extensions are possible.

Very Important Notice

- The Winter 2024 CSE 180 Final is on Wednesday, March 20 from 8:00am – 11:00am in our usual classroom.
  - The Syllabus and First Lecture originally said that it was on Tuesday, March 19.
  - That's been fixed. Please adjust your calendars accordingly!

Important Notices: Midterm

CSE 180 Midterm is on **Wednesday, February 14,** **and it will be given in-person, except for students who receive Remote Exam permission due to serious illnesses.**

- Usual class time, 65 minutes (extended for DRC students).
  - Hope that all DRC students have submitted their forms for accommodations..
- **No** early/late Exams. **No** make-up Exams. **No** devices.
- **You may bring one double-sided 8.5" x 11" sheet to the Midterm, with anything that you want written or printed on it that you can read unassisted) …**
  - (Okay, you may bring two sheets with writing on only one side of each sheet.)
  - **But you must not use any other material or receive any help during the Midterm, whether you're in the classroom, remote, or in a DRC room**
  - As the Syllabus emphasize, Academic Integrity violations have serious consequences!
- Write your answers on the exam itself, readably, using **either ink or a #2 pencil**.
- We might assign seats to you as you enter the room.

Important Notices: Midterm (continued)

- If you need to take the Midterm Remotely due to serious illness, please send me an email whose subject is "Taking CSE 180 Midterm Remotely" justifying that request.
  - Send that email before 8:00pm on Tuesday, February 13, the day before the Midterm. I'll send you instructions before 10:30am on Wednesday, February 14.
  - Only students whose requests have been approved may take the Midterm Remotely.
- The CSE 180 Midterm from Winter 2023 was posted under Resources→Exams on Piazza on Wednesday, February 1.
  - We haven't covered all the material in that previous Midterm yet.
  - Also, last year's Midterm used that quarter's database schema; your schema is different.
  - Solution to Winter 2023 Midterm has also been posted on Piazza … but I strongly suggest that you take it yourself first, rather than just reading the solution.
- **All past Lectures including the first half of this one, Lecture 7 (Views) will be included on the Midterm.**
  - **The second half of Lecture 7 (Indexes) won't be on the Midterm, even if we discuss it in class.**
- At the end of the Midterm, you'll hand in your Midterm paper, and you'll show your UCSC ID.
  - Do not hand in your 8.5" x 11" sheet.
- **There will be Lab Sections and Tutoring during the rest of the week after the Midterm, on topics including:**
  - Lectures, Lab2 solution, Gradiance, Lab3.
  - Lab3 will be posted on Piazza on February 14, after the Midterm.

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 and Load.
  - Syntax not specified in the SQL standard

CREATE TABLE

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

- PRIMARY KEY
- DEFAULT
- NOT NULL

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 movieTitle, movieYear
    FROM Movies
    WHERE studioName = 'Paramount';
```

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

  ```
  SELECT movieTitle FROM ParamountMovies WHERE movieYear=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 (movieTitle , movieYear , length , genre , studioName , producerC#)

MovieExec (execName , address , cert# , netWorth)

```
CREATE VIEW MovieProd AS
       SELECT m.movieTitle, e.execName, m.length
       FROM Movies m, MovieExec e
        WHERE m.producerC# = e.cert#;

      SELECT DISTINCT length
       FROM MovieProd
       WHERE execName = 'George Lucas';
```

Renaming Attributes in CREATE VIEW

Movies (movieTitle , movieYear , length , genre , studioName , producerC#)

MovieExec (execName , address , cert# , netWorth)

```
CREATE VIEW MovieProd(theTitle, theProducer, theLength) AS
       SELECT m.movieTitle, e.execName, m.length
       FROM Movies m , MovieExec e
        WHERE m.producerC# = e.cert# ;

      SELECT DISTINCT theLength
       FROM MovieProd
       WHERE theProducer = '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 <u>not</u> stored as a table
  - The tables underlying the view are stored in the database, but only the <u>description</u> of the view is in the database.
- 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 movieTitle , movieYear
        FROM Movies
        WHERE studioName = 'Paramount';

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

CREATE VIEW ParamountStars AS
       SELECT DISTINCT s.starName
       FROM ParamountMovies p, StarsIn s
       WHERE p.movieTitle = s.movieTitle AND p.movieYear = s.movieYear ;
```

DROP VIEW

```
CREATE VIEW ParamountMovies AS
        SELECT movieTitle , movieYear
        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;

Materialized Views

- Normally, only the definition of a View is stored in the database; the contents of that view are not stored.
  - That's what we'll assume in this class!
- Some database systems support **Materialized Views**, where users may request that the tuples in a view be physically stored in the database.
  - However ….
- If the relations used in the query that defines the view are updated, then the Materialized View contents becomes out-of-date.
  - **Maintaining** the Materialized View would require changing the view contents whenever its underlying relations are updated.
  - This can be expensive, so it's unusual to maintain Materialized Views.
  - Moreover, some views are not updatable … as we'll soon see.

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(movieTitle, movieYear, length, genre, studioName , producerC#)

```
CREATE VIEW ParamountMovies AS
    SELECT movieTitle, movieYear
    FROM Movies
    WHERE studioName = 'Paramount';

 INSERT INTO ParamountMovies VALUES ('StarTrek', 1979);
```

The INSERT will fail if some other column of Movies (besides movieTitle and movieYear) doesn't have a default, and that column also doesn'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 movieYear = 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, 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 movieYear = 1990;
```

How much would each of these indexes help?

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

CREATE INDEX StudioIndex ON Movies(studioName);

CREATE INDEX YSIndex ON Movies(movieYear, studioName);

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

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

Indexes and Ordering

```
SELECT *
FROM Movies
WHERE studioName = 'Disney' AND movieYear < 1990;
```

How much would each of these indexes help?

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

CREATE INDEX StudioIndex ON Movies(studioName);

CREATE INDEX YSIndex ON Movies(movieYear, studioName);

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

How much would each of the indexes help if the WHERE clause was just movieYear < 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 attribute, 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.