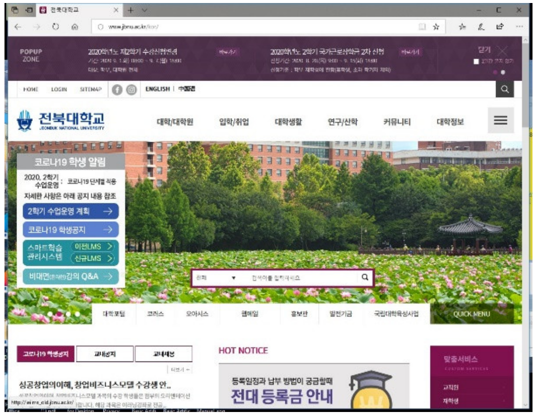
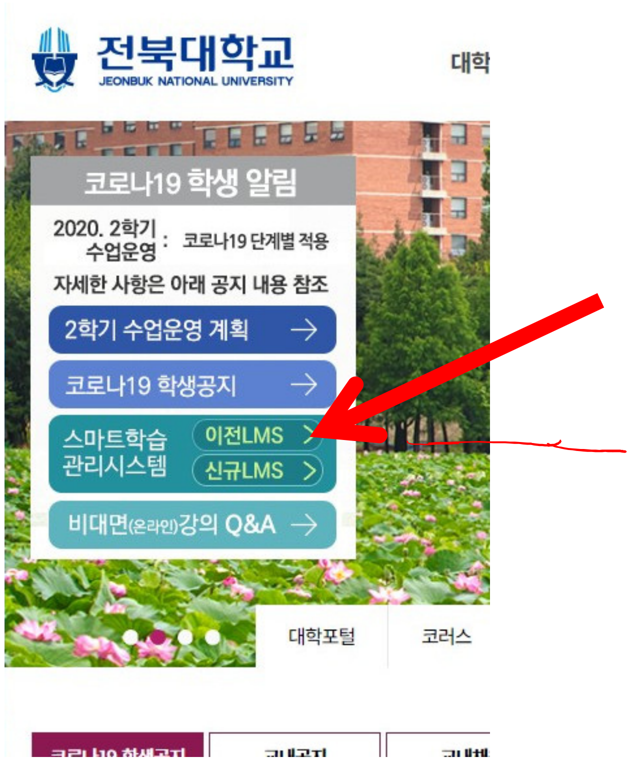
Introduction to Data Structure (Data Management)

Felipe P. Vista IV

Announcement



http://jbnu.ac.kr/



http://ieilms_old.jbnu.ac.kr/ → http://ieilmsold.jbnu.ac.kr/

Announcement

- Everybody, make sure that your name in ZOOM is in the following format:
  - University ID Num Name
  - Ex: 202054321 Juan Dela Cruz
  - Otabek change your names w/ ID Num
  - Not changing your name, you might be marked Absent

Course Outline

- Introduction, Data Models, SQL Basics
- SQL Aggregates, Grouping, Subqueries
- Wrapping-up SQL, Relational Algebra (RA), Datalog
- NoSQL, JSON
- JSON, SQL++
- SQL++, RA Part II, Query Evaluation
- Storage, Indexing Basics
- Basics of Query Optimization, Parallel Databases
- Map Reduce, Spark
- E/R Diagrams, Constraints
- Design Theory
- Transactions
- DB Techniques for Machine Learning

Data Management

DATA MODEL
(CH 2.1 – 2.2)

Data Model

- language/notation for discussing data
  - describe data/ information
  - how data is connected to each other, processed, & stored inside system

Data Model

- language/notation for discussing data
  - describe data/ information
  - how data is connected to each other, processed, & stored inside system

- models that will be used in this course
  - relational: data is a collection of tables
  - semi-structured: data is a tree

Data Model

- language/notation for discussing data
  - describe data/ information
  - how data is connected to each other, processed, & stored inside system

- models that will be used in this course
  - relational: data is a collection of tables
  - semi-structured: data is a tree

