Structured Query Language

Slides based on “Database Management Systems” 3rd ed, Ramakrishnan and Gehrke
Loading Oracle tables

- Need to copy the tables directory from the class home at /data/htdocs/cs630 to your own tables directory under your own cs630 directory, then execute dbbook.sh:

- Login to topcat, then use Linux commands:
  
  cd cs630
  
  cp –r /data/htdocs/cs630/tables .
  
  cd tables
  
  sh dbbook.sh

- To drop these tables:
  
  sqlplus user/user@//dbs3.cs.umb.edu/dbs3< dropdb.sql
Relational Query Language: SQL

- Supports simple, yet powerful querying of data.
  - Precise semantics for relational queries.
  - DML (Data Manipulation Language)
  - DDL (Data Definition Language)
- SQL developed by IBM (system R) in the 1970s
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision) : third and most important standard
    - Early enough to affect Oracle, DB2, other important commercial databases, so the real common ground.
  - SQL-99 (major extensions, triggers, recursive queries)
SQL’s Most Important Data Types

- **Character strings**
  - `CHAR(n)`, `VARCHAR(n)`: fixed and variable-length strings
    - We will stick to ASCII characters here to avoid problems
    - But note that Unicode is supported now by Oracle, mysql, ... and SQL 2003

- **Numerical**:
  - `INTEGER (INT)` Usually 32-bit, but not guaranteed by SQL-92.
  - Floating point: `FLOAT` or `DOUBLE PRECISION`: Usually 64-bit IEEE floating point, but not guaranteed by SQL-92.
    - Oracle: use `BINARY_DOUBLE` for IEEE format.
  - Fixed precision: `DECIMAL(n,d)`
    - 1234.56 is of type `DECIMAL(6,2)`, precision 6, scale 2

- **DATE and TIME and TIMESTAMP**
  - Not used in R&G, see coverage pp. 262-269 in Murach
  - Timezones a challenge: SQL-92 says DB uses GMT in its data, but mysql does this only for TIMESTAMP
### SQL 2003 Data Types, from http://www.w3resource.com/sql/data-type.php, with notes in color

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER(n) or CHAR(n)</td>
<td>Character string, fixed length n. A string of text in an implementer-defined format. The size argument is a single nonnegative integer that refers to the maximum length of the string. Values for this type must enclosed in single quotes. <strong>Character sets: another topic.</strong></td>
</tr>
<tr>
<td>CHARACTER VARYING(n) or VARCHAR(n)</td>
<td>Variable length character string, maximum length n.</td>
</tr>
<tr>
<td>BINARY(n)</td>
<td>Fixed length binary string, maximum length n. <strong>Not in SQL-92, but BIT(n) there.</strong></td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>Stores truth values - either TRUE or FALSE. <strong>Not in SQL-92</strong></td>
</tr>
<tr>
<td>BINARY VARYING(n) or VARBINARY(n)</td>
<td>Variable length binary string, maximum length n. <strong>BIT VARYING in SQL-92.</strong></td>
</tr>
</tbody>
</table>
# SQL 2003 Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>INTEGER(p)</code></td>
<td>Integer numerical, precision p. Not in SQL-92 with (p). MySQL: p means display size, not precision</td>
</tr>
<tr>
<td><code>SMALLINT</code></td>
<td>Integer numerical precision 5. SQL-92: precision is implementation dependent.</td>
</tr>
<tr>
<td><code>INTEGER</code></td>
<td>Integer numerical, precision 10. It is a number without decimal point with no digits to the right of the decimal point, that is, with a scale of 0. SQL-92: precision is implementation dependent.</td>
</tr>
</tbody>
</table>
# SQL 2003 Data Types

