CSCB20
Introduction to Databases and Web Application

Week 2 - RA and Introduction to SQL

Dr. Purva R Gawde

Thanks to Dr. Anna Bretscher for the material in this set of slides

Topics covered this week

- Review of Last Week
- Relational Algebra Examples
- Introduction to SQL
- Mapping from Relational Algebra to SQL

Review of last Week

Schemas

- Relation schema : relation name and attribute list.
    - Optionally: domains of attributes.
    - Instructor(ID: Integer, name: Text, dept_name: Text, salary: Integer)
- Database : collection of relations.
- Database schema : set of all relation schemas in the database.

What is Algebra?

- Mathematical system consisting of:
  - Operands --- variables or values from which new values can be constructed.
  - Operators --- symbols denoting procedures that construct new values from given values.

What is Relational Algebra?

- An algebra whose operands are relations or variables that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.

The result is an algebra that can be used as a query language for relations.

The Core Relational Algebra

- Selection ("σ"):
  - choosing (selecting) certain rows.
- Projection ("Π"):
  - choosing (projecting) certain columns.
- Product ("×") & Join ("⋈"):
  - compositions of relations.
- Union ("∪"), Intersection ("∩"), & Difference ("−"):
  - the usual set operations;
  - but both operands must have a matching schema.
- Rename ("ρ"):
  - renaming of relations and attributes.

Relational Algebra Example

Running Example

- Example:

```
Beers(name, manf)
Bars(name, addr, license)
Drinkers(name, addr, phone)
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)
```

- Underlines indicate the key attributes

Selection

$R1 := \sigma_p(R2)$

p is a predicate/condition — as in “if” statements — written over the attributes of R2 that evaluates to true or false per tuple.

R1 is the set of all the tuples of R2 that satisfy p; that is, those tuples from R2 for which p evaluates true.

Example: Selection

Relation Sells:

bar	beer	price
Joe’s	Bud	2.50
Joe’s	Miller	2.75
Sue’s	Bud	2.50
Sue’s	Miller	3.00

JoeMenu := $\sigma_{bar=\text{“Joe’s”}}$(Sells)

bar	beer	price
Joe’s	Bud	2.50
Joe’s	Miller	2.75

Projection

$R1 := \pi_L(R2)$

- L is a list of attributes from the schema of R2.
- R1 is constructed by
    - taking each tuple from R2,
    - extracting the attr's from the tuple in list L, and
    - creating from those components a tuple for R1.
    - Eliminate duplicate tuples in R1, if any.

Example: Projection

Relation Sells:

bar	beer	price
Joe’s	Bud	2.50
Joe’s	Miller	2.75
Sue’s	Bud	2.50
Sue’s	Miller	3.00

Menu := $\pi_{beer,\ price}$(Sells):

beer	price
Bud	2.50
Miller	2.75
Bud	2.50
Miller	3.00

More on Projection

$R1 := \Pi_L(R2)$

- Using the same Π_L operator, we allow the list L to contain arbitrary expressions involving attributes:
    - Arithmetic on attributes,
        - e.g., A + B -> C.
    - Duplicate occurrences of the same attribute.

Example: Projection

R = (

A	B
1	2
3	4

)

$R1 := \Pi_{A + B \text{ -> } C,\ A,\ A}(R) =$

C	A1	A2
3	1	1
7	3	3

Cartesian Product

- R1 := R2 × R3

- Pair each tuple t2 ∈ R2 with each tuple t3 ∈ R3.
- The concatenation t2t3 is a tuple of R1.

- The schema of R1 is the union of the attributes of R2 and R3.

- **Note**. If there is an attr. named A in both R2 and R3, we get both copies; by convention, we rename them R2.A and R3.A, respectively.