- other models
  - key-value pairs: used by NoSQL systems
  - graph data model: used by RDF (semi-structured can also be used)
  - object oriented: usually layered on relational, J2EE

Relational Model

- data is a collection of relations/tables:

Relational Model

- data is a collection of relations/tables:
- mathematically, relation is a set of tuple
  - each tuple* appears 0 or 1 times in a table, order of rows is unspecified

* tuples (in relational DB) – one record/ row

Relational Model

- data is a collection of relations/tables:
- mathematically, relation is a set of tuple
  - each tuple* appears 0 or 1 times in a table, order of rows is unspecified

* tuples (in relational DB) – one record/ row

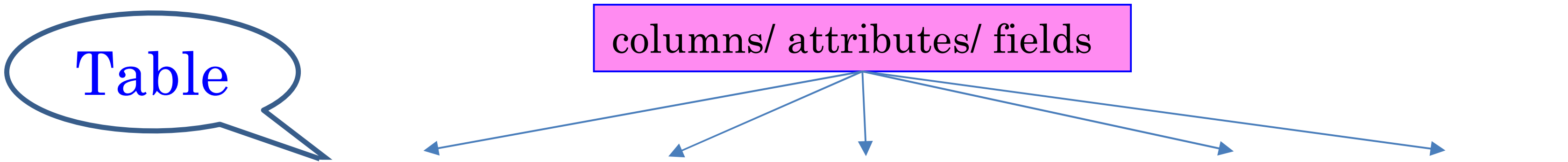
Table

Name	Country	Occupation	Years_Biking	Has_Bike
Soheil	Iran	Graduate Student	20	1
Nwabisa	South Africa	Teacher	12	1
Matt	USA	Teacher	15	0
Mikki	USA	Teacher	10	0
Divan	Iran	Student	13	1
Khan Boy	South Korea	Heavy Equipment Operator	18	0
Pat	Hong Kong	Teacher	9	1
Janin	Philippines	Artist	13	1

Relational Model

- data is a collection of relations/tables:
- mathematically, relation is a set of tuple
  - each tuple* appears 0 or 1 times in a table, order of rows is unspecified

* tuples (in relational DB) – one record/ row



Table

columns/ attributes/ fields

Name	Country	Occupation	Years_Biking	Has_Bike
Soheil	Iran	Graduate Student	20	1
Nwabisa	South Africa	Teacher	12	1
Matt	USA	Teacher	15	0
Mikki	USA	Teacher	10	0
Divan	Iran	Student	13	1
Khan Boy	South Korea	Heavy Equipment Operator	18	0
Pat	Hong Kong	Teacher	9	1
Janin	Philippines	Artist	13	1

Relational Model

- data is a collection of relations/tables:
- mathematically, relation is a set of tuple
  - each tuple* appears 0 or 1 times in a table, order of rows is unspecified

* tuples (in relational DB) – one record/ row

Table

columns/ attributes/ fields

rows/ tuples/ records

Name	Country	Occupation	Years_Biking	Has_Bike
Soheil	Iran	Graduate Student	20	1
Nwabisa	South Africa	Teacher	12	1
Matt	USA	Teacher	15	0
Mikki	USA	Teacher	10	0
Divan	Iran	Student	13	1
Khan Boy	South Korea	Heavy Equipment Operator	18	0
Pat	Hong Kong	Teacher	9	1
Janin	Philippines	Artist	13	1

Relational Schema

- Each column has a “domain” or type
  - SQL has Java-style types for numbers, strings, etc
  - domain is the constraint* on data allowed in the table

* constraint– limit or restrict

Relational Schema

- Each column has a “domain” or type
  - SQL has Java-style types for numbers, strings, etc
  - domain is the constraint* on data allowed in the table
- “names” & “type” form part of schema of table

```
Crew(Name:string, Country:string, Occupation:string,
     Years_Biking:int, Has_Bike:Boolean)
```

