SQL History

- The Structured English QUEry Language, or SEQUEL, was developed by Donald Chamberlain and Raymond Boyce as part of the System R project at the IBM San Jose Research Laboratory, first appearing in 1971.

- For existing trademark reasons, it was later renamed Structured Query Language (SQL), but still pronounced “SEE-kwul”

- ANSI and ISO standards emerged and evolved:

- Commercial systems offer most, if not all, SQL–92 features, plus varying feature sets from later standards and special proprietary features.

- **Note:** Some examples we discuss or in the text may work on your particular system.

- SQL is used for
  - Creating databases
  - Adding, changing, or deleting database structures
  - Adding, modifying, or deleting database records (tuples)
  - Querying databases (information retrieval)

- Four parts
  - Data Definition language (DDL)
  - Data Manipulation language (DML)
  - Data Control language (DCL)
  - Transaction Control Language (TCL)

SQL DDL

- SQL Data types
Instructors setup your database

- **CREATE DATABASE** dbname;
- **GRANT ...** to give you and us access

```sql
create database someones_db;
grant all on someones_db.* to someones identified by 'qwerty123';
grant all on someones_db.* to ccpalmer;
grant all on someones_db.* to jenkins;
```

That way only you and the instructors can **USE** your database.

For example:

```
SHOW GRANTS FOR Saisi;
```

Once you have a database, you connect to it.

* via ssh and terminal mode - typical use

```bash
$ ssh to **not sunapee**.cs.dartmouth.edu from somewhere.
$ mysql --user=ccpalmer --password --host=sunapee.cs.dartmouth.edu ccptestdb
> (that did the USE for you)
> /* run your MySQL commands, e.g. */
> SHOW TABLES;
```

```
+----------------------+
<table>
<thead>
<tr>
<th>Tables_in_ccptestdb</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseballSalaries2003</td>
</tr>
<tr>
<td>EMPLOYEE</td>
</tr>
<tr>
<td>Institution</td>
</tr>
<tr>
<td>advisor</td>
</tr>
<tr>
<td>animal</td>
</tr>
<tr>
<td>baseball</td>
</tr>
<tr>
<td>classroom</td>
</tr>
<tr>
<td>course</td>
</tr>
<tr>
<td>department</td>
</tr>
<tr>
<td>dept</td>
</tr>
<tr>
<td>faculty</td>
</tr>
<tr>
<td>instructor</td>
</tr>
<tr>
<td>prereq</td>
</tr>
<tr>
<td>section</td>
</tr>
<tr>
<td>student</td>
</tr>
<tr>
<td>takes</td>
</tr>
<tr>
<td>teaches</td>
</tr>
<tr>
<td>testball</td>
</tr>
<tr>
<td>time_slot</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
+----------------------+
```
Remain logged into MySQLWorkBench

- SHOW TABLES;
- Open SQL Script file (sandbox/soccer.sql)
- CREATE TABLE
  - NOT NULL specification
  - UNIQUE
  - REFERENCES
- INSERT INTO ... VALUES ...
- Gotta be careful though - suppose we had this:

```
CREATE TABLE College
(
  cName  varchar(20) UNIQUE,
  state  varchar(2),
  enr    numeric(5,0)
);
```

That wouldn’t work for two OSU’s “Oklahoma State Univ” and “Ohio State Univ”.

- SELECT

```
SELECT A1, A2, ..., An
FROM R1, R2, ..., Rm
WHERE condition ;
```

- back to the example ... do several SELECT S
  - including SELECT *

```
SELECT * FROM Tryout;
SELECT * FROM Player WHERE pName="David";
```

- DISTINCT eliminates duplicates

```
SELECT * FROM Player WHERE pName="David";
SELECT DISTINCT * FROM Player WHERE pName="David";
```
NULL s are handled together, sometimes rising to top or sinking to bottom of list

- **ORDER BY** will sort the results

```sql
SELECT * FROM Tryout WHERE decision = "yes" ORDER BY pPos;
SELECT * FROM Player ORDER BY pName ASC, HS DESC;
```

- **Special Operators**
  - LIKE

```sql
SELECT * FROM College WHERE cName LIKE "%SU";
```

- BETWEEN, IS NULL, IN, EXISTS

- **AND, OR, NOT**

```sql
SELECT * FROM Player WHERE HS < 1000;
SELECT * FROM Player WHERE HS < 1000 AND yCard = "yes";
```

- **Arithmetic and Aggregate functions**
  - Other aggregates include MIN, MAX, SUM, and AVG

```sql
SELECT COUNT(yCard) AS NumYellowCards FROM Player WHERE HS > 999 AND yCard="yes";
```

- **Renaming with AS**

```sql
SELECT DISTINCT pName AS CityPlayer FROM Player WHERE HS > 1000;
```

- **Sub Queries**

```sql
SELECT DISTINCT pName FROM Player WHERE pID NOT IN (SELECT pID FROM Tryout WHERE decision="yes");
```

- **GROUP BY and HAVING**

```sql
/* How many goalie tryouts did each college have */
SELECT cName, COUNT(cName) FROM Tryout WHERE pPos = "goalie" GROUP BY cName;
/* How many goalie tryouts did each college have */
SELECT cName, COUNT(cName) FROM Tryout WHERE pPos = "goalie" GROUP BY cName HAVING COUNT(cName) < 2;
```
GROUP BY is frequency distributions created by GROUP BY clause within SELECT statement
- HAVING is extension of GROUP BY feature
- Applied to output of GROUP BY operation
- Used in conjunction with GROUP BY clause in second SQL command set
- Similar to WHERE clause in SELECT statement

Commit

EXPLAIN

Changing a table’s data

- UPDATE

Changing a table’s DESIGN

```
ALTER TABLE tablename
{ADD | MODIFY} { columnname datatype [ {ADD | MODIFY} columnname datatype ] } ;
```

```
ALTER TABLE tablename
ADD constraint [ ADD constraint ] ;
```

Primary & Foreign Keys

We know keys have to be non-NULL and UNIQUE.
Just add PRIMARY KEY (attr) at end of CREATE ;

```
CREATE TABLE Tryout
( pID numeric(5,0),
cName varchar(20),
pPos varchar(8),
decision varchar(3),
PRIMARY KEY (pID),
FOREIGN KEY (cName) REFERENCES College(cName)
) ;
```

- See the SQL source
- Then try adding pID as a PRIMARY KEY to Player and recreate and initialize the tables... should get duplicate key error on pID=40004.
- Delete the Referencing key first, and then it works.

1. Lecture notes based on texts by Coronel, Widom, Ullman, Jukic, and Silberschatz. ↩