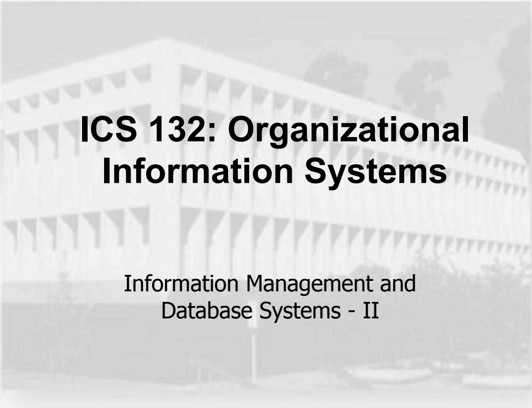


ICS 132: Organizational Information Systems

Information Management and
Database Systems - II

SQL

- SQL is the Structured Query Language
  - originally developed for IBM's System/R in 1970s
  - now an open standard (actually, a bunch of them)
- a common interface for relational DB's
  - manipulation
    - creating tables, updating them, adding data
  - examination
    - looking data up: *queries*

SQL

- queries have three basic components
  - select *something*
    - what aspects of the data do we want to see
  - from *somewhere*
    - what tables contain it
  - where *condition*
    - filtering of results
- basic syntax
  - `select attribute1, attribute2,… from relation1, relation2, … where predicate`

SQL

- some basic examples
  - `select title from books`
  - `select title from books where author=‘dourish’`
  - `select title from books where author=‘dourish’ and price < 35.00`
  - `select grade from students where id=‘12312312’`
  - `select id,name from students where grade=‘f’`