* constraint– limit or restrict

Relational Schema

- Each column has a “domain” or type
  - SQL has Java-style types for numbers, strings, etc
  - domain is the constraint* on data allowed in the table
- “names” & “type” form part of schema of table

```
Crew(Name:string, Country:string, Occupation:string,
     Years_Biking:int, Has_Bike:Boolean)
```

- particular data is “instance” of the relationship
  - Data changes over time
  - DBMS usually stores the current(latest) instance

Name: ~~Johnny~~
↓
~~Paul~~
↓
→ Peter

* constraint– limit or restrict

Keys

- “key “
  - subset of columns that uniquely identifies tuple

Keys

- “key “
  - subset of columns that uniquely identifies tuple
- another constraint on the table
  - No two tuples can have the same values for those columns

Keys

- “key “
  - subset of columns that uniquely identifies tuple
- another constraint on the table
  - No two tuples can have the same values for those columns

— K K + 1983
— K K + 2017

- examples
  TAble
  - `Movie(title, year, length, genre)`: key is (title, year)
  - What is good key for “`Crew`”?

Keys

- "key "
  - subset of columns that uniquely identifies tuple
- another constraint on the table
  - No two tuples can have the same values for those columns
- examples
  - `Movie(title, year, length, genre)`: key is (title, year)
  - What is good key for "`Crew`"?
- it forms part of the schema

pk SK

```
Crew(Name:string, Country:string, Occupation:string,
     Years_Biking:int, Has_Bike:Boolean)
```

Keys (continuation)

- there can be multiple keys in a table

Keys (continuation)

- there can be multiple keys in a table
- only one of the keys in a table can be "Primary key"
  - DBMS often designed that search using primary key is the fastest
  - other keys in the table are called "secondary"

Keys (continuation)

- there can be multiple keys in a table

CREW

- only one of the keys in a table can be "Primary key"
  - DBMS often designed that search using primary key is the fastest
  - other keys in the table are called "secondary"

Location

- "Foreign key"
  - column (or columns) whose value is a key of another table
  - a reference to another row in another table

Keys (continuation)

CREW

Name	Country	Occupation	Years_Biking	Has_Bike	LocID
Soheil	Iran	Graduate Student	20	1	3
Nwabisa	South Africa	Teacher	12	1 B	2
Matt	USA	Teacher	15	0	1
Mikki	USA	Teacher	10	0	1
Divan	Iran	Student	13	1	3
Khan Boy	South Korea	Heavy Equipment Operator	18	0	5
Pat	Hong Kong	Teacher	9	1 B	4
Janin	Philippines	Artist	13	1	3

LocID	Dong	City	Province
1	Songcheondong	Jeonju	Jeollabukdo
2	Soshindong	Jeonju	Jeollabukdo
3	Deokjindong	Jeonju	Jeollabukdo
4	Inhudong	Jeonju	Jeollabukdo
5	Uadong	Jeonju	Jeollabukdo

LOCATION

Keys (continuation)

Primary Key

Foreign Key

Name	Country	Occupation	Years_Biking	Has_Bike	LocID
Soheil	Iran	Graduate Student	20	1	3
Nwabisa	South Africa	Teacher	12	1	2
Matt	USA	Teacher	15	0	1
Mikki	USA	Teacher	10	0	1
Divan	Iran	Student	13	1	3
Khan Boy	South Korea	Heavy Equipment Operator	18	0	5
Pat	Hong Kong	Teacher	9	1	4
Janin	Philippines	Artist	13	1	3

CREW

Primary Key

LocID	Dong	City	Province
1	Songcheondong	Jeonju	Jeollabukdo
2	Soshindong	Jeonju	Jeollabukdo
3	Deokjindong	Jeonju	Jeollabukdo
4	Inhudong	Jeonju	Jeollabukdo
5	Uadong	Jeonju	Jeollabukdo

Dong	City	Pr

LOCATION

Data Management

SQL BASICS (CH 2.3)

