MSiA-413 Introduction to Databases and Information Retrieval

Lecture 10
Creating and Updating SQL Databases

Instructor: Nikos Hardavellas

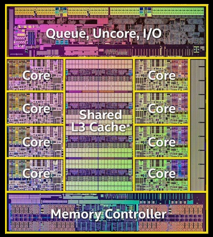
Slides adapted from Steve Tarzia

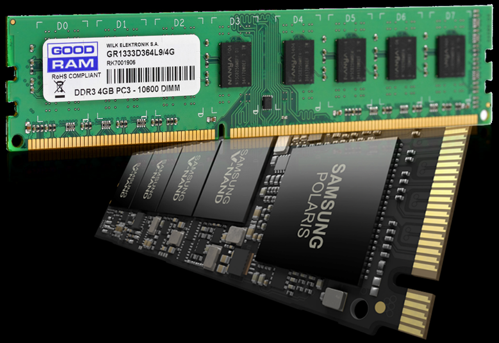
Last Lecture: introduced the *storage hierarchy*

Larger, but slower ↓

delay		*capacity*
0.3ns	CPU Registers	1 kB (kilobyte)
5ns	CPU Caches (L2)	16 MB
50ns	Random Access Memory (RAM)	16 GB
100µs	Flash Storage (SSD)	1 TB
5ms	Magnetic Disk	8 TB

- Disk is about *ten billion* times larger than registers, but has about *ten million* times larger delay (latency)
- Goal is to work as much as possible in the top levels
- Large, rarely-needed data is stored at the bottom level





What is next

- The correct flow dictates we should continue with *database indexing*
- However, the next homework is out, and it is kind of long
- We will switch gears to modifying SQL databases instead
    - This will help you with your homework
    - Indexing will come later

Modifying SQL databases

- Define tables
- Add rows to tables
- Delete rows from tables
- Update columns in a row
- Alter tables by adding or removing:
  - Columns
  - Indexes
  - Foreign keys
- … and much more

- `CREATE TABLE …`
- `INSERT INTO …`
- `DELETE FROM …`
- `UPDATE …`
- `ALTER TABLE …`

I'll be showing the SQLite dialect in these slides. For homework, look up the detailed syntax online:

https://sqlite.org/lang.html

Deleting rows

- DELETE command deletes rows in a table matching some criterion
- Very similar to the `SELECT` statements you are familiar with
- Just replace `SELECT` with `DELETE` and don't specify any columns
- This deletes all the rows in the Classes table for classes in a certain room:

```
DELETE FROM Classes WHERE ClassRoomID=12
```

- If you do not include a WHERE clause, all the rows in that table will be deleted: 🚷😰

```
DELETE FROM Classes
```

- To be safe, run a `SELECT` query first to see what will be deleted:

```
SELECT * FROM Classes WHERE ClassRoomID=12
```

Foreign Keys affect deletions

- SchoolScheduling database
    - Foreign key in the *Classes* table refers to the *Class_Rooms* table
    - What happens if we try to delete a classroom that has several associated classes?
- If you try to delete a row that is a parent to another row there are several possible results, depending on the particular foreign key settings:
    - `RESTRICT` is the default behavior: it would block the deletion
        - You would have to delete the classes first, then the classrooms
    - `CASCADE` causes the child rows to be deleted as well
        - Classes would be deleted
    - `SET NULL` causes the child rows to have the column set to null
        - Classes would remain, but with a NULL ClassRoomId

Updating rows

- UPDATE command is used to change one or more columns in the rows that match some criterion

```
UPDATE Departments SET DeptName="Social Studies”
  WHERE DeptName="History"
```

- Just like DELETE, a single UPDATE command can affect many rows and it can use subqueries:

```
UPDATE Students SET StudMajor=
  (SELECT MajorID FROM Majors WHERE Major="English")
```

- Can also refer to existing column values and use math functions:

```
UPDATE Student_Schedules SET Grade=Grade+5 WHERE ClassID=1500
```

Updating multiple columns

- Use a comma-separated list to update multiple columns at once:

```
UPDATE my_table
  SET column1=value1,
      column2=value2,
      column3=value3
  WHERE id=123
```