SQL

- queries across multiple tables
  - relational model splits data into different tables
  - queries need to integrate across multiple tables
  - selects that combine table are called *joins*
- example
  - tables: "students" (id, name), "grades" (id, score)
  -
    ```
    select name, grade
    from students, grades
    where students.id = grades.id
    ```

SQL

- joins aren't as clever as you'd think
  - a basic pairwise combination of possible elements
    - `select name,grade from students,grades where grade=‘A’`

SQL

- joins aren't as clever as you'd think
  - a basic pairwise combination of possible elements
    - `select name,grade from students,grades where grade='A'`
    - `select name,grade from students,grades where grade='A'` ***and students.id = grades.id***

SQL

- joins aren't as clever as you'd think
  - a basic pairwise combination of possible elements
    - `select name,grade from students,grades where grade='A'`
    - `select name,grade from students,grades where grade='A'` ***and students.id = grades.id***
  - need to resolve ambiguous references
    - `select` ***students***`.id,name,grade from from students,grades where grade='A' and students.id=grades.id`

SQL

- combining results
  - union, intersect, except
  - these are operators over *selections*
- examples
  - `select title from books where author = 'dourish' except select title from books where title = 'context-aware computing'`
  - `select id from homework1 where score > 85 intersect select id from homework2 where score > 85`
  - *NB:* neither of these are the easiest ways to do them…

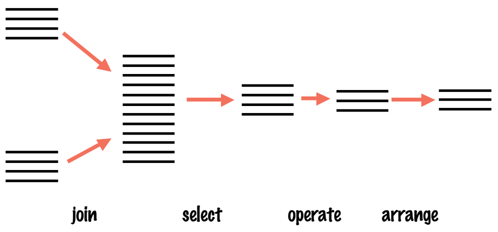
SQL

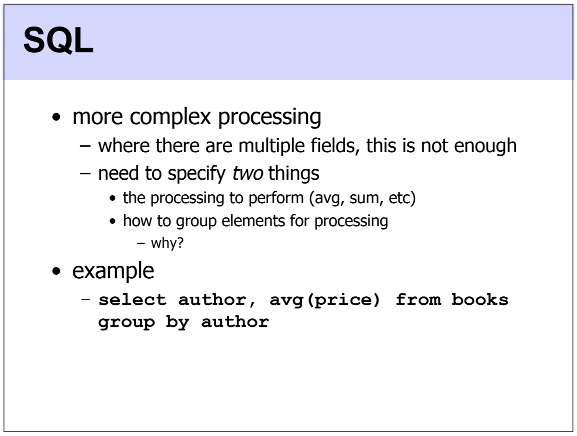
- postprocessing (order, group)
  - need to organise results
  - order (sort), group (clustering)
- examples
  - `select id,name,score from students order by score`
  - `select id,name,score from students order by score limit 10`
  - `select model, price from products where price < 100 order by price desc`
  - `select manufacturer from price_list group by manufacturer`

SQL

- some functions over results
  - e.g. avg(), sum(), count(), min(), max() …
  - functions apply to single columns