| DECIMAL(p, s) | Exact numerical, precision p, scale s. A decimal number, that is number that can have a decimal point in it. The size argument has two parts: precision and scale. The scale can not exceed the precision. Precision comes first, and a comma must separate from the scale argument. How many digits the number is to have - a precision indicates that and maximum number of digits to the right of decimal point have, that indicates the scale. |
| NUMERIC(p, s) | Exact numerical, precision p, scale s. (Same as DECIMAL). |
### SQL 2003 Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOAT(p)</td>
<td>Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision.</td>
</tr>
<tr>
<td>REAL</td>
<td>Approximate numerical mantissa precision 7. <strong>Better to use</strong> FLOAT.</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Approximate numerical mantissa precision 16. Usually IEEE Standard floating point, but not guaranteed by the SQL standard. Oracle uses NUMBER for FLOAT, use BINARY_DOUBLE for IEEE format, like Java double.</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>Approximate numerical mantissa precision 16. Same as FLOAT.</td>
</tr>
</tbody>
</table>
## SQL 2003 Data Types

<table>
<thead>
<tr>
<th>DATA TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
</table>
| DATE TIME | Composed of a number of integer fields, representing an absolute point in time, i.e GMT or UTC. (Timezone is a “session variable”)
| TIMESTAMP | Composed of a number of integer fields, representing an absolute point in time, i.e GMT or UTC. (Timezone is a “session variable”)
| INTERVAL | Composed of a number of integer fields, representing a period of time, depending on the type of interval.
| COLLECTION (ARRAY, MULTISSET) | ARRAY (offered in SQL99) is a set-length and ordered collection of elements, MULTISET (added in SQL2003) is a variable-length and unordered collection of elements. Both the elements must be of a predefined datatype.
| Not in SQL-92 | COLLECTION (ARRAY, MULTISSET) | ARRAY (offered in SQL99) is a set-length and ordered collection of elements, MULTISET (added in SQL2003) is a variable-length and unordered collection of elements. Both the elements must be of a predefined datatype.
| Not in SQL-92 | COLLECTION (ARRAY, MULTISSET) | ARRAY (offered in SQL99) is a set-length and ordered collection of elements, MULTISET (added in SQL2003) is a variable-length and unordered collection of elements. Both the elements must be of a predefined datatype.
| XML | Stores XML data. It can be used wherever a SQL datatype is allowed, such as a column of a table. | XML | Stores XML data. It can be used wherever a SQL datatype is allowed, such as a column of a table. |
**Example Schema**

**Students**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Smith</td>
<td>20</td>
</tr>
<tr>
<td>53650</td>
<td>Jones</td>
<td>25</td>
</tr>
<tr>
<td>53681</td>
<td>Adams</td>
<td>22</td>
</tr>
</tbody>
</table>

**Courses**

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>114</td>
<td>Calculus</td>
<td>M123</td>
</tr>
<tr>
<td>115</td>
<td>Databases</td>
<td>M234</td>
</tr>
</tbody>
</table>

**Enrolled**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>114</td>
<td>A</td>
</tr>
<tr>
<td>53650</td>
<td>115</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>115</td>
<td>B</td>
</tr>
</tbody>
</table>
Creating Relations in SQL

CREATE TABLE Students
(sid CHAR(20),
  name CHAR(20),
  login CHAR(10),
  age INTEGER,
  gpa REAL);

CREATE TABLE Enrolled
(sid CHAR(20),
  cid CHAR(20),
  grade CHAR(2));
Destroying and Altering Relations

**DROP TABLE Students;**  

- Deletes relation *Students*, including schema information *and* all the tuples

**ALTER TABLE Students**  

  **ADD firstYear INTEGER;**

- Add new column to schema
- Every tuple is extended with **NULL** value in added field
- Default value may be specified instead of **NULL**
Structure of SQL SELECT Query

Structure:

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
```

- **relation-list** = list of relation names
  - possibly with a *range-variable* (AKA correlation name or table alias) after each name
- **target-list** = list of attributes of relations in *relation-list*
- **qualification** = conditions \( Attr \ op \ const \) or \( Attr1 \ op \ Attr2 \)
  - \( op \) is one of \( <, >, =, >=, <=, <> \), or string operators
  - Expressions connected using AND, OR and NOT
- **DISTINCT** = optional, eliminates duplicates
  - By default duplicates are NOT eliminated!
Conceptual Evaluation Strategy

- **Semantics of SQL query**
  1. Compute the cross-product of relation-list
  2. Discard resulting tuples if they fail qualifications
  3. Delete attributes that are not in target-list
  4. If DISTINCT is specified, eliminate duplicate rows

- This strategy is least efficient way to compute a query!
  - Optimizer finds efficient strategies to compute the same result
### Example Schema

#### Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

#### Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>name</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>clipper</td>
<td>green</td>
</tr>
</tbody>
</table>

#### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
### Conceptual Evaluation Example

```sql
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid = 103
```

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
A Note on Range Variables

- Really needed only if the same relation appears twice in the **FROM** clause (SELECT … FROM Sailors S1, Sailors S2)

  ```sql
  SELECT  S.sname
  FROM    Sailors S, Reserves R
  WHERE   S.sid=R.sid AND R.bid=103
  ```

  It is good style, however, to use range variables always!

  Instead of …

  ```sql
  SELECT  sname
  FROM    Sailors, Reserves
  WHERE   Sailors.sid=Reserves.sid AND bid=103
  ```
Duplicate Tuples and DISTINCT

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
```

- Would adding **DISTINCT** to this query make a difference?
- What is the effect of replacing `S.sname` by `S.sid` in the **SELECT** clause?
- Would adding **DISTINCT** to this variant of the query make a difference?
Expressions and Strings

“Find rating and number of years to retirement for sailors whose names begin with ‘d’, end with ‘n’ and contain at least three characters”

```
SELECT S.rating, 60 - S.age AS Yr_to_retire
FROM Sailors S
WHERE S.sname LIKE 'd_%n'
```

AS allows to (re)name fields in result.

LIKE is used for string matching

_ stands for any one character

% stands for 0 or more arbitrary characters

Mysql also has RLIKE, with regular expression matching
Expressions and Strings - Example

SELECT  S.rating, 60 - S.age AS Yr_to_retire
FROM  Sailors S
WHERE  S.sname LIKE 'd_%n'

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>Yr_to_retire</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>15</td>
</tr>
</tbody>
</table>
Set Operations

- **UNION**
  - compute the union of any two *union-compatible* sets of tuples

- **INTERSECT**
  - compute the intersection of any two *union-compatible* sets of tuples (not supported in MySQL)

- **EXCEPT or MINUS**
  - Set difference of any two *union-compatible* sets of tuples (not supported in MySQL)

- Duplicates eliminated by default!
  - UNION ALL, INTERSECT ALL, EXCEPT ALL retain duplicates
  - Contrast with non-set SQL operations
Adding and Deleting Tuples

- Insert single tuple

  \[
  \text{INSERT INTO Students (sid, name, login, age, gpa)}
  \text{VALUES (’53688’, ’Smith’, ’smith@ee’, 18, 3.2)};
  \]

- Delete all tuples satisfying condition

  \[
  \text{DELETE FROM Students S WHERE S.name = ’Smith’};
  \]
Data Modifications: Inserts

```
INSERT INTO Table (attr1, attr2, …)
VALUES (val1, val2, …);
```

- Values and attribute domains must match
- Attributes not specified will be assigned value NULL

- Variation: insert tuples returned by SELECT

```
INSERT INTO Table (attr1, attr2, …)
SELECT col1, col2, …
FROM …
[WHERE …
GROUP BY …
HAVING … ];
```
Data Modifications: Updates

- No new tuples created
- Attribute values of existing tuples modified
  ```sql
  UPDATE Table
  SET attr1=expression1, attr2=expression2 [...] WHERE condition;
  ```
- Values and attribute domains must match
- It is possible to use subqueries:
  ```sql
  UPDATE Table
  SET attr1= (SELECT value1
              FROM ...
              WHERE ...) WHERE condition;
  ```
Integrity Constraints (ICs)

