Data Representation

This first course section starts by presenting two common data representation models.

- The *entity-relationship (ER)* model
- The *relational* model

Data Manipulation

This is followed by some methods for manipulating data in the relational model and using it to extract information.

*Relational algebra*
- The *tuple-relational calculus*
- The query language *SQL*
A transaction is a single coherent operation on a database. This might involve substantial amounts of data, or take considerable computation; but is meant to be an all-or-nothing action.

The features that characterise a reliable implementation of transactions are standardly initialized as the ACID properties.

**Task**

Find out what each letter A C I D stands for here, and what those four terms mean.
Remember Codd’s Diagram?

A DATABASE SYSTEM

STORED ~ DATA

SOFTWARE

BATCH PGMS

ON-LINE USERS
Homework from Tuesday

A *transaction* is a single coherent operation on a database. This might involve substantial amounts of data, or take considerable computation; but is meant to be an all-or-nothing action.

The features that characterise a reliable implementation of transactions are standardly initialized as the *ACID* properties.

<table>
<thead>
<tr>
<th>Initials</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Atomicity</td>
</tr>
<tr>
<td>C</td>
<td>Consistency</td>
</tr>
<tr>
<td>I</td>
<td>Isolation</td>
</tr>
<tr>
<td>D</td>
<td>Durability</td>
</tr>
</tbody>
</table>
ACID Transactions for Reliable Multiuser Databases

**Atomicity**  All-or-nothing: a transaction either runs to completion, or fails and leaves the database unchanged.

This may involve a *rollback* mechanism to undo a partially-complete transaction.

**Consistency**  Applying a transaction in a valid state of the database will always give a valid result state.

This requires maintaining constraints and cascades; and rolling back a transaction if it will break any of these.
**Isolation** Concurrent transactions have the same effect as sequential ones: the outcome is as if they were done in order.

Transactions may, in fact, run at the same time: but should never see each other’s intermediate state. In concurrent programming languages this is known as *sequential consistency*.

**Durability** Once a transaction is committed, it will not be rolled back.

May need many levels: non-volatile memory; uninterruptible power; distributed commit protocols.
ACID Transactions for Reliable Multiuser Databases

**Atomicity**  All-or-nothing: a transaction either runs to completion, or fails and leaves the database unchanged.

**Consistency**  Applying a transaction in a valid state of the database will always give a valid result state.

**Isolation**  Concurrent transactions have the same effect as sequential ones: the outcome is as if they were done in order.

**Durability**  Once a transaction is committed, it will not be rolled back.

Implementation of these is especially challenging for databases that are widely distributed and with multiple simultaneous users.
NoSQL

Not every database uses or needs SQL and the relational model.

For these, there is NoSQL — or, less dogmatically, Not Only SQL.

NoSQL databases can be highly effective in some application domains: with certain kinds of data, or needing high performance for one operation.

Sometimes the strong guarantees and powerful language of RDBMS simply aren’t needed, and alternatives do the job better.

Example NoSQL approaches

| Key-value | Column-oriented | Document-oriented | Graph databases |

Some of these weaken the ACID requirements – for example, offering only eventual consistency in exchange for greater decentralisation.

Balancing the tradeoffs here can be hard to assess, especially at extremes of size and speed. Strange things happen at scale.

e.g. Twitter Snowflake
Database Popularity

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.