Inserting new rows

- `INSERT` command creates one row with the column values specified
- List the column values in the same order that the columns were defined:

```
INSERT INTO Buildings VALUES ("FD", "Ford", 5, 1, 0);
```

- Or, explicitly list the columns being set (this is more clear):

```
INSERT INTO Buildings (BuildingName, BuildingCode,
         NumberOfFloors, ElevatorAccess, SiteParkingAvailable)
  VALUES ("Ford", "FD", 5, 1, 0);
```

- Unspecified columns will get the default value specified when the table was created (more on this later)

Bulk loading data

Several options for inserting lots of rows:

1. Write code in a programming language like R or Python to read the source data and run lots of `INSERT` statements or one really big `INSERT` statement:
   ```
   INSERT INTO table VALUES (1, "cat", 5), (2, "dog", 2), (3, "mouse”, 9) …
   ```
2. Insert values that exist in some database
   ```
   INSERT INTO table SELECT …
   ```
3. Import a CSV file
   - CSV (**C**omma **S**eparated **V**alues) is a very simple, standard spreadsheet format
   - Exact import steps are different for each DBMS
   - In DB Browser for SQLite use *File* → *Import* → *Table from CSV file*
4. Use an ETL software package (**E**xtract, **T**ransform, **L**oad)

Creating tables

- `CREATE TABLE` command defines:
  - Table name
  - Column names
  - Column types (int, float, text, etc.)
  - Whether columns are optional or required (NOT NULL)
  - Primary key
  - Foreign keys
  - Unique keys
  - Indexes (non-unique keys)
- In other words, everything that we drew in the data model diagrams

CREATE TABLE syntax examples from SchoolScheduling.sqlite

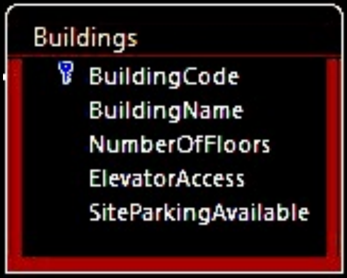

Buildings
BuildingCode
BuildingName
NumberOfFloors
ElevatorAccess
SiteParkingAvailable

Table name

Required column, not optional

Columns

```
CREATE TABLE Buildings (
  BuildingCode nvarchar (3) NOT NULL,
  BuildingName nvarchar (25),
  NumberOfFloors smallint,
  ElevatorAccess bit NOT NULL DEFAULT 0,
  SiteParkingAvailable bit NOT NULL DEFAULT 0,
  PRIMARY KEY (BuildingCode)
);
```

Text with at most 25 characters

Column cannot be NULL, but it will take a value of zero if none is specified.

Description of types in SQLite: **http://www.sqlite.org/datatype3.html**

`CREATE TABLE` syntax example (autoincrement, unique, conflict resolution)

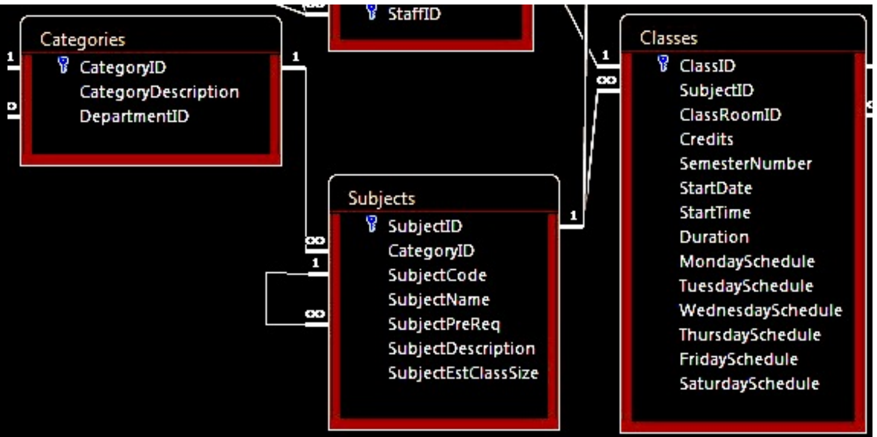
DBMS will assign value automatically

```
CREATE TABLE Movie
(MovieId PRIMARY KEY AUTOINCREMENT,
 name NVARCHAR(100),
 year DATETIME,
 UNIQUE(name, year) ON CONFLICT ABORT);
```