Example: Product

R1 = (

A	B
1	2
3	4

)

R2 = (

B	C
5	6
7	8
9	10

)

R3 =(

A	R1.B	R2.B	C
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

Theta Join

- $R1 := R2 \bowtie_\theta R3$
- Take the product $R2 \times R3$.
- Then apply σ_θ to the results.
- Thus, $R2 \bowtie_\theta R3 \equiv \sigma_\theta (R2 \times R3)$.
- As with σ, θ can be any boolean-valued condition.
- Historic versions of this operator allowed only A θ B,
  - where θ is =, <, etc.;
  - hence the name “theta-join.”

Example: Theta-Join

Sells = (

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Bars = (

name	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo= (

bar	beer	price	name	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Miller	3.00	Sue's	River Rd.

)

Natural Join

- R1 := R2 ⋈ R3
- A useful join variant (natural join) connects two relations by:
    - Equating attributes of the same name, and
    - Projecting out one copy of each pair of equated attributes.

Example: Natural Join

Sells = (

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

)

Bars = (

bar	addr
Joe's	Maple St.
Sue's	River Rd.

)

Note: Bars.name needs to become Bars.bar to make the natural join "work."

BarInfo := Sells ⋈ Bars

BarInfo= (

bar	beer	price	addr
Joe's	Bud	2.50	Maple St.
Joe's	Miller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Miller	3.00	River Rd.

)

Example: Theta Join

Sells = (

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

)

Bars = (

name	addr
Joe's	Maple St.
Sue's	River Rd.

)

Note: attribute bar in sells is equated attribute name is Bars relation.

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo= (

bar	beer	price	name	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Joe's	River Rd.
Sue's	Miller	3.00	Joe's	River Rd.

)

Renaming

- The ϱ operator gives a new schema to a relation. It is a way to “rename” a relation.
- $R1 := \varrho_{R1(A1,\ldots,An)}(R2)$
- makes R1 as a relation with attr's $A_1,\ldots,A_n$, and the same tuples as R2.
- Simplified notation. $R_1(A_1,\ldots,A_n) := R_2$.

Example: Renaming

Bars = (

name	addr
Joe's	Maple St.
Sue's	River Rd.

)

$$R := \rho_{R(bar,\ addr)}\ \mathbf{Bars}$$

R (

bar	addr
Joe's	Maple St.
Sue's	River Rd.

)

Building complex expressions

- Combine operations with parentheses and precedence rules.
- Three notations, as in arithmetic.
  - Sequences of assignment statements.
  - Expressions with several operators. (Precedence)
  - Expression trees.

Sequence of Assignments

- Create temporary relation names.
- Renaming can be implied by giving relations a list of attributes.
- Example: R3 := R1 $\bowtie_{\theta}$ R2 can be written:
  - R4 := R1 X R2
  - R3 := σ_{θ} (R4)

Expressions with several operators

- Example: the theta-join R3 := R1 $\bowtie_\theta$ R2 can be written:
    - R3 := σ_θ (R1 X R2)

- Precedence of relational operators:

1. [σ, π, ρ] (highest).
2. [X, ⋈].
3. ∩.
4. [∪, —]

Expression Trees

- Leaves are operands --- either variables standing for relations or particular, constant relations.
- Interior nodes are operators, applied to their child or children.