<table>
<thead>
<tr>
<th>Rank</th>
<th>DBMS</th>
<th>Database Model</th>
<th>Score Feb 2018</th>
<th>Score Jan 2018</th>
<th>Score Feb 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oracle</td>
<td>Relational DBMS</td>
<td>1303.28</td>
<td>-38.66</td>
<td>-100.55</td>
</tr>
<tr>
<td>2</td>
<td>MySQL</td>
<td>Relational DBMS</td>
<td>1252.47</td>
<td>-47.24</td>
<td>-127.83</td>
</tr>
<tr>
<td>3</td>
<td>Microsoft SQL Server</td>
<td>Relational DBMS</td>
<td>1122.04</td>
<td>-26.03</td>
<td>-81.42</td>
</tr>
<tr>
<td>4</td>
<td>PostgreSQL</td>
<td>Relational DBMS</td>
<td>388.38</td>
<td>+2.19</td>
<td>+34.70</td>
</tr>
<tr>
<td>5</td>
<td>MongoDB</td>
<td>Document store</td>
<td>336.42</td>
<td>+5.47</td>
<td>+0.92</td>
</tr>
<tr>
<td>6</td>
<td>DB2</td>
<td>Relational DBMS</td>
<td>189.97</td>
<td>-0.30</td>
<td>+2.07</td>
</tr>
<tr>
<td>7</td>
<td>Microsoft Access</td>
<td>Relational DBMS</td>
<td>130.07</td>
<td>+3.37</td>
<td>-3.32</td>
</tr>
<tr>
<td>8</td>
<td>Redis</td>
<td>Key-value store</td>
<td>127.02</td>
<td>+3.88</td>
<td>+12.98</td>
</tr>
</tbody>
</table>

341 systems in ranking, February 2018

http://db-engines.com/en/ranking
Popularity of Different Kinds of Database

Ranking scores per category in percent, February 2018

Wide column stores 3.3%
Time Series DBMS 0.4%
Search engines 4.3%
Document stores 7.6%
Graph DBMS 1.2%
Key-value stores 4.2%
Native XML DBMS 0.3%
RDF stores 0.3%
Relational DBMS 78.1%

This chart shows the popularity of each category. It is calculated with the popularity (i.e. the ranking scores) of all individual systems per category. The sum of all ranking scores is 100%.

http://db-engines.com/en/ranking_categories
-- Find stars near a spot in the sky
FROM PhotoTag p, fGetNearbyObjEq(229.329,21.574,3) n
WHERE n.objID=p.objID AND p.type=3

-- How many stars can you see in the sky?
SELECT COUNT(*) FROM star
Students and Courses

<table>
<thead>
<tr>
<th>uun</th>
<th>name</th>
<th>age</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0456780</td>
<td>John</td>
<td>18</td>
<td>john@inf</td>
</tr>
<tr>
<td>s0378435</td>
<td>Helen</td>
<td>20</td>
<td>helen@phys</td>
</tr>
<tr>
<td>s0412375</td>
<td>Mary</td>
<td>18</td>
<td>mary@inf</td>
</tr>
<tr>
<td>s0189034</td>
<td>Peter</td>
<td>22</td>
<td>peter@math</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Title</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf1</td>
<td>Informatics 1</td>
<td>1</td>
</tr>
<tr>
<td>math1</td>
<td>Mathematics 1</td>
<td>1</td>
</tr>
<tr>
<td>geo1</td>
<td>Geology 1</td>
<td>1</td>
</tr>
<tr>
<td>dbs</td>
<td>Database Systems</td>
<td>3</td>
</tr>
<tr>
<td>adbs</td>
<td>Advanced Databases</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uun</th>
<th>code</th>
<th>mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0456780</td>
<td>inf1</td>
<td>71</td>
</tr>
<tr>
<td>s0412375</td>
<td>math1</td>
<td>82</td>
</tr>
<tr>
<td>s0412375</td>
<td>geo1</td>
<td>64</td>
</tr>
<tr>
<td>s0189034</td>
<td>math1</td>
<td>56</td>
</tr>
</tbody>
</table>
Simple Query

Extract all records for students older than 19.

\[
\text{SELECT } * \\
\text{FROM Student} \\
\text{WHERE age > 19}
\]

Returns a new table with the same schema as Student but only some of its rows.

\[
\{ \text{S } | \text{ S } \in \text{Student } \land \text{ S.age } > 19 \}
\]

<table>
<thead>
<tr>
<th>uun</th>
<th>name</th>
<th>age</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0378435</td>
<td>Helen</td>
<td>20</td>
<td>helen@phys</td>
</tr>
<tr>
<td>s0189034</td>
<td>Peter</td>
<td>22</td>
<td>peter@math</td>
</tr>
</tbody>
</table>
Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
  AND Takes.code = Course.code
  AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once,

