Databases: SQL

IAT 352: Internet Computing Technologies

Moh Rajabi Seraji
Simon Fraser University

Querying Relational Data

- SQL – the most popular query language
- Query is a question about the data
- The answer consists of a **new relation** containing the result

```
SELECT *
FROM Students S
WHERE S.GPA > 3.2

SELECT S.Login, S.GPA
FROM Students S
WHERE S.GPA > 3.2
```

Sid	Name	Login	Age	GPA
53666	Chu	chu@cs.com	18	3.4
53688	Jones	jones@ee.com	18	3.2
53650	Jones	jones@math.com	19	3.8
53831	Madayan	madayan@music.com	11	1.8
53832	Guldu	guldu@music.com	12	2.0

Sid	Name	Login	Age	GPA
53666	Chu	chu@cs.com	18	3.4
53650	Jones	jones@math.com	19	3.8

Login	GPA
chu@cs.com	3.4
jones@math.com	3.8

CRUD Operations

- Create, Read, Update, Delete
- **Create** Database, Create Table, Insert records
- **Read**, i.e. Select records (and attributes) that satisfy the criteria
- **Update** values in existing records
- **Delete** records, tables, databases

- SQL provides commands for all of the above
- SQL commands can be executed via command line, DB admin client, or send to DB server from the application

Typical Web Application Architecture

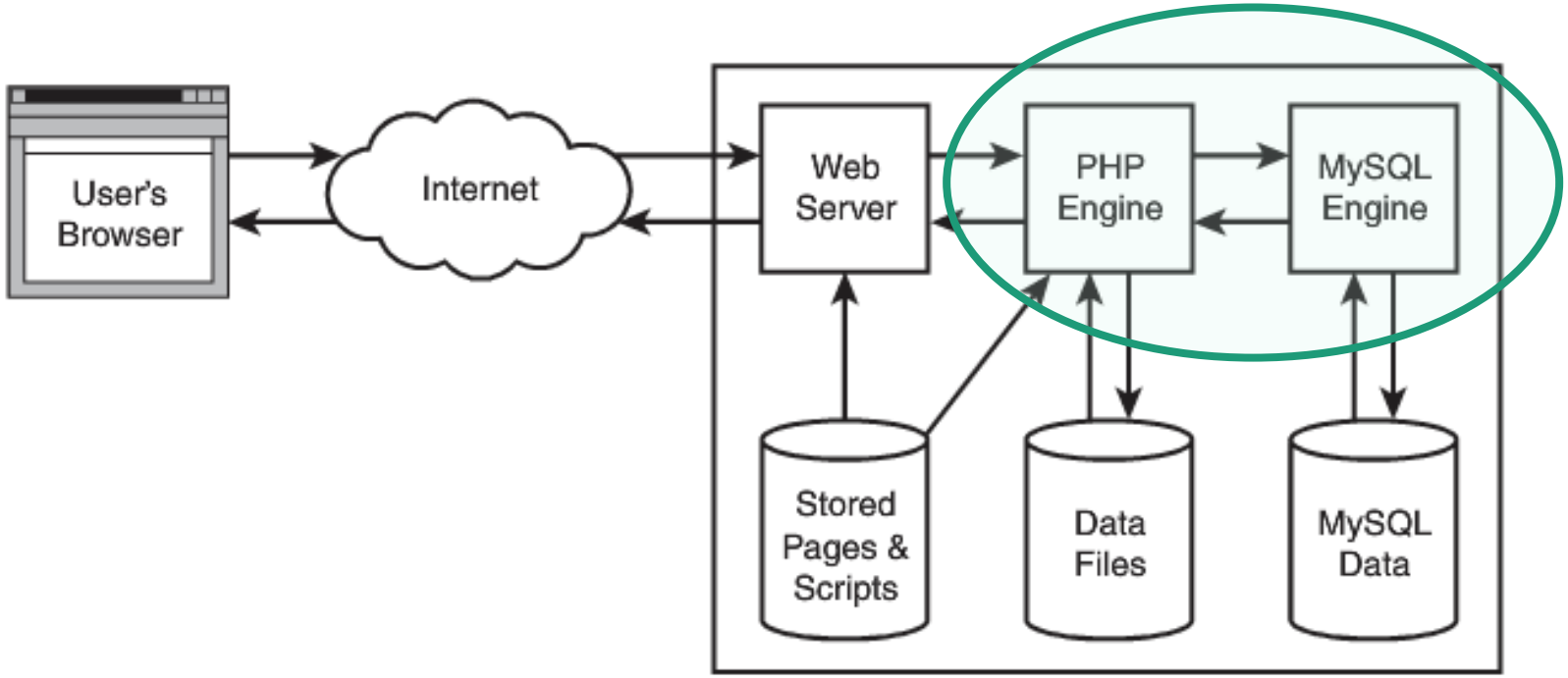


Figure 18.1 User information is stored or processed by these elements of a typical web application environment.