SQL ("sequel")

- Standard Query Language for relational data
  - used for DBs in many different contexts*
  - Inspire query languages for non-relational (ex. SQL++)

* context – condition that create particular situation or event

SQL ("sequel")

- Standard Query Language for relational data
  - used for DBs in many different contexts*
  - Inspire query languages for non-relational (ex. SQL++)

- Everything not quoted ('…') are case insensitive**

'Bike' <> 'bIke' <> 'BIKE'

Bike = bIkE = BIKE

* context – condition that create particular situation or event

** case insensitive – does not matter if upper case or lower case

SQL (“sequel”)

- Provide standard type
  - numbers: INT, FLOAT, DECIMAL(p,s)
    - DECIMAL(p,s): p = precision, s = scale
    - Ex: DECIMAL(4,2) → is a number with 4 digits before decimal point & 2 digits after
  - strings: CHAR(n), VARCHAR(n)
    - CHAR(n): string with fixed length of n
    - VARCHAR(n): variable length string with maximum length of n

XXXX.XX

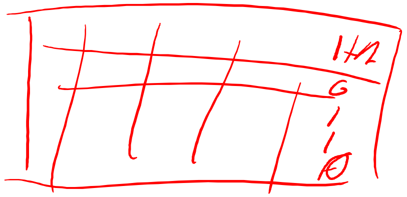
CHAR(5) = 'HAPPY' = 'SAD' = FLOWER

VARCHAR(10) = HAPPY = SAD = FLOWER FLOWE

SQL (“sequel”)(continuation)

↓ COL/FIELD

- Provide standard type
  - BOOLEAN
    rows
    tuples
    - 1 0
      True or False
  - DATE, TIME, TIMESTAMP
    - DATE : store year, month, and day values
    - TIME : store hour, minute, and second values
    - TIMESTAMP: store year, month, day, hour, minute and second values
  - Additional types vary depending on the vendor
    - SQLite: http://www.sqlite.org/datatype3.html
    - SQLite do not have separate storage class for BOOLEAN
    - Values are stored as integer: 0 → False, 1 → True
      INT = 0/1



SQL Statements

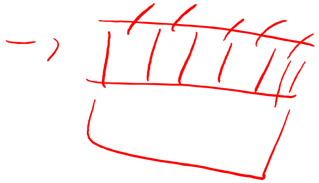
syntax

- create table …
- drop table …
- alter table … add/remove …
- insert into … values …
- delete from … where …
- update … set … where

create table …

TABLE NAME   crew   CREW

```
CREATE TABLE Crew(
      name VARCHAR(20) PRIMARY KEY,
      country VARCHAR(20),
      occupation VARCHAR(40),
      years_biking INT,
      has_bike INT);
```



BOOL

drop table ...

```
DROP TABLE Crew;
```

alter table … add/remove

```
ALTER TABLE Crew
     ADD phone_num INT;
```

insert into ... values

```
INSERT INTO Crew VALUES
      (‘Ipe’,‘Sa Puso Mo’,
      ‘Researcher’,35,1);
```

CREW = 8

delete from ... where

```
DELETE FROM Crew
    WHERE name=‘Ipe’;
```

update ... set... where

```
UPDATE Crew                               =13
     SET years_biking = years_biking + 3
     WHERE name='Matt';
```

Data Management

SQLITE

SQLite

- SQLite
  - C-library that implements relational DBMS
  - simple and lightweight: good for embedded software
    - but does not provide all of the functionalities that other DBMSs do
  - Can be used as part of any C/C++/Java program
    - Can be used for an iPhone app, and possibly Android

http://www.sqlite.org/lang.html (SQL Syntax)
http://www.sqlite.org/datatype3.html (SQL Data type)
http://www.w3schools.com/sql/default.asp (w3school SQL tutorial)

