

# Database Technology

## Topic 3: SQL

Olaf Hartig

[olaf.hartig@liu.se](mailto:olaf.hartig@liu.se)

# Structured Query Language

- **Declarative** language (what data to get, not how)
- Considered one of the major reasons for the commercial success of relational databases
- Statements for data definitions, queries, and updates
  - Both DDL and DML

- Terminology:

Relational Model	SQL
relation	<b>table</b>
tuple	<b>row</b>
attribute	<b>column</b>

- *Syntax notes:*
  - Some interfaces require each statement to end with a semicolon
  - SQL is not case-sensitive

# SQL DDL

# Creating Tables

```
CREATE TABLE <tablename> (  
    <colname> <datatype> [<constraint>],  
    ...,  
    [<constraint>],  
    ...  
);
```

- Data types: integer, decimal, number, varchar, char, etc.
- Constraints: not null, primary key, foreign key, unique, etc.

# Creating Tables (Example)

```
CREATE TABLE WORKS_ON (  
    ESSN    integer,  
    PNO     integer,  
    HOURS   decimal(3,1),  
  
    constraint pk_workson  
        primary key (ESSN, PNO),  
  
    constraint fk_works_emp  
        FOREIGN KEY (ESSN) references EMPLOYEE(SSN),  
  
    constraint fk_works_proj  
        FOREIGN KEY (PNO) references PROJECT(PNUMBER)  
);
```

# Modifying Table Definitions

- Add, delete, and modify columns and constraints

```
ALTER TABLE EMPLOYEE ADD COLUMN JOB VARCHAR(12);  
ALTER TABLE EMPLOYEE DROP COLUMN ADDRESS CASCADE;
```

```
ALTER TABLE WORKS_ON DROP FOREIGN KEY fk_works_emp;
```

```
ALTER TABLE WORKS_ON ADD CONSTRAINT fk_works_emp  
      FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN);
```

- Delete a table and its definition

```
DROP TABLE EMPLOYEE;
```

# SQL Queries

# Basic SQL Retrieval Queries

- All retrievals use SELECT statement:

```
SELECT  <return list>
FROM    <table list>
[ WHERE <condition> ] ;
```

where

<return list> is a list of column names (or expressions) whose values are to be retrieved

<table list> is a list of table names required to process the query

<condition> is a Boolean expression that identifies the tuples to be retrieved by the query (if no WHERE clause, all tuples to be retrieved)



# Example

```
SELECT title, year, genre
FROM Film
WHERE director = 'Steven Spielberg'
```

1. Start with the relation named in the FROM clause
2. Consider each tuple one after the other, eliminating those that do not satisfy the WHERE clause
3. For each remaining tuple, create a return tuple with columns for each expression (column name) in the SELECT clause

Film							
title	genre	year	director	minutes	budget	gross	
The Company Men	drama	2010	John Wells	104	15,000,000	4,439,063	
Lincoln	biography	2012	Steven Spielberg	150	65,000,000	181,408,467	
War Horse	drama	2011	Steven Spielberg	146	66,000,000	79,883,359	
Argo	drama	2012	Ben Affleck	120	44,500,000	135,178,251	

# All Attributes

- List all information about the employees of department 5.

```
SELECT FNAME, MINIT, LNAME, SSN, BDATE,  
        ADDRESS, SEX, SALARY, SUPERSSN, DNO  
FROM EMPLOYEE  
WHERE DNO = 5;
```

or

```
SELECT *  
FROM EMPLOYEE  
WHERE DNO = 5;
```



Possible comparison  
operators: =, <>, >, =>, etc.


# Logical Operators

- List the last name, birth date and address for all employees whose name is 'Alicia J. Zelaya'

(assuming that table *EMPLOYEE* has the attributes *FNAME*, *MINIT*, *LNAME*, *BDATE*, *ADDRESS*, etc.)

```
SELECT  LNAME, BDATE, ADDRESS
FROM    EMPLOYEE
WHERE    FNAME = 'Alicia'
           AND MINIT = 'J'
           AND LNAME = 'Zelaya';
```

Possible logical  
operators:  
and, or, not



# Pattern Matching in Strings

- List the birth date and address for all employees whose last name contains the substring 'aya'

```
SELECT BDATE, ADDRESS  
FROM EMPLOYEE  
WHERE LNAME LIKE '%aya%';
```

***LIKE** comparison operator*

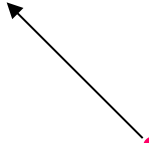
*% represents 0 or more characters*

*\_ represents a single character*

# NULLs

- List all employees that do not have a boss.