The WORLD Database (sample DB from MySQL)

- Country: 239 entries
- City: 4,079 entries
- Country Language: 984 entries

CREATE-USE Syntax

CREATE DATABASE <database name>;
CREATE SCHEMA <database name>;

- **CREATE SCHEMA world;**
- **USE world;**

- Typically, the create database statement is used via admin interface, rather then called during runtime
- Use statement selects the database the queries are directed to. It is typically a part of establishing a connection.

CREATE TABLE Syntax

CREATE TABLE <table>
(<column definitions>)

table = table name
definitions = list of column definitions

- Typically, preset via admin interface, such as phpMyAdmin. Rarely from application during the runtime.

WORLD: Example

```
DROP SCHEMA IF EXISTS world;
CREATE SCHEMA world;
USE world;
SET AUTOCOMMIT=0;

--
-- Table structure for table `city`
--

DROP TABLE IF EXISTS `city`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `city` (
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Name` CHAR(35) NOT NULL DEFAULT '',
  `CountryCode` CHAR(3) NOT NULL DEFAULT '',
  `District` CHAR(20) NOT NULL DEFAULT '',
  `Population` INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`),
  CONSTRAINT `city_ibfk_1` FOREIGN KEY (`CountryCode`) REFERENCES `country` (`Code`)
) ENGINE=InnoDB AUTO_INCREMENT=4080 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
```

DROP: creating database that exists would cause an error

CREATE Table: attributes, their types, keys

Primary Key for this table

CountryCode is a foreign key that REFERENCES table country and attr. Code

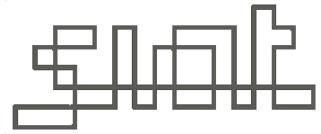
Oh, do I need to write all this?

- No, GUI for tools like phpMyAdmin provides easy to use interface to establish cross links
- When creating tables, you need to know
  - What attributes are,
  - What are the types and ranges for attribute values, so you can set proper types and sizes for attributes
  - What the keys are.

Data types

- TINYINT [(length)]
- INT [(length)]
- BIGINT [(length)]
- DOUBLE [(length,decimals)]
- DATE
- CHAR [(length)], VARCHAR [(length)]
- TEXT [BINARY]
- BLOB [BINARY STREAM]
- [There are several variants for each – review in the textbook, know differences].

Filling DB with Data: INSERT Syntax



```
INSERT INTO <table>
(<columns>)
VALUES (<values>), (<values>), …
```

Where:

table = table name
columns = list of columns [optional if all columns are used]
values = list of values matching the columns

Sample INSERT Statement

```
INSERT INTO `city` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `city` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `city` VALUES (3,'Herat','AFG','Herat',186800);
```

```
INSERT INTO `countrylanguage` VALUES ('CAF','Sara','F',6.4);
INSERT INTO `countrylanguage` VALUES ('CAN','Chinese','F',2.5);
INSERT INTO `countrylanguage` VALUES ('CAN','Dutch','F',0.5);
INSERT INTO `countrylanguage` VALUES ('CAN','English','T',60.4);
INSERT INTO `countrylanguage` VALUES ('CAN','Eskimo Languages','F',0.1);
INSERT INTO `countrylanguage` VALUES ('CAN','French','T',23.4);
INSERT INTO `countrylanguage` VALUES ('CAN','German','F',1.6);
INSERT INTO `countrylanguage` VALUES ('CAN','Italian','F',1.7);
INSERT INTO `countrylanguage` VALUES ('CAN','Polish','F',0.7);
INSERT INTO `countrylanguage` VALUES ('CAN','Portuguese','F',0.7);
INSERT INTO `countrylanguage` VALUES ('CAN','Punjabi','F',0.7);
INSERT INTO `countrylanguage` VALUES ('CAN','Spanish','F',0.7);
INSERT INTO `countrylanguage` VALUES ('CAN','Ukrainian','F',0.6);
INSERT INTO `countrylanguage` VALUES ('CCK','English','T',0.0);
```

Oh, do I need to write all this?

- If data is generated on the fly (someone enters a recipe):
  - Your PHP code will generate INSERT statement, with data entered by the user, and sends it to DB
- If data is available UPFRONT in other format (e.g. csv, json, etc.)
  - Write a script, separate from your application, that prepares INSERT statements as above, saves them in the file, and then you import it into the DB via phpMyAdmin (or similar)
  - Write a PHP script that will read the data from the file and will keep sending the INSERT statements as if the user had filled those in above

DELETE/UPDATE Syntax

```
DELETE FROM <table>
        [ WHERE  <condition> ]

UPDATE <table>  SET <column>=<value>, …
        [ WHERE  <condition> ]
```