<table>
<thead>
<tr>
<th>uun</th>
<th>name</th>
<th>age</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0456780</td>
<td>John</td>
<td>18</td>
<td>john@inf</td>
</tr>
<tr>
<td>s0378435</td>
<td>Helen</td>
<td>20</td>
<td>helen@phys</td>
</tr>
<tr>
<td>s0412375</td>
<td>Mary</td>
<td>18</td>
<td>mary@inf</td>
</tr>
<tr>
<td>s0189034</td>
<td>Peter</td>
<td>22</td>
<td>peter@math</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uun</th>
<th>code</th>
<th>mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0456780</td>
<td>inf1</td>
<td>71</td>
</tr>
<tr>
<td>s0412375</td>
<td>math1</td>
<td>82</td>
</tr>
<tr>
<td>s0189034</td>
<td>math1</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>code</th>
<th>title</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf1</td>
<td>Informatics 1</td>
<td>1</td>
</tr>
<tr>
<td>math1</td>
<td>Mathematics 1</td>
<td>1</td>
</tr>
<tr>
<td>geo1</td>
<td>Geology 1</td>
<td>1</td>
</tr>
<tr>
<td>dbs</td>
<td>Database Systems</td>
<td>3</td>
</tr>
<tr>
<td>adbs</td>
<td>Advanced Databases</td>
<td>4</td>
</tr>
</tbody>
</table>
Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
    AND Takes.code = Course.code
    AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test,

<table>
<thead>
<tr>
<th>uun</th>
<th>name</th>
<th>age</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0456780</td>
<td>John</td>
<td>18</td>
<td>john@inf</td>
</tr>
<tr>
<td>s0378435</td>
<td>Helen</td>
<td>20</td>
<td>helen@phys</td>
</tr>
<tr>
<td>s0412375</td>
<td>Mary</td>
<td>18</td>
<td>mary@inf</td>
</tr>
<tr>
<td>s0189034</td>
<td>Peter</td>
<td>22</td>
<td>peter@math</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uun</th>
<th>code</th>
<th>mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0456780</td>
<td>inf1</td>
<td>71</td>
</tr>
<tr>
<td>s0412375</td>
<td>math1</td>
<td>82</td>
</tr>
<tr>
<td>s0189034</td>
<td>math1</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>code</th>
<th>title</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf1</td>
<td>Informatics 1</td>
<td>1</td>
</tr>
<tr>
<td>math1</td>
<td>Mathematics 1</td>
<td>1</td>
</tr>
<tr>
<td>geo1</td>
<td>Geology 1</td>
<td>1</td>
</tr>
<tr>
<td>dbs</td>
<td>Database Systems</td>
<td>3</td>
</tr>
<tr>
<td>adbs</td>
<td>Advanced Databases</td>
<td>4</td>
</tr>
</tbody>
</table>
Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
    AND Takes.code = Course.code
    AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test,

<table>
<thead>
<tr>
<th>Student</th>
<th>Takes</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>uun</td>
<td>code</td>
<td>mark</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>s0456780</td>
<td>inf1</td>
<td>71</td>
</tr>
<tr>
<td>s0378435</td>
<td>inf1</td>
<td></td>
</tr>
<tr>
<td>s0412375</td>
<td>inf1</td>
<td></td>
</tr>
<tr>
<td>s0189034</td>
<td>inf1</td>
<td></td>
</tr>
<tr>
<td>s0412375</td>
<td>math1</td>
<td>82</td>
</tr>
<tr>
<td>s0189034</td>
<td>math1</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>code</th>
<th>title</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf1</td>
<td>Informatics 1</td>
<td>1</td>
</tr>
<tr>
<td>math1</td>
<td>Mathematics 1</td>
<td>1</td>
</tr>
<tr>
<td>geo1</td>
<td>Geology 1</td>
<td>1</td>
</tr>
<tr>
<td>dbs</td>
<td>Database Systems</td>
<td>3</td>
</tr>
<tr>
<td>adbs</td>
<td>Advanced Databases</td>
<td>4</td>
</tr>
</tbody>
</table>
Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
    AND Takes.code = Course.code
    AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