SQLite

- SQLite
  - C-library that implements relational DBMS
  - simple and lightweight: good for embedded software
    - but does not provide all of the functionalities that other DBMSs do
  - Can be used as part of any C/C++/Java program
    - Can be used for an iPhone app, and possibly Android
- Sqlite3
  - standalone program that can run queries & manage a SQLite database

http://www.sqlite.org/lang.html (SQL Syntax)
http://www.sqlite.org/datatype3.html (SQL Data type)
http://www.w3schools.com/sql/default.asp (w3school SQL tutorial)

Physical data independence

- SQL does not specify how data is stored in the disk

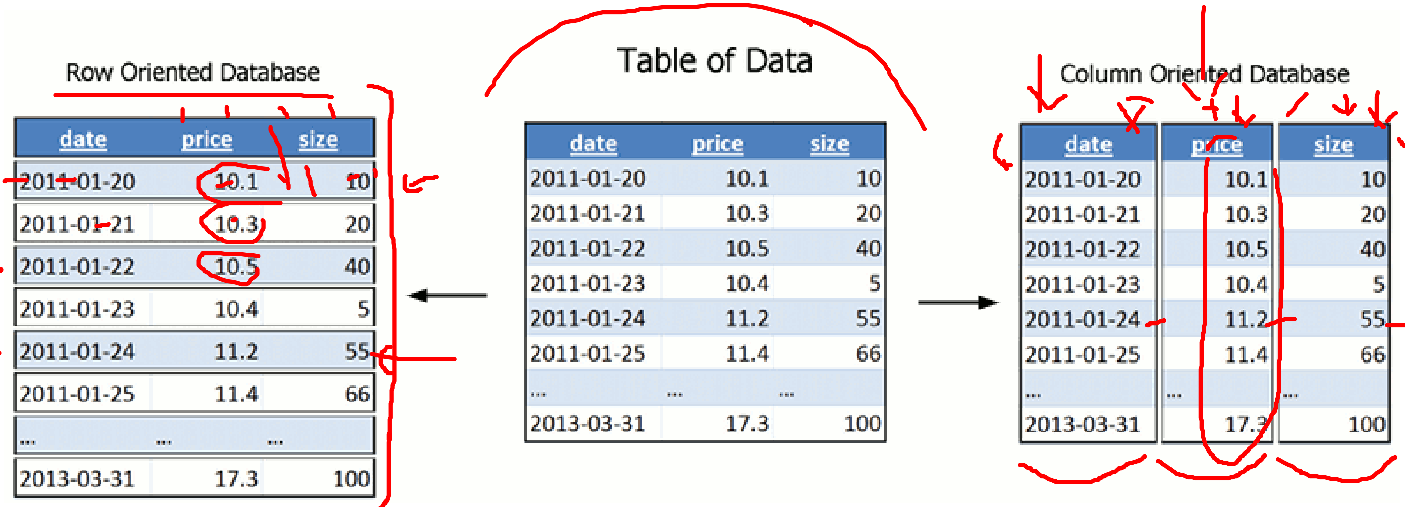
Physical data independence

- SQL does not specify how data is stored in the disk
- No need to consider encodings of data types
  - Ex: DECIMAL(10,2) – . - - _ 4.25
  - Ex: VARCHAR(255)
    - Do we need 255 bytes to store the word ‘annyeong’?

Physical data independence

- SQL does not specify how data is stored in the disk
- No need to consider encodings of data types
  - Ex: DECIMAL(10,2)
  - Ex: VARCHAR(255)
    - Do we need 255 bytes to store the word ‘annyeong’?
- No need to consider how the tuples are arranged
  - Is it row- or column-ordered?
  - Most DBMS are row-ordered but Google’s BigQuery id column-ordered

Row- vs Column-ordered?