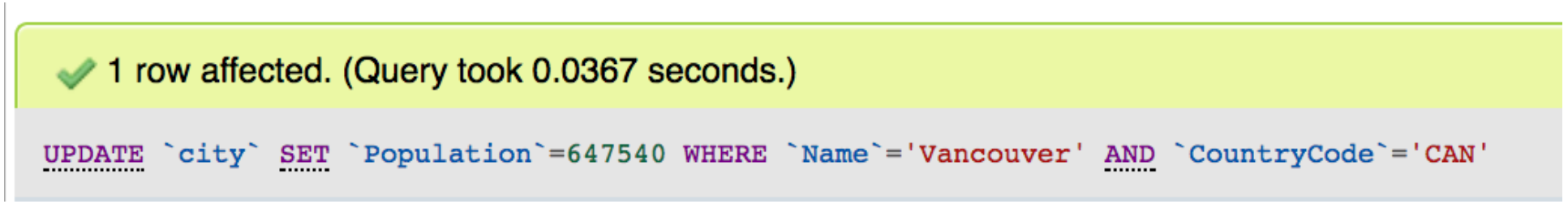
Where:

table = table name

column = column name

condition = a boolean expression that identifies tuples to be updated/deleted

Update/Delete



DELETE FROM employees WHERE Salary < 25000

*** Deletes all Employees who earn less than 25K**

DELETE FROM employees

*** Deletes all records in employees table**

SELECT Syntax

```
SELECT [DISTINCT] <columns>
FROM     <tables>
[ WHERE  <condition> ]
```

Where:

columns = list of attributes to be retrieved
tables = list of relations needed to process the query
condition = a boolean expression that identifies which tuples are to be retrieved

SELECT Full Syntax

```
SELECT [DISTINCT] items
FROM tables
[ WHERE  condition]
[GROUP BY  group_type]
[HAVING where_definition]
[ORDER BY order_type]
[LIMIT limit_criteria]
[PROCEDURE proc_name(arguments)]
[lock_options]
;
```

Selecting columns

- List all the cities above 8,000,000 (8 million)
- SELECT * FROM `city` WHERE Population > 8000000;

+ Options

	ID	Name	CountryCode	District	Population
Edit Copy Delete	206	São Paulo	BRA	São Paulo	9968485
Edit Copy Delete	939	Jakarta	IDN	Jakarta Raya	9604900
Edit Copy Delete	1024	Mumbai (Bombay)	IND	Maharashtra	10500000
Edit Copy Delete	1890	Shanghai	CHN	Shanghai	9696300
Edit Copy Delete	2331	Seoul	KOR	Seoul	9981619
Edit Copy Delete	2515	Ciudad de México	MEX	Distrito Federal	8591309
Edit Copy Delete	2822	Karachi	PAK	Sindh	9269265
Edit Copy Delete	3357	Istanbul	TUR	Istanbul	8787958
Edit Copy Delete	3580	Moscow	RUS	Moscow (City)	8389200
Edit Copy Delete	3793	New York	USA	New York	8008278

Selecting columns

- List all the cities and their population over 8 million
- SELECT Name, Population
  FROM city
  WHERE Population > 8000000;

Name	Population
São Paulo	9968485
Jakarta	9604900
Mumbai (Bombay)	10500000
Shanghai	9696300
Seoul	9981619
Ciudad de México	8591309
Karachi	9269265
Istanbul	8787958
Moscow	8389200
New York	8008278

Selecting distinct values

- List all continents in the DB that have country records
- SELECT Continent
  FROM country;
- SQL does not remove duplicates, it returns multisets!
- SELECT **DISTINCT** Continent
  FROM country;

+ Options

Continent
North America
Asia
Africa
Europe
South America
Oceania
Antarctica

+ Options

Continent
North America
Asia
Africa
North America
Europe
Europe
North America
Asia
South America
Asia
Oceania
Antarctica
Antarctica
North America
Oceania
Europe
Asia
Africa
Europe
Africa
Africa
Asia
Europe
Asia
North America

Expressions & Strings in Select

- What would the density be in each country if the population increased by 10%
  - NewDensity=(Population*1.1)/SurfaceArea
- SELECT Name, **(Population*1.1/SurfaceArea)**
  FROM country

Name	(Population*1.1/SurfaceArea)
Aruba	587.046632
Afghanistan	38.325998
Angola	11.362637
Anguilla	91.666667
Albania	130.141923
Andorra	183.333333
Netherlands Antilles	298.375000
United Arab Emirates	32.118421
Argentina	14.650842

Expressions & Strings in Select

- What would be density in each country if the population increased by 10%
  - NewDensity=(Population*1.1)/SurfaceArea
- SELECT Name, (Population*1.1/SurfaceArea) **AS NewDensity**
  FROM country

Name	NewDensity
Aruba	587.046632
Afghanistan	38.325998
Angola	11.362637
Anguilla	91.666667
Albania	130.141923
Andorra	183.333333
Netherlands Antilles	298.375000
United Arab Emirates	32.118421
Argentina	14.650842