compound unique data integrity constraint

conflict resolution

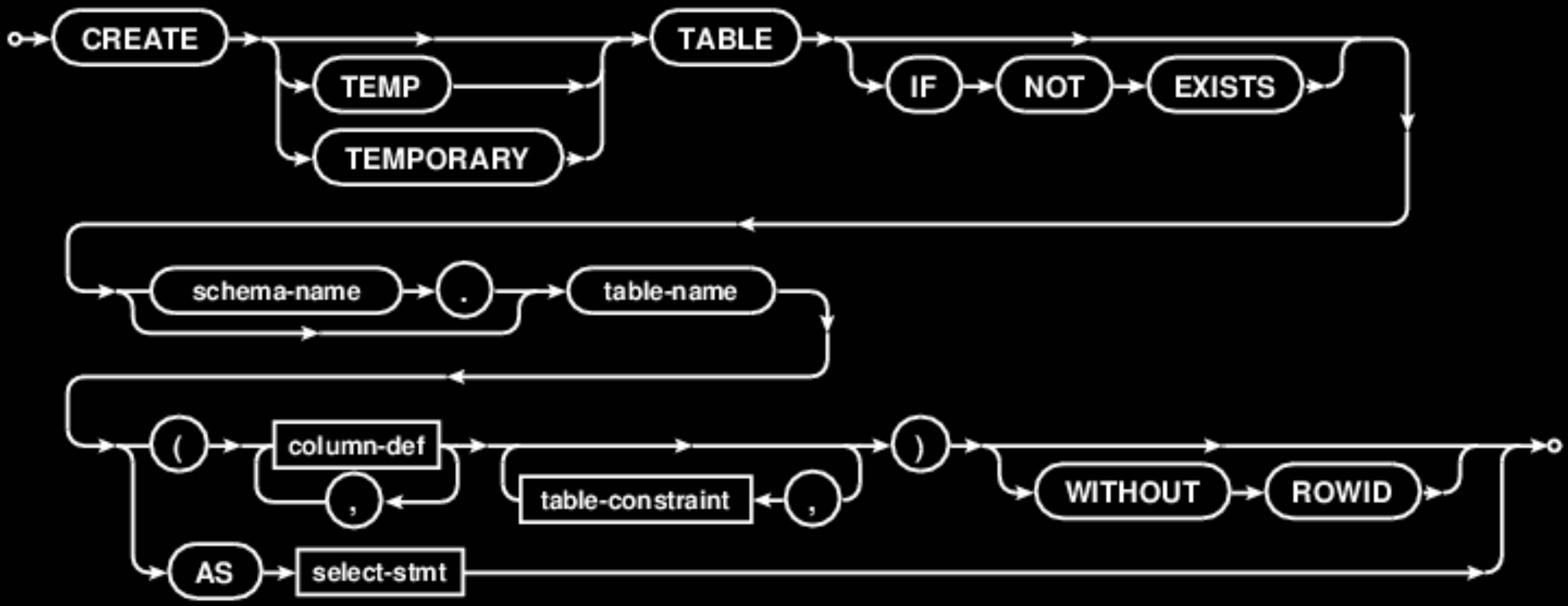
CREATE TABLE syntax examples from SchoolScheduling.sqlite