- **IC**: condition that must hold for any instance of the database; e.g., *domain constraints*
  - Specified when schema is defined.
  - Checked when relations are modified.
- A *legal* instance satisfies all specified ICs
  - It is the DBMS’s role to enforce IC
- **ICs we study**
  - Primary key constraints
  - Foreign key constraints: referential integrity
  - Unique constraints
  - NOT NULL constraints
Primary and Candidate Keys in SQL

- Primary keys specified by keyword PRIMARY KEY
- Candidate keys specified by keyword UNIQUE

Distinctions between the two:
- Any attribute in the primary key is NOT allowed to have NULL values
- Primary key attributes may have special roles in the DBMS internals (although from the logical point of view is same as unique)

Declaration
- In-line with the respective attribute (column constraint)
  - Only if one-attribute key!
- Or as separate constraint line (table constraint)
### Keys in SQL - Examples

#### Schema and Instance

**Students**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Smith</td>
<td>20</td>
</tr>
<tr>
<td>53650</td>
<td>Jones</td>
<td>25</td>
</tr>
<tr>
<td>53681</td>
<td>Adams</td>
<td>22</td>
</tr>
</tbody>
</table>

**Courses**

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>114</td>
<td>Calculus</td>
<td>M123</td>
</tr>
<tr>
<td>115</td>
<td>Databases</td>
<td>M234</td>
</tr>
</tbody>
</table>

**Enrolled**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>114</td>
<td>A</td>
</tr>
<tr>
<td>53650</td>
<td>115</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>115</td>
<td>B</td>
</tr>
</tbody>
</table>
For a given student and course, there is a single grade.

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid))

Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade.

CREATE TABLE Enrolled
(sid CHAR(20) PRIMARY KEY,
cid CHAR(20),
grade CHAR(2),
UNIQUE (cid, grade) )
Foreign Keys, Referential Integrity

- **Foreign key**
  - Set of fields in relation A that refer to a tuple in relation B
  - Must correspond to primary key of relation B (or UNIQUE)

- Not necessary for field names in A and B to be the same!!!

  FOREIGN KEY (attr1) REFERENCES B (attr2)

  ✓ For mysql, we always need “(attr2)” or whatever
  ✓ For Oracle, we can drop “(attr2)” if it’s the primary key of B

- E.g. sid in Enrolled is a foreign key referring to Students:
  - Enrolled(sid: string, cid: string, grade: string)

- **Referential integrity** is achieved by enforcing all foreign keys
  - no “dangling references”
Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

```
CREATE TABLE Enrolled (
    sid CHAR(20), cid CHAR(20), grade CHAR(2),
    PRIMARY KEY (sid,cid),
    FOREIGN KEY (sid) REFERENCES Students(sid)
)
```

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>114</td>
<td>A</td>
</tr>
<tr>
<td>53650</td>
<td>115</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>115</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Smith</td>
<td>20</td>
</tr>
<tr>
<td>53650</td>
<td>Jones</td>
<td>25</td>
</tr>
<tr>
<td>53681</td>
<td>Adams</td>
<td>22</td>
</tr>
</tbody>
</table>
### Example Schema

#### Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

#### Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>name</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>clipper</td>
<td>green</td>
</tr>
</tbody>
</table>

#### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Query Example 1

“Find names of sailors who’ve reserved boat #103”

SELECT ??
Query Example 1

“Find names of sailors who’ve reserved boat #103”

```sql
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid = 103
```
“Find names of sailors who’ve reserved a red boat”

```
SELECT ??
```
Query Example 2

- “Find names of sailors who’ve reserved a red boat”

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
AND B.color = 'red';
```

### Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
</table>

### Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>name</th>
<th>color</th>
</tr>
</thead>
</table>

### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
</table>