    - collapse multiple entries to a single value
- examples
  - `select count(*) from students where grade='a'`
  - `select avg(score) from grades`

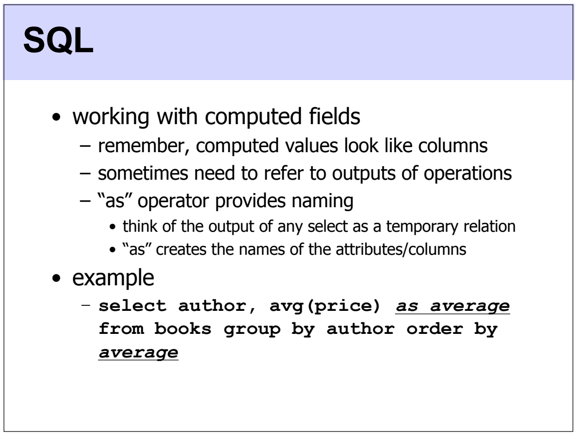
processing stages





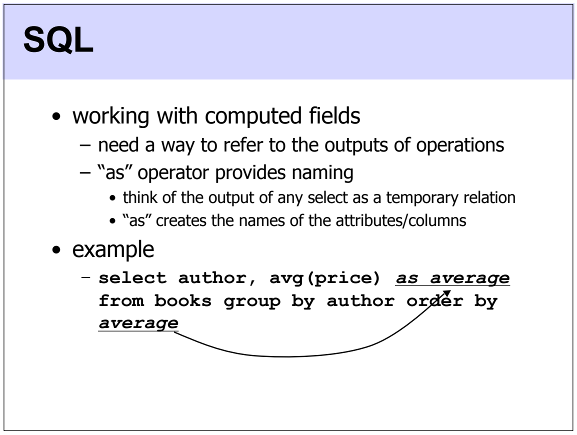
SQL

- more complex processing
  - where there are multiple fields, this is not enough
  - need to specify *two* things
    - the processing to perform (avg, sum, etc)
    - how to group elements for processing
      - why?
- example
  - **`select author, avg(price) from books group by author`**



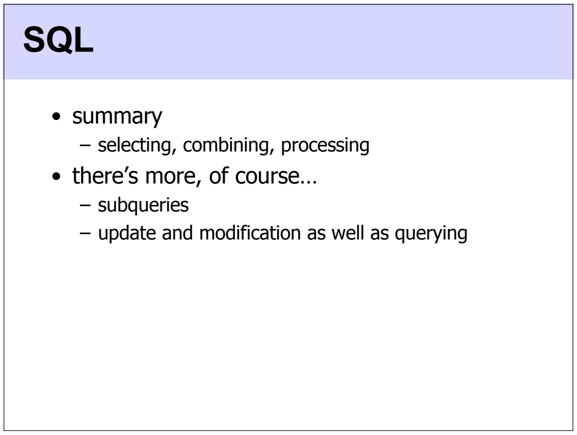
SQL

- working with computed fields
  - remember, computed values look like columns
  - sometimes need to refer to outputs of operations
  - "as" operator provides naming
    - think of the output of any select as a temporary relation
    - "as" creates the names of the attributes/columns
- example
  - **`select author, avg(price)`** ***as average*** **`from books group by author order by`** ***average***



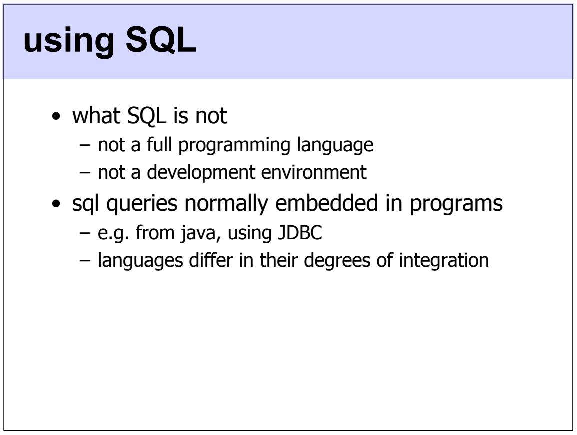
SQL

- working with computed fields
  - need a way to refer to the outputs of operations
  - "as" operator provides naming
    - think of the output of any select as a temporary relation
    - "as" creates the names of the attributes/columns
- example
  - **`select author, avg(price)`** ***as average*** **`from books group by author order by`** ***average***



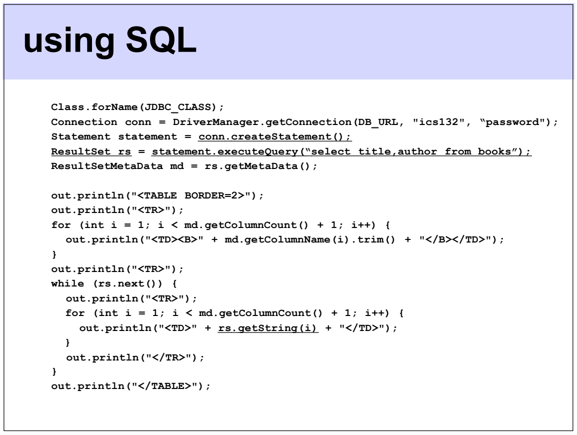
SQL

- summary
  - selecting, combining, processing
- there's more, of course…
  - subqueries
  - update and modification as well as querying



using SQL

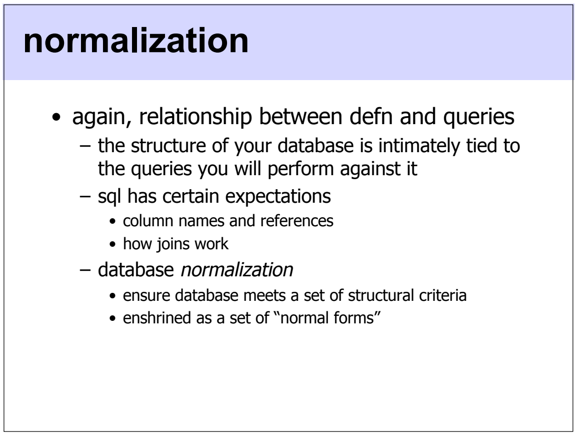
- what SQL is not
  - not a full programming language
  - not a development environment
- sql queries normally embedded in programs
  - e.g. from java, using JDBC
  - languages differ in their degrees of integration



using SQL

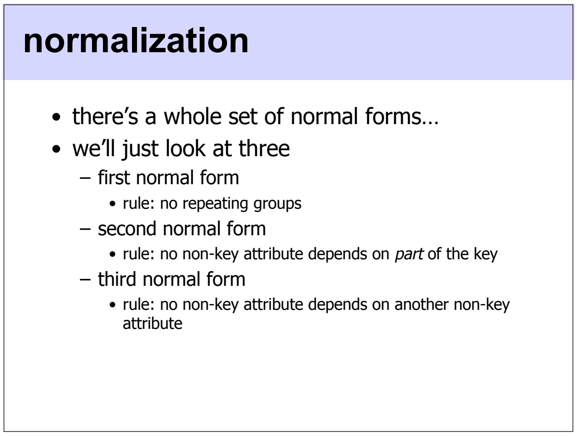
```
Class.forName(JDBC_CLASS);
Connection conn = DriverManager.getConnection(DB_URL, "ics132", "password");
Statement statement = conn.createStatement();
ResultSet rs = statement.executeQuery("select title,author from books");
ResultSetMetaData md = rs.getMetaData();