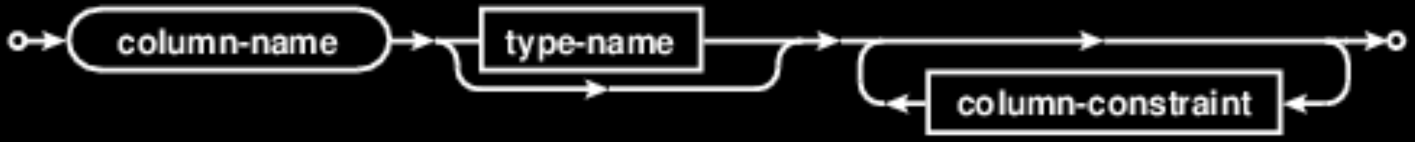
```
CREATE TABLE Subjects (
  SubjectID int NOT NULL DEFAULT 0 ,
  CategoryID nvarchar (10)
      REFERENCES Categories(CategoryID),
  SubjectCode nvarchar (8),
  SubjectName nvarchar (50),
  SubjectPreReq nvarchar (8) DEFAULT NULL
      REFERENCES Subjects(SubjectCode),
  SubjectDescription text,
  SubjectEstClassSize smallint NOT NULL DEFAULT 0,
  PRIMARY KEY (SubjectID),
  UNIQUE (SubjectCode)
);
```

Foreign keys

CREATE TABLE syntax diagram



column-def:



Details: **https://www.sqlite.org/lang_createtable.html**

Column constraint syntax

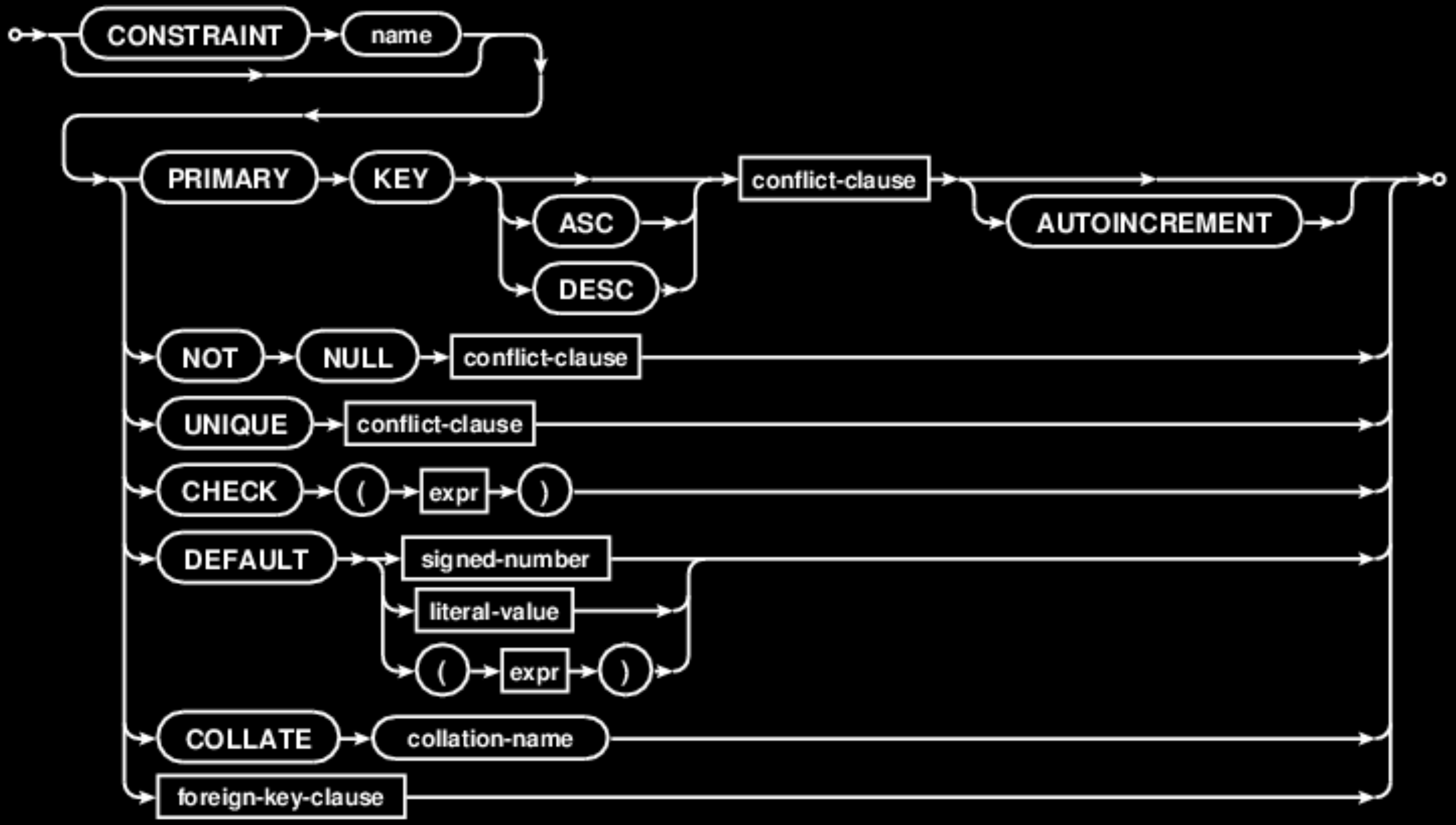
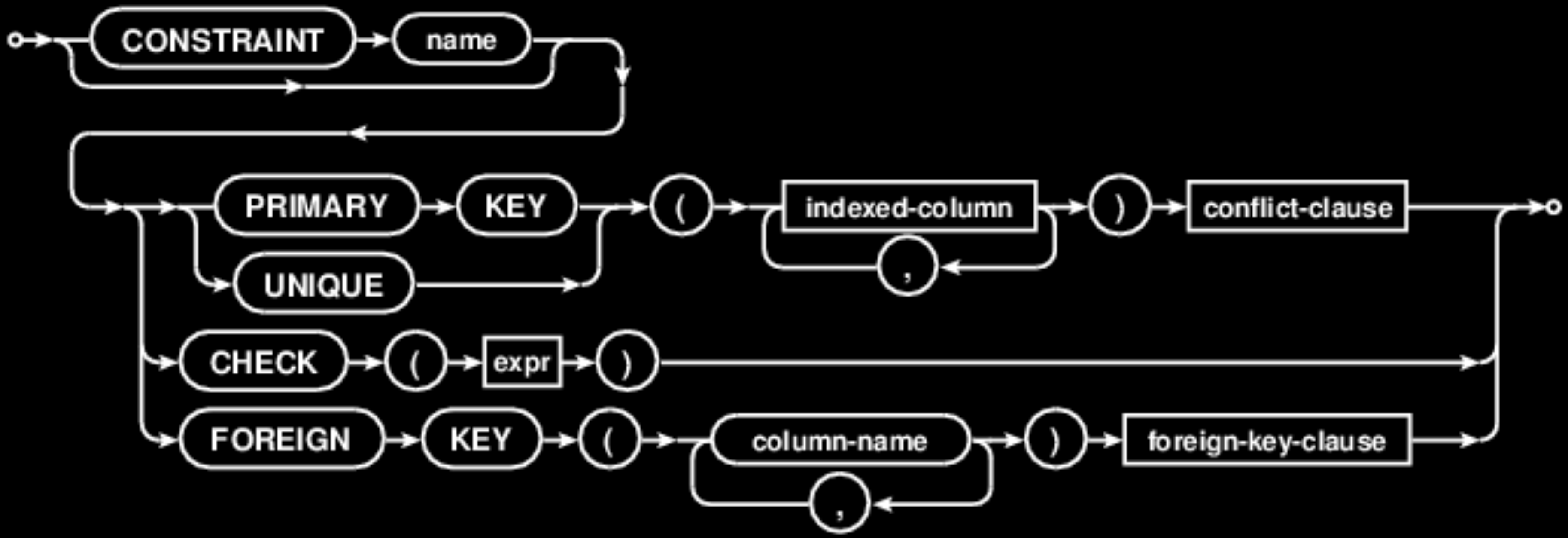


Table constraint syntax



Changing the schema with `ALTER TABLE`

- Add column:
  - `ALTER TABLE ADD COLUMN …`
  - Default value will be filled-in for existing rows (perhaps NULL)
- Remove a column
  - `ALTER TABLE DROP COLUMN …`
- Some DBMSs (but not SQLite) allow changing name or type of column:
  - `ALTER TABLE ALTER COLUMN …`

For more information...

- Check SQL as Understood by SQLite at:
- **https://sqlite.org/lang.html**