Example: Tree for Query

- Using the relations

  Bars(name, addr) and

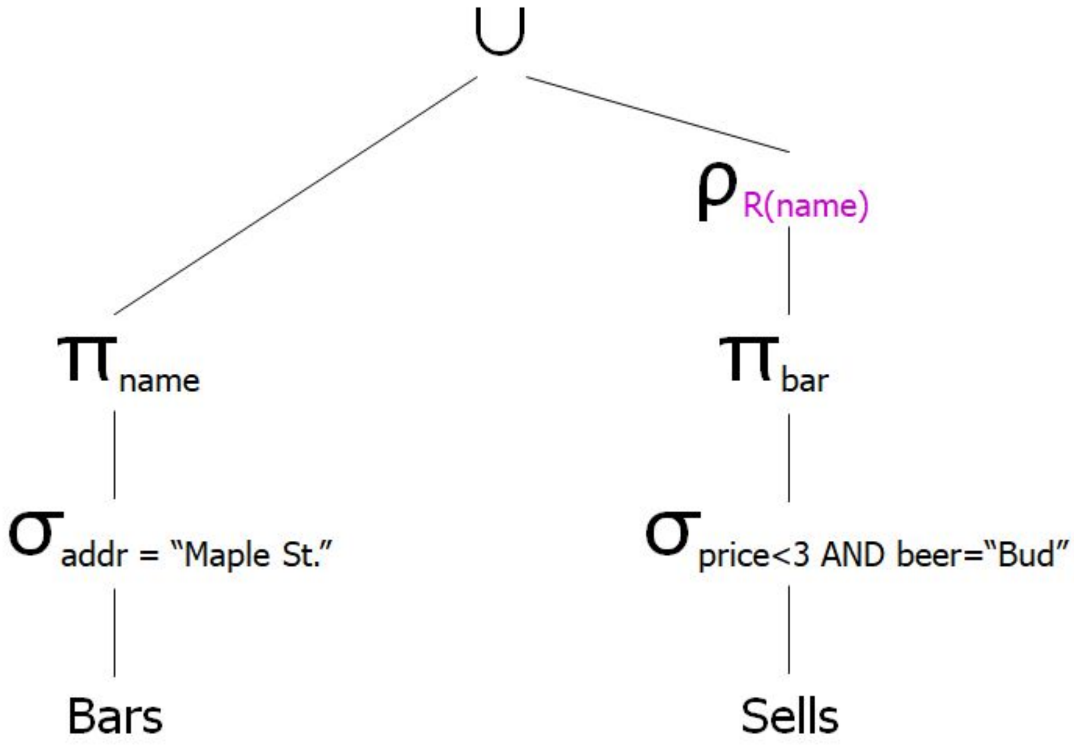
  Sells(bar, beer, price),

  find the names of all the bars that are

  either on Maple St. or sell Bud for less

  than $3.

As a Tree

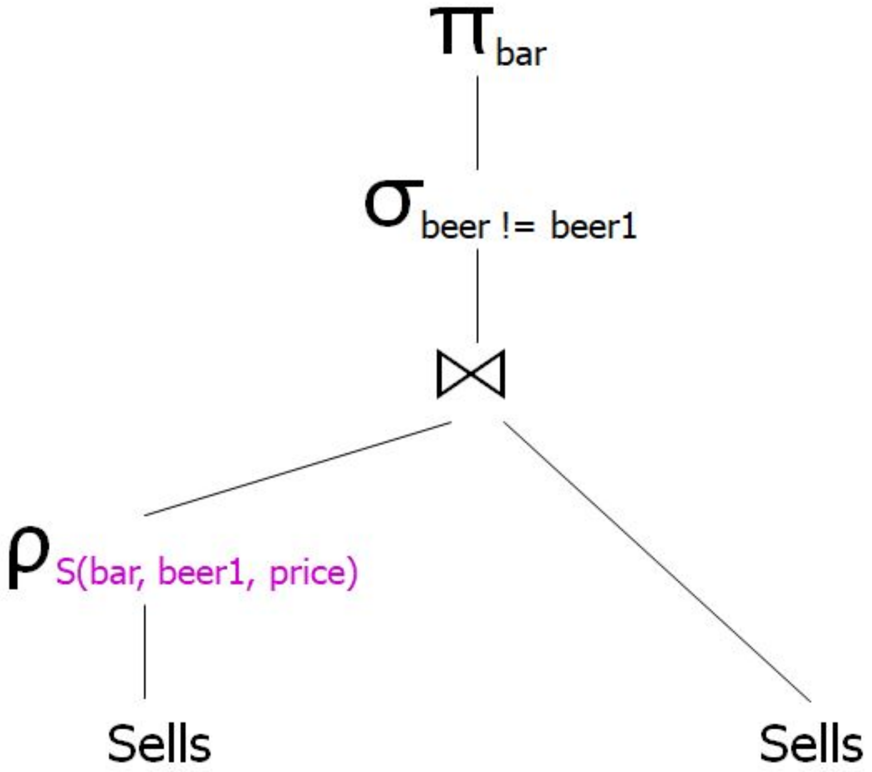


∪
ρ R(name)
π name
π bar
σ addr = "Maple St."
σ price<3 AND beer="Bud"
Bars
Sells

Example of self-join

- Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.
- Strategy:
    - by renaming, define a copy of Sells, called S(bar, beer1, price).
    - The natural join of Sells and S consists of quadruples (bar, beer, beer1, price) such that the bar sells both beers at this price.