Expressions & Strings in Select

- List all countries where density, after increase of population by 10%, would be over 1,000/km^2
- SELECT Name, (Population*1.1/SurfaceArea) AS NewDensity
  FROM country
  WHERE NewDensity > 1000

New name not available **while** processing the query

```
SELECT Name, (Population*1.1/SurfaceA
FROM `country`
WHERE NewDensity >1000 LIMIT 0, 25
```

MySQL said:

```
#1054 - Unknown column 'NewDensity' in 'where clause'
```

Expressions & Strings in Select

- List all countries where density, after increase of population by 10%, would be over 1,000/km^2
- SELECT Name, (Population*1.1/SurfaceArea) **AS NewDensity**
  FROM country
  WHERE Population*1.1/SurfaceArea > 1000

Name	NewDensity
Bermuda	1349.056604
Gibraltar	4583.333333
Hong Kong	6939.720930
Macao	28905.555556
Monaco	24933.333333
Maldives	1055.704698
Malta	1323.481013
Singapore	6349.029126
Holy See (Vatican City State)	2749.999959

Expressions & Strings in Select

- List all countries where density, after increase of population by 10%, would be over 1,000/km^2
- SELECT *
  FROM (SELECT Name, (Population*1.1/SurfaceArea) **AS NewDensity** FROM country) NC
  WHERE NC.NewDensity > 1000

NESTED statement produces a **new table**, with alias NC

Name	NewDensity
Bermuda	1349.056604
Gibraltar	4583.333333
Hong Kong	6939.720930
Macao	28905.555556
Monaco	24933.333333
Maldives	1055.704698
Malta	1323.481013
Singapore	6349.029126
Holy See (Vatican City State)	2749.999959

Pattern Matching

- Find the names of countries whose names begin with “T”
- SELECT Name
  FROM `country`
  WHERE Name LIKE 'T%'

Name
Turks and Caicos Islands
Togo
Thailand
Tajikistan
Tokelau
Turkmenistan
Tonga
Trinidad and Tobago
Tunisia
Turkey
Tuvalu
Taiwan
Tanzania

Data from Multiple Tables

- List official languages for each country
- Information is in TWO tables: country and countrylangauge
- SELECT country.Name, countrylanguage.Language
  FROM country, countrylanguage
  WHERE country.Code = countrylanguage.CountryCode AND
  countrylanguage.IsOfficial = 'T'

Name	Language
Bosnia and Herzegovina	Serbo-Croatian
Belarus	Belorussian
Belarus	Russian
Belize	English
Bermuda	English
Bolivia	Aimará
Bolivia	Ketšua
Bolivia	Spanish
Brazil	Portuguese
Barbados	English
Brunei	Malay
Bhutan	Dzongkha

Explicitly referring to attributes within tables

Specifying condition how to match records in two tables

Listing two tables separated by comma

The operation is called **JOIN**, several types of JOINs exist

Relational Algebra

- The relational algebra is a set of operations for combining and manipulating relational database tables
  - a theoretical way of talking about database queries and databases without needing an actual database system (filled with a lot of data)
  - Captures all essential rules for manipulating databases
- Relational algebra is a basis for SQL

Relational Algebra Summary

Summary of the Relational Algebra			
	Operation	Description	Example
Relational operations	selection	Selects only certain table rows	$\sigma_{age>4}(People)$ results in a table with all the rows from table People where the row's value on attribute age is greater than 4
	projection	Selects only specified table columns	$\Pi_{age,salary}(People)$ results in a table consisting of just the age and salary columns of table People
	join	Combines tables	**People** $join_{age > 4}$ **Employee** a table consisting of all the rows in the cross product of People and Employee where the age attribute is greater than 4
Set operations	union	Union of the rows in two tables	**Contestant** $\cup$ **Employee** results in a table whose rows are the union of the rows in Contestant and Employee
	intersection	Intersection of the rows in two tables	**Contestant** $\cap$ **Employee** results in a table whose rows are the intersection of the rows in Contestant and Employee
	set-difference	Set difference of the rows in two tables	**Contestant - Employee** results in a table whose rows are the set difference of the rows in Contestant and Employee
	cross-product	Cartesian product of rows in two tables	**Contestant** $\times$ **Employee** a table indicating the cross-product of all the rows in Contestant and Employee

Selection (rows)

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance $S2$ of Sailors

- SELECT *
  FROM S2
  WHERE rating > 8;

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
58	Rusty	10	35.0

Figure 4.4 $\sigma_{rating>8}(S2)$

Projection (columns)

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance $S2$ of Sailors

- SELECT sname, rating FROM S2;

sname	*rating*
yuppy	9
Lubber	8
guppy	5
Rusty	10

Figure 4.5 $\pi_{sname,rating}(S2)$

Cross Product (Cartesian Product)

- SxR: Rows are computed by combining EACH row from S with EACH row from R. In cross-product rows fields of S are followed by all fields of R

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Figure 4.1 Instance $S1$ of Sailors