<table>
<thead>
<tr>
<th>uun</th>
<th>name</th>
<th>age</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0456780</td>
<td>John</td>
<td>18</td>
<td>john@inf</td>
</tr>
<tr>
<td>s0378435</td>
<td>Helen</td>
<td>20</td>
<td>helen@phys</td>
</tr>
<tr>
<td>s0412375</td>
<td>Mary</td>
<td>18</td>
<td>mary@inf</td>
</tr>
<tr>
<td>s0189034</td>
<td>Peter</td>
<td>22</td>
<td>peter@math</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uun</th>
<th>code</th>
<th>mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0456780</td>
<td>inf1</td>
<td>71</td>
</tr>
<tr>
<td>s0412375</td>
<td>math1</td>
<td>82</td>
</tr>
<tr>
<td>s0189034</td>
<td>math1</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>code</th>
<th>title</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf1</td>
<td>Informatics 1</td>
<td>1</td>
</tr>
<tr>
<td>math1</td>
<td>Mathematics 1</td>
<td>1</td>
</tr>
<tr>
<td>geo1</td>
<td>Geology 1</td>
<td>1</td>
</tr>
<tr>
<td>dbs</td>
<td>Database Systems</td>
<td>3</td>
</tr>
<tr>
<td>adbs</td>
<td>Advanced Databases</td>
<td>4</td>
</tr>
</tbody>
</table>
**Example Query**

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
    AND Takes.code = Course.code
    AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

<table>
<thead>
<tr>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>mary@inf</td>
</tr>
<tr>
<td>Peter</td>
<td>peter@math</td>
</tr>
</tbody>
</table>
Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
    AND Takes.code = Course.code
    AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Expressed in tuple relational calculus:

\[
\{ R \mid \exists S \in \text{Student}, T \in \text{Takes}, C \in \text{Course} .
\quad R.\text{name} = S.\text{name} \land R.\text{email} = S.\text{email} \land S.\text{uun} = T.\text{uun}
\quad \land T.\text{code} = C.\text{code} \land C.\text{title} = "Mathematics 1" \}
\]
Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
  AND Takes.code = Course.code
  AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Implemented in relational algebra,

\[
π_{\text{name}, \text{email}}(σ_{\text{Student.uun} = \text{Takes.uun} ∧ \text{Takes.code} = \text{Course.code} ∧ \text{Course.name} = "Mathematics 1"}(\text{Student} \times \text{Takes} \times \text{Course}))
\]
Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
  AND Takes.code = Course.code
  AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Implemented in relational algebra, in several possible ways:

\[
\pi_{\text{name}, \text{email}}(\sigma_{\text{title}=\text{"Mathematics 1"}}(\text{Student} \bowtie \text{Takes} \bowtie \text{Course}))
\]

\[
\pi_{\text{name}, \text{email}}(\text{Student} \bowtie (\text{Takes} \bowtie (\sigma_{\text{title}=\text{"Mathematics 1"}}(\text{Course}))))
\]
Nested Query

As **SELECT** both takes in and produces tables, we can use the result of one query in building another.

```
SELECT Student.name, Student.email
FROM Student, Takes, MathCourse
WHERE Student.uun=Takes.uun AND Takes.code=MathCourse.code
```
Nested Query

As **SELECT** both takes in and produces tables, we can use the result of one query in building another.

```sql
SELECT Student.name, Student.email
FROM Student, Takes,
     (SELECT code FROM Course WHERE title='Mathematics 1') AS C
WHERE Student.uun=Takes.uun AND Takes.code=C.code
```

Inner query **(SELECT code ... ) AS C** computes a table of course codes.