Row Oriented Database

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Table of Data

date	price	size
2011-01-20	10.1	10
2011-01-21	10.3	20
2011-01-22	10.5	40
2011-01-23	10.4	5
2011-01-24	11.2	55
2011-01-25	11.4	66
...	...	...
2013-03-31	17.3	100

Column Oriented Database

date
2011-01-20
2011-01-21
2011-01-22
2011-01-23
2011-01-24
2011-01-25
...
2013-03-31

price
10.1
10.3
10.5
10.4
11.2
11.4
...
17.3

size
10
20
40
5
55
66
...
100

Operation	Column-oriented	Row-oriented
Aggregate single column, e.g. Sum(price)	√ Fast	Slow
Compression	√ Higher, since similar data together	-
Retrieve few columns from table with many columns	√ Faster	Must skip unnecessary data
Insert/update single new record	Slow	√ Fast
Retrieve single record(tuple)	Slow	√ Fast

http://www.timestored.com/time-series-data/what-is-a-column-oriented-database

SQLite's "Wait a minute…"!

- Allows NULL keys
  - One tuple at the most can have NULL in the key
  - SQL standard →Primary Key always NOT NULL but SQLite otherwise

SQLite's "Wait a minute…"!

- Allows NULL keys
  - One tuple at the most can have NULL in the key
  - SQL standard →Primary Key always NOT NULL but SQLite otherwise
- No support for BOOLEAN or DATE/TIME columns
  - Instead of Boolean: values stored as integer: 0 → False, 1 → True
  - Instead of Date/Time: store as TEXT, REAL or INT class
    - use SQLite functions to convert from DATE/TIME ←~→ SQLite[TEXT, REAL, INT]

SQLite’s “Wait a minute…”!

- Do not always enforce domain constraints
  - Allow inserting a STRING even if INT is expected

SQLite's "Wait a minute…"!

- Do not always enforce domain constraints
  - Allow inserting a STRING even if INT is expected
- Do not enforce Foreign Key constraint by default
  - `PRAGMA foreign_keys = ON;`

SQLite: How to run sqlite3

- Linux and Mac
  - Open terminal
  - type “`sqlite3 <DB Name>`”
  - `sqlite3 BikerMice`
    - typing only “`sqlite3`” without DB name will still work
    - but all your data will not be saved/available when you exit
    - because you are using DB in memory and not saved into a file

SQLite: How to run sqlite3

- Linux and Mac
  - Open terminal
  - type "`sqlite3 <DB Name>`"
  - `sqlite3 BikerMice`
    - typing only "`sqlite3`" without DB name will still work
    - but all your data will not be saved/available when you exit
    - because you are using DB in memory and not saved into a file
- For linux
  - If sqlite3 not found or error, install by typing at terminal
  - `sudo apt-get install sqlite3`

SQLite: How to run sqlite3

- Windows
  - Go to https://www.sqlite.org/download.html
  - Download “sqlite-tools”, current version is “sqlite-tools-win32-x86-3330000.zip”
  - Extract zip file
  - Go to command line and then run same as Linux/Mac
  - type “`sqlite3 <DB Name>`”
  - `sqlite3 BikerMice`
    - typing only “`sqlite3`” without DB name will still work
    - but all your data will not be saved/available when you exit
    - because you are using DB in memory and not saved into a file

SQLite: Commands (not SQL)

- `.help`
  - list other . commands
- `.header(s) ON/OFF`
  - show/hide column headers in query results
- `.mode [mode type]`
  - change how to separate the columns in each row/tuple (for better formatting)
- `.read [file name]`
  - read and execute SQL code from the given file
- `.separator [string]`
  - change the separator for output mode or importing files, i.e. .separator ,
- `.nullvalue [string]`
  - print the given string in place of NULL values

SQLite: Commands (not SQL)

- `.import [file name] [table name]`
  - load the file to the table, be careful to set the separator correctly!
- `.show`
  - see how we have set our parameters
- `.exit`
  - exit from sqlite3

SQLite: Basic SQL Statements

- CREATE - creates a new table
  - `CREATE TABLE [table] ( … );`
- INSERT INTO - inserts new data into a table
  - `INSERT INTO [table] VALUES ([value1],[value2],.);`
- SELECT - extracts data from a table
  - `SELECT [column(s)] FROM [table_name];`
- UPDATE - updates data in a table
  - `UPDATE FROM [table] SET … WHERE …;`
- DELETE - deletes data from a table
  - `DELETE FROM [table] WHERE …;`