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance $S2$ of Sailors

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

Figure 4.3 Instance $R1$ of Rese

(*sid*)	*sname*	*rating*	*age*	(*sid*)	*bid*	*day*
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

Figure 4.11 $S1 \times R1$

Joins

- The most useful operation in relational algebra and the most commonly used operation to **combine information from two or more tables**
- Join can be defined as **cross product** followed by selection and projection
- Important: must be computed on the fly, otherwise ineffective
- Most common: **conditional joins** (use selection condition)
- **Equijoin**: condition consist of equalities
- **Natural joins**: equijoin on all fields with the same name

Join from Cross Product

sid	bid	day
22	28	1996-10-10
58	103	1996-11-12

sid	sname	rating	age
28	Yuppy	9	35
31	Lubber	8	55.5
44	Guppy	5	35
58	Rusty	10	35

- SELECT *
  FROM S2 JOIN R1
- Basically a cross product

sid	sname	rating	age	sid	bid	day
28	Yuppy	9	35	22	28	1996-10-10
28	Yuppy	9	35	58	103	1996-11-12
31	Lubber	8	55.5	22	28	1996-10-10
31	Lubber	8	55.5	58	103	1996-11-12
44	Guppy	5	35	22	28	1996-10-10
44	Guppy	5	35	58	103	1996-11-12
58	Rusty	10	35	22	28	1996-10-10
58	Rusty	10	35	58	103	1996-11-12

- SELECT *
  FROM S2 JOIN R1 ON S2.sid=R1.sid

sid	sname	rating	age	sid	bid	day
58	Rusty	10	35	58	103	1996-11-12

Join Examples

A	B
1	4
2	3
3	2

S

C	D
1	4
1	3
3	2
4	3
5	3

T

A	B	C	D

S join$_{A+B=D}$ T (empty table)

A	B	C	D
1	4	1	4
1	4	1	3
3	2	3	2

S join $_{A = C}$ T

A	B	C	D
1	4	4	3
2	3	3	2

S join $_{B = C}$ T

A	B	C	D
1	4	1	4
1	4	3	2
1	4	5	3
1	4	4	3
2	3	1	4
2	3	3	2
2	3	5	3
2	3	4	3
3	2	1	4
3	2	3	2
3	2	5	3
3	2	4	3

S join $_{A+B <= C+D}$ T

Join from Cross Product

- SELECT *
  FROM S2 JOIN R1

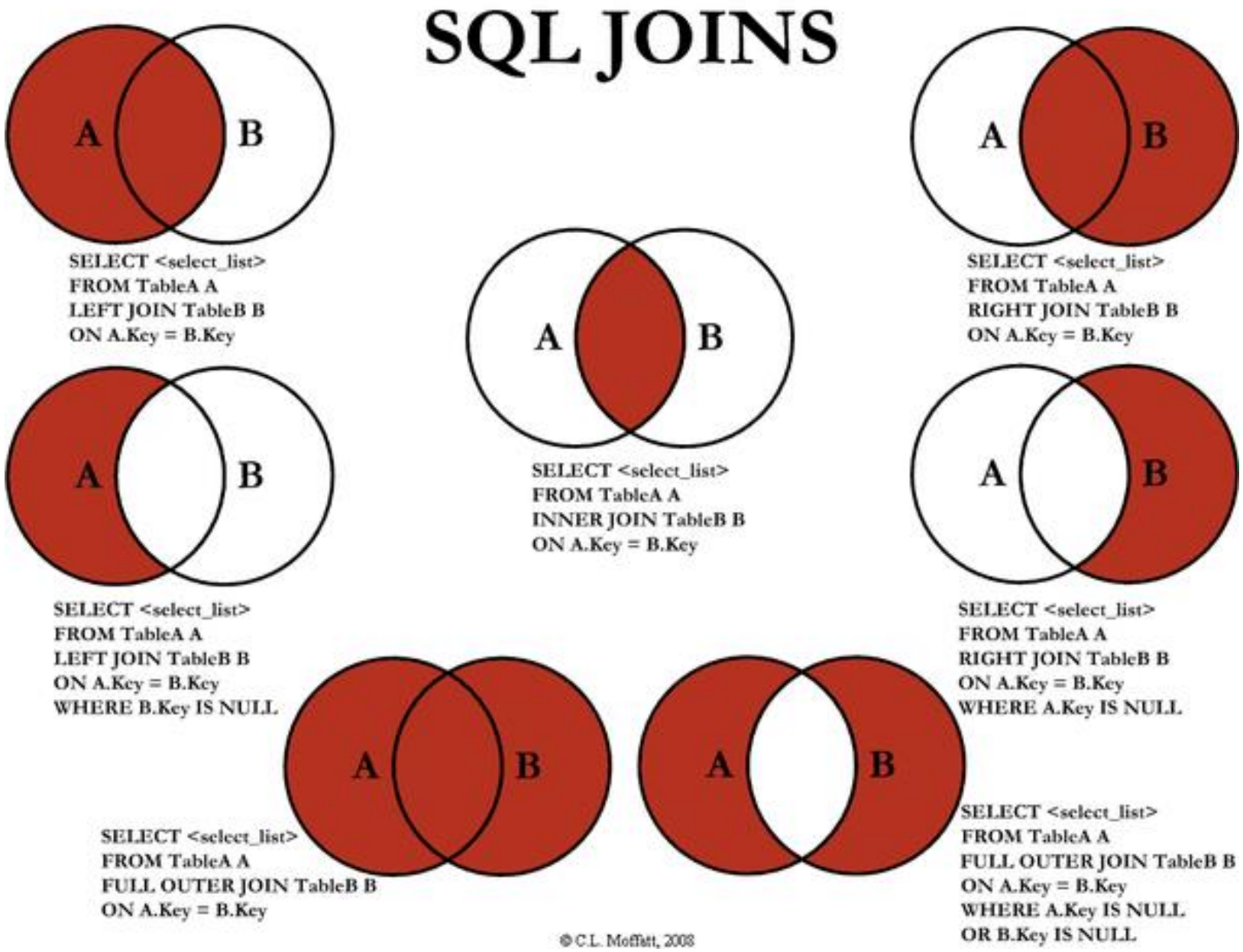
sid	sname	rating	age	sid	bid	day
28	Yuppy	9	35	22	28	1996-10-10
28	Yuppy	9	35	58	103	1996-11-12
31	Lubber	8	55.5	22	28	1996-10-10
31	Lubber	8	55.5	58	103	1996-11-12
44	Guppy	5	35	22	28	1996-10-10
44	Guppy	5	35	58	103	1996-11-12
58	Rusty	10	35	22	28	1996-10-10
58	Rusty	10	35	58	103	1996-11-12