Adding nested queries does not change the expressive power of SQL; but it may make some queries more succinct, or easier to understand.

Of course, as usual, the execution plan used by a RDBMS to compute the query is quite independent of whether we use nested queries or not — it will rearrange and rewrite as necessary to reduce computation cost.
Travelling Salesman

Given a table of travel times between all $n^2$ pairs of towns:

What is the quickest route to visit them all?
Is there any route shorter than $X$?

Checking any route is fast.

However, there are a lot of routes to check ($n!$).

Can we do it faster?

Not much, no. The fastest algorithm known takes time related to $2^n$.

This exponential growth means that a small problem can quickly become very large.
This travelling salesman problem is more widely applicable than the name suggests, and it also arises in:

- Commercial distribution logistics;
- Optimising chip layout;
- Assembling DNA sequence fragments; etc.
The travelling salesman problem is **NP-hard**: one of a large class of equivalent computational challenges where:

- Checking a potential solution is quick...
- ...but there are a lot of potential solutions...
- ...and we don’t know how to do it any faster.

There’s a bounty on these. You find a fast way to solve travelling salesman, or a proof that there isn’t one, and you collect $1M.

Apply to the *Clay Mathematics Institute*, 10 Memorial Boulevard, Providence, Rhode Island, USA.
P.S. If you are an actual salesman:

Distances based on real space simplify the general problem;

There are fast algorithms which with very high probability return an answer very close to the optimum.

Other NP-hard problems are not be so easy to work around.
Limits of Computation

Various physical factors set upper bounds on how much computation is possible:

- The speed of light, $c$;
- Planck’s constant, $\hbar$;
- Universal gravitational constant, $G$;
- Quantisation of energy states;
- Heisenberg uncertainty in observation;
- Mass/energy equivalence.

For a 1kg computer, this sets a limit on computation of $10^{50}$ operations per second on $10^{31}$ bits.

Working at $10^9$K, “the ultimate laptop looks like a small piece of the big bang”.

Matter organised to provide the greatest possible computing power is fancifully known as computronium.

In the 1960’s Hans-Joachim Bremermann was one of the first people to estimate upper limits to computation.

The Bremermann limit is the computation which could be performed using the earth, over the period of its existence so far.

This is around $10^{93}$ bits of computation.

That’s enough to solve the travelling salesman problem for 300 cities.

But just the once.
Find the names of all students who are taking either Informatics 1 or Mathematics 1.

```
SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.uun = T.uun AND T.code = C.code
    AND (C.title = 'Informatics 1'
         OR C.title = 'Mathematics 1')
```
Disjunction Query

Find the names of all students who are taking either Informatics 1 or Mathematics 1.

```
SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.uun = T.uun AND T.code = C.code
  AND C.title = 'Informatics 1'

UNION

SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.uun = T.uun AND T.code = C.code
  AND C.title = 'Mathematics 1'
```
Conjunction Query

Find the names of all students who are taking both Informatics 1 and Mathematics 1.

```sql
SELECT S.name
FROM Student S, Takes T1, Course C1, Takes T2, Course C2
WHERE S.uun = T1.uun AND T1.code = C1.code
  AND S.uun = T2.uun AND T2.code = C2.code
  AND C1.title = 'Informatics 1'
  AND C2.title = 'Mathematics 1'
```
Conjunction Query

Find the names of all students who are taking both Informatics 1 and Mathematics 1.

\[
\text{SELECT } S\.name \\
\text{FROM Student S, Takes T, Course C} \\
\text{WHERE } S\.uun = T\.uun \text{ AND T.code} = \text{C.code} \text{ AND C.title = 'Informatics 1'}
\]

\[
\text{INTERSECT}
\]

\[
\text{SELECT } S\.name \\
\text{FROM Student S, Takes T, Course C} \\
\text{WHERE } S\.uun = T\.uun \text{ AND T.code} = \text{C.code} \text{ AND C.title = 'Mathematics 1'}
\]
Difference Query

Find the names of all students who are taking Informatics 1 \textit{but not} Mathematics 1.