Note:
Queries are case-insensitive in SQLite

SQLite: SQL keywords, operators, etc…

- WHERE clause
  - w/out WHERE

ALL

30 col

5k T/rows

```
SELECT * FROM Crew;
```

OUTPUT

F/L

```
name|country|occupation|years_biking|has_bike|locID
Soheil|Iran|Graduate Student|20|1|3
Nwabisa|South Africa|Teacher|12|1|2
Matt|USA|Teacher|15|0|1
Mikki|USA|Teacher|10|0|1
Divan|Iran|Student|13|1|3
Khan Boy|South Korea|Heavy Equipment
Operator|15|0|5
Pat|Hong Kong|Teacher|9|1|4
Janin|Philippines|Artist|13|1|3
```

SQLite: SQL keywords, operators, etc…

- WHERE clause
  - filter records; using WHERE

```
            F/L        TN          F            =
SELECT name FROM Crew WHERE (occupation = “Teacher”;
```

  - Careful using STRING for WHERE clause: “Teacher” <> “teacher” <> “tEAcHeR”
  - Use LOWER or UPPER

```
SELECT name,country,occupation,years_biking FROM
Crew WHERE LOWER(occupation) = “teacher”;
```

```
*OUTPUT*
name|country|occupation|years_biking
Nwabisa|South Africa|Teacher|12
Matt|USA|Teacher|15
Mikki|USA|Teacher|10
Pat|Hong Kong|Teacher|9
```

SQLite: SQL keywords, operators, etc…

- AND, OR operator
  - filter records based on more than one condition

```
SELECT name FROM Crew WHERE lower(occupation) =
“teacher” AND years_biking >= 10 AND has_bike;
```

= 0=F

1=T

Boolean

OR

```
*OUTPUT*
name
Nwabisa
```

SQLite: SQL keywords, operators, etc…

- LIKE operator
  - used in a WHERE clause to search for a specified pattern in a column
  - Often used wildcards
  - % → represents zero, one or multiple characters
  - _ → (underscore), represents a single character

```
SELECT name, occupation FROM Crew WHERE
LOWER(occupation) LIKE '%student';
```

= 'student' ≠ "Grads"

- Return name & occupation whose occupation has the word "student" at the end

?

%2019% → X

(20191008)3

```
*OUTPUT*
Name|occupation
Soheil|Graduate Student
Divan|Student
```

SQLite: SQL keywords, operators, etc…

- LIKE operator
  - used in a WHERE clause to search for a specified pattern in a column
  - Often used wildcards
  - % → represents zero, one or multiple characters
  - _ → (underscore), represents a single character

```
SELECT name,country FROM Crew WHERE LOWER(name)
LIKE '_a%';
```

"xaxxx+"
xa
xaxx
~~xxaxx~~
x x a%

  - Return name & country whose name has "a" as second letter in their name

```
*OUTPUT*
name|country
Matt|USA
Pat|Hong Kong
Janin|Philippines
```

SQLite: SQL keywords, operators, etc…

- AS
  - give an alias name to a table or a column

```
SELECT name, occupation AS Work, FROM Crew WHERE
 has_bike;

*OUTPUT*
name|Work
Soheil|Graduate Student
Nwabisa|Teacher
Divan|Student
Pat|Teacher
Janin|Artist
```

SQLite: SQL keywords, operators, etc…

- AS
  - give an alias name to a table or a column
  - combine several columns into into one field

```
SELECT locID AS Code, dong||’, ’||city||’,
 ’||province AS Address FROM location WHERE
LOWER(city) = “jeonju”;
```

vAdd = A + c + p

vAdd = Add

```
*OUTPUT*
Code|Address
1|Songcheondong Jeonju Jeollabukdo
2|Soshindong, Jeonju, Jeollabukdo
3|Deokjindong Jeonju Jeollabukdo
4|Hyojadong Jeonju Jeollabukdo
5|Inhudong Jeonju Jeollabukdo
6|Uadong Jeonju Jeollabukdo
```