- SELECT *
  FROM S2 JOIN R1 ON S2.sid=R1.sid

sid	sname	rating	age	sid	bid	day
58	Rusty	10	35	58	103	1996-11-12

- SELECT *
  FROM S2 JOIN R1 ON S2.sid>R1.sid

sid	sname	rating	age	sid	bid	day
28	Yuppy	9	35	58	103	1996-11-12
31	Lubber	8	55.5	58	103	1996-11-12
44	Guppy	5	35	58	103	1996-11-12



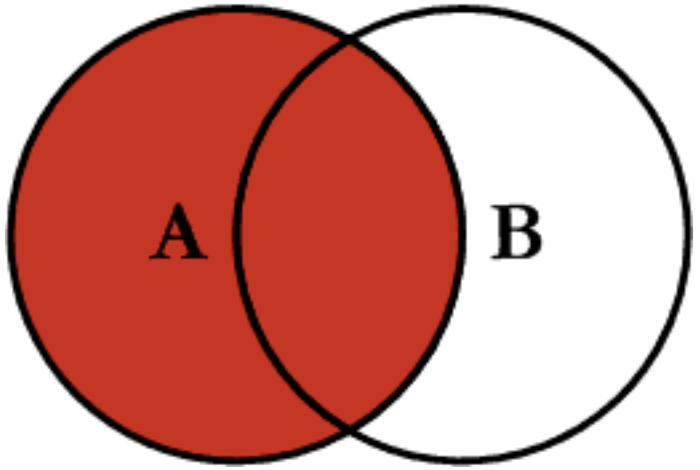
Source: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

LEFT Join

- Check if every sailor is assigned to a boat

```
SELECT *
FROM S2 LEFT JOIN R1
ON S2.sid=R1.sid
```

sid	sname	rating	age	sid	bid	day
58	Rusty	10	35	58	103	1996-11-12
28	Yuppy	9	35	*NULL*	*NULL*	*NULL*
31	Lubber	8	55.5	*NULL*	*NULL*	*NULL*
44	Guppy	5	35	*NULL*	*NULL*	*NULL*



RIGHT Join

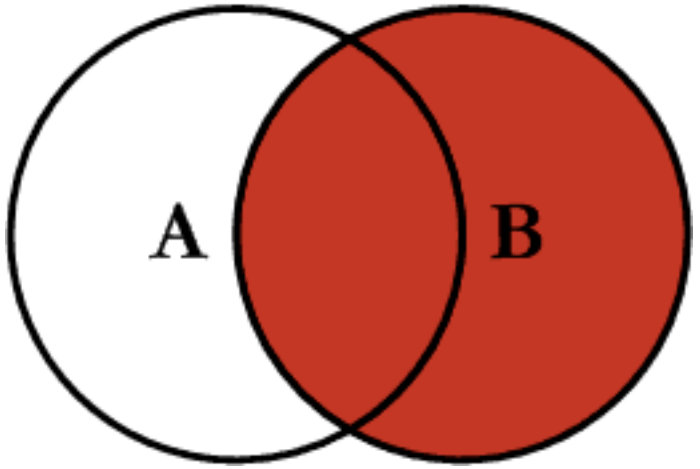
- Check if every boat has at least one sailor with a rating assigned

```
SELECT *
FROM S2 RIGHT JOIN R1
ON S2.sid=R1.sid
```

sid	sname	rating	age	sid	bid	day
NULL	*NULL*	*NULL*	*NULL*	22	28	1996-10-10
58	Rusty	10	35	58	103	1996-11-12

sid	sname	rating	age
28	Yuppy	9	35
31	Lubber	8	55.5
44	Guppy	5	35
58	Rusty	10	35

sid	bid	day
22	28	1996-10-10
58	103	1996-11-12



JOINS: Should I use WHERE or ON?