The Tree



Limitations of Relational Algebra

- Relational algebra is set-based
- Real-life applications need more
  - Expensive (and often unnecessary) to eliminate duplicates
  - Important (and often expensive) to order output
  - Need a way to apply scalar expressions to values
  - What’s *not* there often as important as what is

Introduction to SQL

Why SQL?

- SQL is a very-high-level language.
  - Structured Query Language
  - Say “what to do” rather than “how to do it.”
  - Avoid a lot of data-manipulation details needed in languages like C++ or Java.
- Database management system figures out “best” way to execute query.
  - Called “query optimization.”

SQL(Structured Query Language)

- SQL is a standard language for accessing databases.
- SQL is used to access and manipulate data in:
  - Postgresql, MySQL, SQL Server, Access, Oracle, Sybase, IBM DB2, and other database systems.

The SQL language has several parts:

- Data-definition language (DDL)
  - provides commands for defining relation schemas, deleting relations, and modifying relation schemas

- Data-manipulation language (DML).
  - provides the ability to query information from the database and modify tuples in the database.

Relational DBMS

MySQL

Microsoft SQL Server

PostgreSQL

SQLite

https://www.sqlite.org/index.html

SQL Query Form

SELECT desired attributes
FROM one or more tables
WHERE condition about tuples of the tables

SQL and RA*

ORDER BY

SELECT

HAVING

GROUP BY

WHERE

FROM

*We haven't covered all RA operators in this slide

The Project Operation

$\Pi_{\text{ID, name, salary}}$ (instructor)

SQL Notation:

SELECT col_1,..., col_N
FROM instructor

OR

SELECT *
FROM instructor

(means select all columns)

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

The *instructor* relation

SELECT ID, name, salary
FROM instructor

The Select Operation

$\sigma_{\text{salary} >= 85000}$ (instructor)

SQL Notation:

SELECT *
FROM instructor
WHERE salary >= 85000

SELECT col_1,…, col_N
FROM instructor
WHERE salary >= 85000

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

The *instructor* relation

The Natural Join Operation

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

The *instructor* relation

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

The *teaches* relation

The Natural Join Operation

instructor ⋈ teaches

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

SQL Notation:
```
SELECT *
FROM instructor NATURAL JOIN Teaches;
```

The Cartesian Product Operation - Without Common attribute

Relations: r,s

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

r X s

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

SQL Notation:
SELECT * FROM r INNER JOIN s
or
SELECT * FROM r, s

Note: can have as many relations as needed…but what may be a concern?

The Cartesian Product Operation

Relations: r,s

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

r X s

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

SQL Notation:
SELECT *
FROM r INNER JOIN s
or
SELECT * FROM r, s

What if we don't want ALL rows?
For example, we want rows where A's value and C's value are equal?

SELECT * FROM r INNER JOIN s ON A = C

Inner Join

SQL Notation:

SELECT Column1, Column2, …, ColumnK

FROM   Table A INNER JOIN Table B ON join_constraints

WHERE contraints

ORDER BY ColumnX

There are many other options, we will see these later…

Self Join

Suppose we want to join a table to itself.

We want to find those departments that are in the same building

department A

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

department B

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

SQL Notation

```
SELECT A.dept_name, B.dept_name

FROM department A INNER JOIN department B ON A.building = B.building
```

The Union Operation

For r ∪ s to be valid:

1. r, s must have the same arity
   a. (same number of attributes)
2. The attribute domains must be compatible
   a. i.e, 2nd column of r deals with the same type of values as does the 2nd column of s.