out.println("<TABLE BORDER=2>");
out.println("<TR>");
for (int i = 1; i < md.getColumnCount() + 1; i++) {
  out.println("<TD><B>" + md.getColumnName(i).trim() + "</B></TD>");
}
out.println("<TR>");
while (rs.next()) {
  out.println("<TR>");
  for (int i = 1; i < md.getColumnCount() + 1; i++) {
    out.println("<TD>" + rs.getString(i) + "</TD>");
  }
  out.println("</TR>");
}
out.println("</TABLE>");
```



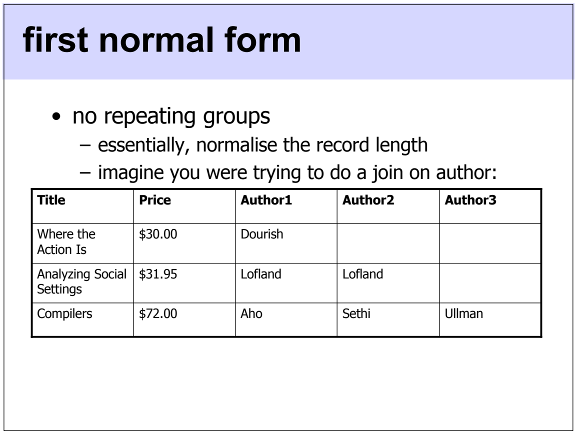
normalization

- again, relationship between defn and queries
  - the structure of your database is intimately tied to the queries you will perform against it
  - sql has certain expectations
    - column names and references
    - how joins work
  - database *normalization*
    - ensure database meets a set of structural criteria
    - enshrined as a set of "normal forms"



normalization

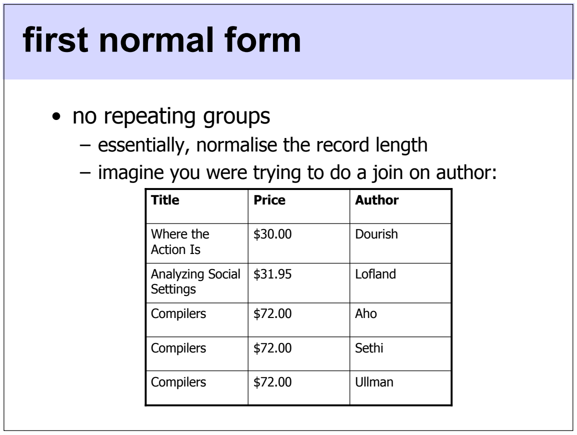
- there's a whole set of normal forms…
- we'll just look at three
  - first normal form
    - rule: no repeating groups
  - second normal form
    - rule: no non-key attribute depends on *part* of the key
  - third normal form
    - rule: no non-key attribute depends on another non-key attribute



first normal form

- no repeating groups
  - essentially, normalise the record length
  - imagine you were trying to do a join on author:

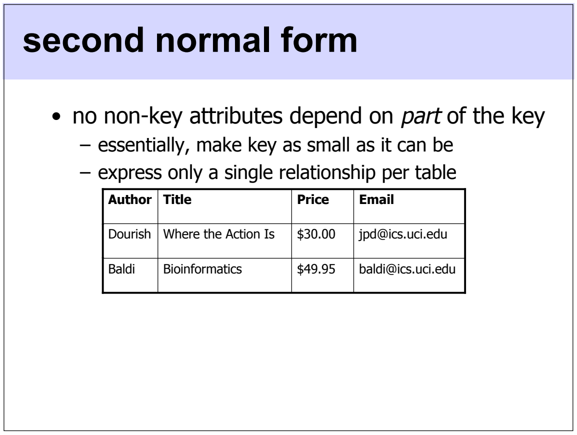
Title	Price	Author1	Author2	Author3
Where the Action Is	$30.00	Dourish		
Analyzing Social Settings	$31.95	Lofland	Lofland	
Compilers	$72.00	Aho	Sethi	Ullman



first normal form

- no repeating groups
  - essentially, normalise the record length
  - imagine you were trying to do a join on author:

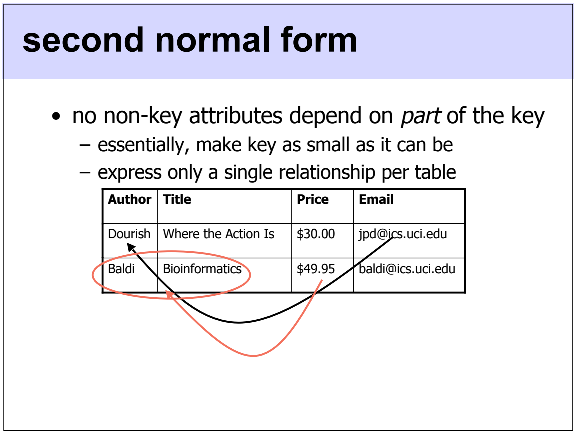
Title	Price	Author
Where the Action Is	$30.00	Dourish
Analyzing Social Settings	$31.95	Lofland
Compilers	$72.00	Aho
Compilers	$72.00	Sethi
Compilers	$72.00	Ullman



second normal form

- no non-key attributes depend on *part* of the key
  - essentially, make key as small as it can be
  - express only a single relationship per table

Author	Title	Price	Email
Dourish	Where the Action Is	$30.00	jpd@ics.uci.edu