- Joining with WHERE: ANSI-89 standard syntax
- Joining with JOIN…ON; ANSI-92 syntax
- No performance difference: Optimizers treat both the same
- However, JOIN…ON syntax has other benefits:
  - Readability
  - Explicit criteria AND ordering of joining
  - Ability to filter what is being joined prior to joining
  - WHERE serves to filter the final results

Data from Multiple Tables

- List official languages for each country
- Information is in TWO tables: country and countrylangauge
- SELECT country.Name, countrylanguage.Language
  FROM country **JOIN** countrylanguage **ON**
  country.Code = countrylanguage.CountryCode
  WHERE countrylanguage.IsOfficial = 'T'

Name	Language
Bosnia and Herzegovina	Serbo-Croatian
Belarus	Belorussian
Belarus	Russian
Belize	English
Bermuda	English
Bolivia	Aimará
Bolivia	Ketšua
Bolivia	Spanish
Brazil	Portuguese
Barbados	English
Brunei	Malay
Bhutan	Dzongkha
Canada	English
Canada	French

Condition for joining moved from WHERE to ON clause

Join EXAMPLE

- List countries where English is an official language spoken by over 50% of population
- SELECT C.Name, CL.Language
  FROM country C JOIN countrylanguage CL ON C.Code = CL.CountryCode
  WHERE CL.IsOfficial = 'T' AND CL.Language="English"
  AND CL.Percentage > 50

Name	Language
Australia	English
Belize	English
Bermuda	English
Canada	English
United Kingdom	English
Gibraltar	English
Ireland	English
New Zealand	English
United States	English
Virgin Islands, U.S.	English

JOIN Example

- List countries with their capitals in Europe
- SELECT country.Name, city.Name
  FROM country **JOIN** city **ON** country.Capital=city.ID
  WHERE country.continent = “Europe”

JOIN Examples: Three Tables

- How many people in each capital speak other than official language?
- SELECT country.Name, city.Name, countrylanguage.Language,
  city.Population*countrylanguage.Percentage/100 AS LanguageUsers
  FROM country JOIN city JOIN countrylanguage ON country.Capital=city.ID AND country.Code = countrylanguage.CountryCode
  WHERE countrylanguage.IsOfficial = 'F'

Name	Name	Language	LanguageUsers
American Samoa	Fagatogo	Tongan	72.01300
Antigua and Barbuda	Saint John´s	Creole English	22967.99927
Australia	Canberra	Arabic	3227.23000
Australia	Canberra	Canton Chinese	3549.95308
Australia	Canberra	German	1936.33808
Australia	Canberra	Greek	5163.56808
Australia	Canberra	Italian	7099.90615
Australia	Canberra	Serbo-Croatian	1936.33808
Australia	Canberra	Vietnamese	2581.78404

Aggregate Operators

- COUNT ([DISTINCT]) A:
  the number of (unique) values for attribute A.
- SUM ([DISTINCT]) A:
  the sum of all (unique) values for attribute A.
- AVG ([DISTINCT]) A:
  the average of all (unique) values for attribute A.
- Max A:
  Maximum value found for attribute A
- Min A:
  Minimum value found for attribute A

Aggregate Functions

- How many countries in the World?
- SELECT COUNT(*) FROM country;
- What is the population of the most populous country?
- SELECT MAX(Population) FROM country;
- Which is the most populous country?
- SELECT MAX(Population), Name FROM country

MAX(Population)	Name
1277558000	Aruba

- SELECT Name, Population FROM country WHERE Population = MAX(Population)
  - Invalid use of group function
- SELECT Name, Population FROM country WHERE Population = (SELECT MAX(Population) FROM country)

Population	Name
1277558000	China

Nested statement returns a number we can test against

Aggregate Functions (con't)

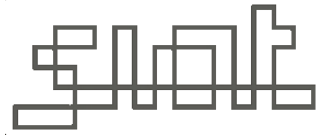
- Number of countries in Europe?
- SELECT COUNT(*)
  FROM country
  WHERE Continent = "Europe"
- 46

Aggregate function applies to the selection made by WHERE

- Number of languages around the world?
- SELECT COUNT(language) FROM countrylanguage
- 984
- SELECT COUNT(**DISTINCT** language) FROM countrylanguage
- 457