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

SQL Notation:

```
(SELECT * FROM r)
UNION
(SELECT * FROM s)
```

Use UNION ALL to keep duplicates.

r ∪ s

A	B
α	1
α	2
β	1
β	3

The Intersection Operation

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cap s = r - (r - s)$

A	B
α	2

SQL Notation:

```
(SELECT *
FROM r)
INTERSECT
(SELECT *
FROM s)
```

The Difference Operation

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r – s

A	B
α	1
β	1

SQL Notation:

```
(SELECT *
FROM r)
EXCEPT
(SELECT *
FROM s)
```

Data types in SQLite

- **NULL**. The value is a NULL value.
- **INTEGER**. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
- **REAL**. The value is a floating point value, stored as an 8-byte IEEE floating point number.
- **TEXT**. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
- **BLOB**. The value is a blob of data, stored exactly as it was input

NULL values

- Every type can have the special value null.
- A value of null indicates the value is unknown or that it may not exist at all.
- Sometimes we do not want a null value at all – we can add such a constraint.

SQL - Data Definition Language

Studying SQL:
Study DDL and DML

Creating a Table

- **SQL Notation:**

```
CREATE TABLE table_name
    (col_name1 type1 PRIMARY KEY,
    col_name2 type2 NOT NULL,
    … ,
    col_namen typen,
    <integrity---constraint1>,
    … ,
    <integrity---constraintk>);
```

Integrity Constraints

- Primary key(list of attributes) :
  - These attributes form the primary keys for the relation.
  - Primary keys must be non---null and unique.
- Foreign key(list of attributes) references s :
  - The values of these attributes for any tuple in the relation must correspond to values of the primary key attributes of some tuple in relation s.
- not null:
  - Specifies that this airibute may not have the null value. We list this constraint when defining the type of the attribute

Examples

```
CREATE TABLE department
      (dept_name       TEXT,
      building         TEXT,
      budget           INTEGER,
      PRIMARY KEY (dept_name));
```

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

```
CREATE TABLE course
      (course_id                TEXT,
      Title                     TEXT,
      dept_name                 TEXT,
      credit                    INTEGER,
      PRIMARY KEY (course_id),
      FOREIGN KEY (dept_name) REFERENCES department );
```

Editing Tables

```
DROP TABLE table_name;          //remove the table

DELETE FROM table_name          //delete tuples satisfying the predicate
WHERE predicate;

ALTER TABLE table_name          //add a column
ADD column type;

ALTER TABLE table_name          // remove a column
DROP column;
```

Inserting in SQLite

Inserting Single row:

```
INSERT INTO table (column1,column2 ,..)
VALUES( value1, value2 ,...);
```

Inserting multiple rows:

```
INSERT INTO table1 (column1,column 2 ,..)
VALUES
   (value1,value2 ,...),
   ...
   (valuen,valuen ,...);
```

Inserting using SELECT query:

```
INSERT INTO table_name
SELECT QUERY
```

For example:

```
INSERT INTO instructor
SELECT ID, name, dept_name, 18000
FROM student
WHERE dept_name = ‘Music’ AND tot_cred > 144;
```

Updating in SQLite

```
UPDATE table_name
    SET attribute = new_value
OR
UPDATE table_name
    SET attribute = new_value
    WHERE predicate or select statement;
OR
UPDATE table_name
    SET attribute = CASE
        WHEN predicate_1 THEN result_1
        WHEN predicate_2 THEN result_2
        …
        WHEN predicate_n THEN result_n
        ELSE result_0
       END
```

SQLite LIKE operator

- To query data based on partial information, you use the LIKE operator in the WHERE clause of the SELECT statement as follows:

  ```
  SELECT column_list
  FROM table_name
  WHERE column_1 LIKE pattern;
  ```

SQLite provides two wildcards for constructing patterns.

The percent sign % wildcard matches any sequence of zero or more characters.

The underscore _ wildcard matches any single character.

Next week

More SQL..