\begin{verbatim}
SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.uun = T.uun AND T.code = C.code
    AND C.title = 'Informatics 1'

EXCEPT

SELECT S.name
FROM Student S, Takes T, Course C
WHERE S.uun = T.uun AND T.code = C.code
    AND C.title = 'Mathematics 1'
\end{verbatim}
Comparison Query

Find the students' names in all cases where one person scored higher than another in Mathematics 1.

```
SELECT S1.name AS "Higher", S2.name AS "Lower"
FROM Student S1, Takes T1, Student S2, Takes T2, Course C
WHERE S1.uun = T1.uun AND T1.code = C.code
    AND S2.uun = T2.uun AND T2.code = C.code
    AND C.title = 'Informatics 1'
    AND T1.mark > T2.mark
```

<table>
<thead>
<tr>
<th>Higher</th>
<th>Lower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Peter</td>
</tr>
</tbody>
</table>
Aggregates: Operations on Multiple Values

SQL includes a range of mathematical operations on individual values, like $T1.mark > T2.mark$.

SQL also provides operations on whole collections of values, as returned in a `SELECT` query. There are five of these standard aggregate operations:

- **`COUNT(val)`**: The number of values in the `val` field
- **`SUM(val)`**: The total of all values in the `val` field
- **`AVG(val)`**: The mean of all values in the `val` field
- **`MAX(val)`**: The greatest value in the `val` field
- **`MIN(val)`**: The least value in the `val` field

The number of distinct values in the `val` field
The total of the distinct values in the `val` field
The mean of the distinct values in the `val` field

Particular RDBMS implementations may refine and extend these with other operations.
Aggregates: Operations on Multiple Values

SQL includes a range of mathematical operations on individual values, like $T1\text{.mark} > T2\text{.mark}$.

SQL also provides operations on whole collections of values, as returned in a `SELECT` query. There are five of these standard aggregate operations:

- **COUNT**(DISTINCT `val`)  
  The number of distinct values in the `val` field
- **SUM**(DISTINCT `val`)  
  The total of the distinct values in the `val` field
- **AVG**(DISTINCT `val`)  
  The mean of the distinct values in the `val` field
- **MAX**(`val`)  
  The greatest value in the `val` field
- **MIN**(`val`)  
  The least value in the `val` field

Particular RDBMS implementations may refine and extend these with other operations.
Aggregating Query

Find the number of students taking Informatics 1, their mean mark, and the highest mark.

```
SELECT COUNT(DISTINCT T.uun) AS "Number",
       AVG(T.mark) AS "Mean Mark",
       MAX(T.mark) AS "Highest"
FROM Student S, Takes T, Course C
WHERE S.uun = T.uun AND T.code = C.code
  AND C.title = 'Informatics 1'
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Mean Mark</th>
<th>Highest</th>
</tr>
</thead>
<tbody>
<tr>
<td>263</td>
<td>65.66</td>
<td>100</td>
</tr>
</tbody>
</table>
Who Writes SQL?

SQL is one of the world’s most widely used programming languages, but programs in SQL come from many sources. For example:

- Hand-written by a programmer
- Generated by some interactive visual tool
- Generated by an application to fetch an answer for a user
- Generated by one program to request information from another

Most SQL is written by programs, not directly by programmers.

The same is true of HTML, another domain-specific language.

Also XML, Postscript, ...
Homework

Explore SkyServer, its different types of search, and try some SQL yourself. Work through at least the first three pages of the SQL Tutorial there.

http://skyserver.sdss.org/en

Finally: What about http://is.gd/locatepluto?
Summary

ACID Properties


Characteristics that enable reliable use of *transactions*. Challenging to implement for widely distributed databases that serve many users.

More Queries

Nested queries: using one result table as input to another query.

Combining tables with **UNION**, **INTERSECT** and **EXCEPT**.

Aggregate Operations

SQL provides arithmetic operations and comparisons to use within queries. Aggregate operators take all the values in a multiset of results and combine them together.