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE SUPERSSN IS NULL;
```



‘SUPERSSN = NULL’ and  
‘SUPERSSN <> NULL’  
will not return any matching tuples,  
because NULL is **incomparable** to  
any value, including another NULL

# Tables as Sets

- List all salaries:

```
SELECT SALARY  
FROM EMPLOYEE;
```

- SQL considers a table as a multi-set (bag), i.e. tuples may occur more than once in a table
  - This is different from the relational data model
- Why?
  - Removing duplicates is expensive
  - User may want information about duplicates
  - Aggregation operators (e.g., sum)

<i><b>SALARY</b></i>
30000
40000
25000
43000
38000
25000
25000
55000

# Removing Duplicates

- List all salaries:

```
SELECT SALARY  
FROM EMPLOYEE;
```

<i><b>SALARY</b></i>
30000
40000
25000
43000
38000
25000
25000
55000

- List all salaries without duplicates

```
SELECT DISTINCT SALARY  
FROM EMPLOYEE;
```

<i><b>SALARY</b></i>
30000
40000
25000
43000
38000
55000

# Set Operations

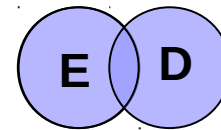
Duplicate tuples are removed.

Queries can be combined by set operations: UNION, INTERSECT, EXCEPT (*MySQL only supports UNION*)

- Example: retrieve the first names of all people in the database.

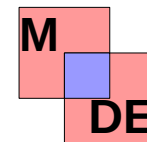
```
SELECT FNAME FROM EMPLOYEE  
UNION
```

```
SELECT DEPENDENT_NAME FROM DEPENDENT;
```



- Example: Which department managers have dependents?  
Show their SSN.

```
SELECT MGRSSN FROM DEPARTMENT  
INTERSECT  
SELECT ESSN FROM DEPENDENT;
```





# Join: Cartesian Product

- List all employees and the names of their departments.

```
SELECT LNAME, DNAME  
FROM EMPLOYEE, DEPARTMENT;
```

EMPLOYEE		DEPARTMENT	
LNAME	DNO	DNAME	DNUM
Smith	5	Research	5
Wong	5	Administration	4
Zelaya	4	Headquarters	1
Wallace	4		
Narayan	5		
English	5		
Jabbar	4		
Borg	1		

LNAME	DNAME
Smith	Research
Wong	Research
Zelaya	Research
Wallace	Research
Narayan	Research
English	Research
Jabbar	Research
Borg	Research
Smith	Administration
Wong	Administration
Zelaya	Administration
Wallace	Administration
Narayan	Administration
English	Administration
Jabbar	Administration
Borg	Administration
Smith	Headquarters
Wong	Headquarters
Zelaya	Headquarters
Wallace	Headquarters
Narayan	Headquarters
English	Headquarters
Jabbar	Headquarters
Borg	Headquarters

# Join: Equijoin

- List all employees and the names of their departments.

```
SELECT LNAME, DNAME
FROM EMPLOYEE, DEPARTMENT
WHERE DNO = DNUM;
```

Equijoin

EMPLOYEE		DEPARTMENT	
LNAME	DNO	DNAME	DNUM
Smith	5	Research	5
Wong	5	Administration	4
Zelaya	4	Headquarters	1
Wallace	4		
Narayan	5		
English	5		
Jabbar	4		
Borg	1		

Cartesian product

Foreign key in EMPLOYEE		Primary key in DEPARTMENT	
LNAME	DNO	DNAME	DNUM
Smith	5	Research	5
Wong	5	Research	5
Zelaya	4	Research	5
Wallace	4	Research	5
Narayan	5	Research	5
English	5	Research	5
Jabbar	4	Research	5
Borg	1	Research	5
Smith	5	Administration	4
Wong	5	Administration	4
Zelaya	4	Administration	4
Wallace	4	Administration	4
Narayan	5	Administration	4
English	5	Administration	4
Jabbar	4	Administration	4
Borg	1	Administration	4
Smith	5	Headquarters	1
Wong	5	Headquarters	1
Zelaya	4	Headquarters	1
Wallace	4	Headquarters	1
Narayan	5	Headquarters	1
English	5	Headquarters	1
Jabbar	4	Headquarters	1
Borg	1	Headquarters	1

# Inner Join

- List all employees and the names of their departments.

```
SELECT LNAME, DNAME  
FROM EMPLOYEE, DEPARTMENT  
WHERE DNO = DNUM;
```

As an alternative, the join condition may be given in the **FROM** clause by using the keywords **INNER JOIN** and **ON** as follows:

```
SELECT LNAME, DNAME  
FROM EMPLOYEE INNER JOIN DEPARTMENT ON DNO = DNUM;
```

# Ambiguous Names: Aliasing

- What if the same attribute name is used in different relations ?

☐ No alias

~~**SELECT** NAME, NAME  
**FROM** EMPLOYEE, DEPARTMENT  
**WHERE** DNO=DNUM;~~

☐ Whole name

**SELECT** EMPLOYEE.NAME, DEPARTMENT.NAME  
**FROM** EMPLOYEE, DEPARTMENT  
**WHERE** EMPLOYEE.DNO=DEPARTMENT.DNUM;

☐ Alias

**SELECT** E.NAME, D.NAME  
**FROM** EMPLOYEE E, DEPARTMENT D  
**WHERE** E.DNO=D.DNUM;

# Self-Join

- List the last name for all employees together with the last names of their bosses