Add(,)

[illegible]

SQLite: SQL keywords, operators, etc…

- Relational operators: =, >, >=, <, <=
- Special functions:
  - DATE(…), LENGTH(string), SUBSTR(string, startIndex, endIndex), etc…

Hello   subs('' , 2, 3)

(x, 2, 5)

#

SQLite: Important notes

- SQLite allows a key to be null

SQLite: Important notes

- SQLite allows a key to be null
- Older versions do not enforce FOREIGN KEY constraints.
  - Newer versions are opt-in at both compile time and runtime (with `PRAGMA FOREIGN_KEYS = ON`)

SQLite: Important notes

- SQLite allows a key to be null
- Older versions do not enforce FOREIGN KEY constraints.
  - Newer versions are opt-in at both compile time and runtime (with `PRAGMA FOREIGN_KEYS = ON`)
- SQLite ignores string length maximums or fixed string lengths: `N` in `VARCHAR(N)` or `CHAR(N)`

SQLite: Important notes

- SQLite allows a key to be null
- Older versions do not enforce FOREIGN KEY constraints.
  - Newer versions are opt-in at both compile time and runtime (with `PRAGMA FOREIGN_KEYS = ON`)
- SQLite ignores string length maximums or fixed string lengths: `N` in `VARCHAR(N)` or `CHAR(N)`
- SQLite does not have a separate data type for dates, times, or combined date and time.
  - Instead, these are represented as specially formatted strings; dates are represented as yyyy-mm-dd

SQLite: Important notes

- SQLite allows a key to be null
- Older versions do not enforce FOREIGN KEY constraints.
  - Newer versions are opt-in at both compile time and runtime (with `PRAGMA FOREIGN_KEYS = ON`)
- SQLite ignores string length maximums or fixed string lengths: `N` in `VARCHAR(N)` or `CHAR(N)`
- SQLite does not have a separate data type for dates, times, or combined date and time.
  - Instead, these are represented as specially formatted strings; dates are represented as yyyy-mm-dd
- And many more as we go and encounter them!

Demo on SQLite

- to start sqlite
  - type “`sqlite3 <DB Name>`” → `sqlite3 BikerMice` db
    - typing only “`sqlite3`” without DB name will still work
    - but all your data will not be saved/available when you exit
    - because you are using DB in memory and not saved into a file

Demo on SQLite

- to start sqlite
  - type "`sqlite3 <DB Name>`" → `sqlite3 BikerMice`
    - typing only "`sqlite3`" without DB name will still work
    - but all your data will not be saved/available when you exit
    - because you are using DB in memory and not saved into a file
- Enable headers (column names)
  - Type inside sqlite → `.headers = ON;`

Demo on SQLite

- to start sqlite
  - type “`sqlite3 <DB Name>`” → `sqlite3 BikerMice`
    - typing only “`sqlite3`” without DB name will still work
    - but all your data will not be saved/available when you exit
    - because you are using DB in memory and not saved into a file
- Enable headers (column names)
  - Type inside sqlite → `.headers = ON;`
- To enable foreign key constraint
  - Type inside sqlite → `PRAGMA foreign_keys = ON;`

Demo on SQLite

- to start sqlite
  - type "`sqlite3 <DB Name>`" → `sqlite3 BikerMice`
    - typing only "`sqlite3`" without DB name will still work
    - but all your data will not be saved/available when you exit
    - because you are using DB in memory and not saved into a file
- Enable headers (column names)
  - Type inside sqlite → `.headers = ON;`
- To enable foreign key constraint
  - Type inside sqlite → `PRAGMA foreign_keys = ON;`
- to exit/quit sqlite3:
  - Type inside sqlite → `.exit`