DISTINCT does not count duplicates

Aggregate Functions – GROUP BY



- For each continent, find the largest population in a country on that continent.
- SELECT MAX(Population), Continent
  FROM country
  GROUP BY Continent

MAX(Population)	Continent
1277558000	Asia
146934000	Europe
278357000	North America
111506000	Africa
18886000	Oceania
0	Antarctica
170115000	South America

Aggregate Functions – GROUP BY

- For each continent, find the countries with the largest population on that continent
- SELECT C.Population, C.Continent, C.Name
  FROM country C, (SELECT MAX(Population) AS MaxP, Continent
  FROM country
  GROUP BY Continent) MPC
  WHERE MPC.MaxP = C.Population AND MPC.Continent = C.Continent

Population	Continent	Name
0	Antarctica	Antarctica
0	Antarctica	French Southern territorie
18886000	Oceania	Australia
170115000	South America	Brazil
0	Antarctica	Bouvet Island
1277558000	Asia	China
0	Antarctica	Heard Island and McDona
111506000	Africa	Nigeria
146934000	Europe	Russian Federation
0	Antarctica	South Georgia and the South Sandwich Islands
278357000	North America	United States

JOIN: using comma notation in this case as ON statement together with nested query would be too complex

Aggregate Functions – GROUP BY

- For each continent, find the countries with the largest population on that continent, sort from the largest to smallest
- SELECT C.Population, C.Continent, C.Name
  FROM country C, (SELECT MAX(Population) AS MaxP, Continent
  FROM country
  GROUP BY Continent) MPC
  WHERE MPC.MaxP > 0 AND MPC.MaxP = C.Population AND
  MPC.Continent = C.Continent
  ORDER BY C.Population DESC

Population ▼ 1	Continent	Name
1277558000	Asia	China
278357000	North America	United States
170115000	South America	Brazil
146934000	Europe	Russian Federation
111506000	Africa	Nigeria
18886000	Oceania	Australia

Nested Queries

Find last name and salary of all managers

```
SELECT lname, salary
FROM employee
WHERE ssn IN
        (SELECT mgrssn
        FROM department)
```

"nested subquery" produces a (multi)set used by the outer query

Nested Queries

Find last name and salary of all those who are not managers.

```
SELECT lname, salary
FROM employee
WHERE ssn NOT IN
        (SELECT mgrssn
        FROM department)
```

More Nested Queries

- Make a list of project numbers for projects that involve an employee named "Smith" either as a worker or as a manager of the department that controls the project.

```
SELECT DISTINCT pnumber
FROM project
WHERE pnumber IN (SELECT pnumber
                  FROM project, department, employee
                  WHERE dnum=dnumber AND
                        mgrssn=ssn AND lname = 'Smith')
   OR  pnumber IN (SELECT pno
                   FROM works_on, employee
                   WHERE essn = ssn  AND lname = 'Smith')
```

manager

worker

More on “IN”

- Can also be used to compare a tuple of values with a set of union-compatible tuples.
- Find the employees that work on the same projects for the same number of hours as John Smith (ssn = 123456789)

```
SELECT DISTINCT essn
FROM works_on
WHERE (pno, hours) IN (SELECT pno, hours
                       FROM works_on
                       WHERE essn = '123456789')
```

Example

- Retrieve the name of each employee who has a dependent with the same first name and is of the same sex as the employee.

```
SELECT E.fname, E.lname
FROM employee E
WHERE E.ssn IN (SELECT essn
                FROM dependent D
                WHERE E.fname = D.dependent_name
                    AND E.sex = D.sex)
```

Correlated Nested Query (using EXISTS)

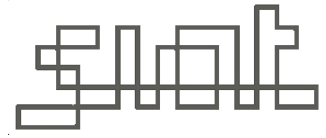
- Retrieve names of employees who have at least 1 dependent.

```
SELECT fname, lname
FROM employee E
WHERE EXISTS
        (SELECT *
        FROM dependent D
        WHERE E.ssn = D.essn)
```

Is a set returned by sub-query for this row non-empty?

evaluated for each employee tuple

Set Comparison Operators



- EXISTS, IN, UNIQUE (see text), NOT EXISTS, NOT IN, NOT UNIQUE are set operators
- SQL also supports <op> ANY and <op> ALL where <op> can be one of:

  {<, <=, =, <>, >=, >}

Example

- List employees whose salary is greater than the salary of ALL employees in department 5.

```
SELECT E.fname, E.lname
FROM employee E
WHERE E.salary > ALL (SELECT E1.salary
                      FROM Employee E1
                      WHERE E1.dno = 5)
```

Example

List employees whose salary is greater than the salary of ANY employee in department 5.

```
SELECT E.fname, E.lname
FROM employee E
WHERE E.salary > ANY (SELECT E1.salary
                      FROM Employee E1
                      WHERE E1.dno = 5)
```