Baldi	Bioinformatics	$49.95	baldi@ics.uci.edu



second normal form

- no non-key attributes depend on *part* of the key
  - essentially, make key as small as it can be
  - express only a single relationship per table

Author	Title	Price	Email
Dourish	Where the Action Is	$30.00	jpd@ics.uci.edu
Baldi	Bioinformatics	$49.95	baldi@ics.uci.edu

second normal form

- no non-key attributes depend on *part* of the key
  - essentially, make key as small as it can be
  - express only a single relationship per table

Author	Email
Dourish	jpd@ics.uci.edu
Baldi	baldi@ics.uci.edu

Author	Title	Price
Dourish	Where the Action Is	$30.00
Baldi	Informatics	$49.95

third normal form

- no attributes depend on other *non*-key attributes
  - again, a row should be about just one relationship

Author	Title	Price	Purchaser	Seller	Employed
Dourish	Where the Action Is	$30.00	Maria	Hans	1/1/03
Dourish	Where the Action Is	$30.00	Joey	Amy	1/1/02
Baldi	Bioinformatics	$49.95	Lisa	Jaime	7/1/01

third normal form

- no attributes depend on other *non*-key attributes
  - again, a row should be about just one relationship

Author	Title	Price	Purchaser	Seller	Employed
Dourish	Where the Action Is	$30.00	Maria	Hans	1/1/03
Dourish	Where the Action Is	$30.00	Joey	Amy	1/1/02
Baldi	Bioinformatics	$49.95	Lisa	Jaime	7/1/01

third normal form

- no attributes depend on other *non*-key attributes
  - again, a row should be about just one relationship

Title	Purchaser	Seller
Where the Action Is	Maria	Hans
Where the Action Is	Joey	Amy
Bioinformatics	Lisa	Jaime

Seller	Employed
Hans	1/1/03
Amy	1/1/02
Jaime	7/1/01

Author	Title	Price
Dourish	Where the Action Is	$30.00
Baldi	Informatics	$49.95

normalization

- normalization transforms database structure
  - eliminates repetition
  - disentangles dependencies
  - clarifies relationships
- two benefits of these transformations
  - semantic
    - cleaner definitions
    - clarifies "meaning"
  - practical
    - optimizes for SQL-based queries

next time

- an assignment on this stuff
  - to be done online
- moving on from machine metaphor
  - organisms
    - performance and competition
    - communication and interaction