```
SELECT E.LNAME AS "Employee",  
        S.LNAME AS "Boss"  
FROM EMPLOYEE E, EMPLOYEE S  
WHERE E.SUPERSSN = S.SSN;
```

<i>Employee Boss</i>	
Smith	Wong
Wong	Borg
Zelaya	Wallace
Wallace	Borg
Narayan	Wong
English	Wong
Jabbar	Wallace

# Self-Joins may also be written as Inner Join

- List the last name for all employees together with the last names of their bosses

```
SELECT E.LNAME AS "Employee",  
       S.LNAME AS "Boss"  
FROM EMPLOYEE E, EMPLOYEE S  
WHERE E.SUPERSSN = S.SSN;
```

```
SELECT E.LNAME "Employee",  
       S.LNAME "Boss"  
FROM EMPLOYEE E INNER JOIN EMPLOYEE S  
ON E.SUPERSSN = S.SSN;
```

<i>Employee Boss</i>	
Smith	Wong
Wong	Borg
Zelaya	Wallace
Wallace	Borg
Narayan	Wong
English	Wong
Jabbar	Wallace

# Left Outer Join

- Every tuple in left table appears in result
- If there exist matching tuples in right table, works like inner join
- If no matching tuple in right table, one tuple in result with left tuple values padded with NULL values for columns of right table

Customer			
<u>custid</u>	name	address	phone
1205	Lee	633 S. First	555-1219
3122	Willis	41 King	555-9876
2134	Smith	213 Main	555-1234
1697	Ng	5 Queen N.	555-0025
3982	Harrison	808 Main	555-4829

Sale		
<u>saleid</u>	date	custid
A17	5 Dec	3122
B823	5 Dec	1697
B219	9 Dec	3122
C41	15 Dec	1205
X00	23 Dec	NULL

```
SELECT *  
FROM Customer LEFT JOIN Sale ON Customer.custid = Sale.custid
```

Customer.custid	name	address	phone	saleid	date	Sale.custid
1205	Lee	633 S. First	555-1219	C41	15 Dec	1205
3122	Willis	41 King	555-9876	A17	5 Dec	3122
3122	Willis	41 King	555-9876	B219	9 Dec	3122
2134	Smith	213 Main	555-1234	NULL	NULL	NULL
1697	Ng	5 Queen N.	555-0025	B823	5 Dec	1697
3982	Harrison	808 Main	555-4829	NULL	NULL	NULL

# Joins Revisited

## Cartesian product

SELECT \* FROM a, b;

A2	A1	B1	B2
A	100	100	W
B	null	100	W
C	300	100	W
D	null	100	W
A	100	200	X
B	null	200	X
C	300	200	X
D	null	200	X
A	100	null	Y
B	null	null	Y
C	300	null	Y
D	null	null	Y
A	100	null	Z
B	null	null	Z
C	300	null	Z
D	null	null	Z

A		B	
A1	A2	B1	B2
100	A	100	W
null	B	200	X
300	C	null	Y
null	D	null	Z

## Equijoin, inner join

SELECT \* from A, B WHERE A1=B1;

A2	A1	B1	B2
A	100	100	W

## Thetajoin

SELECT \* from A, B WHERE A1>B1;

A2	A1	B1	B2
C	300	100	W
C	300	200	X



# Joins Revisited (cont'd)

## Right outer join

SELECT \* FROM A RIGHT JOIN B on A1=B1;

A2	A1	B1	B2
A	100	100	W
null	null	200	X
null	null	null	Y
null	null	null	Z

## Left outer join

SELECT \* FROM A LEFT JOIN B on A1=B1;

A2	A1	B1	B2
A	100	100	W
C	300	null	null
B	null	null	null
D	null	null	null

A		B	
A1	A2	B1	B2
100	A	100	W
null	B	200	X
300	C	null	Y
null	D	null	Z

## Full outer join (“union” of right+left)

SELECT \* FROM A FULL JOIN b on A1=B1;

A2	A1	B1	B2
A	100	100	W
null	null	200	X
null	null	null	Y
null	null	null	Z
C	300	null	null
B	null	null	null
D	null	null	null

# Subqueries

- List all employees that do not have any project assignment with more than 10 hours

❑ ~~SELECT LNAME FROM EMPLOYEE, WORKS\_ON  
WHERE SSN = ESSN AND HOURS <= 10.0;~~

{>, >=, <, <=, <>}  
+  
{ANY, SOME, ALL}

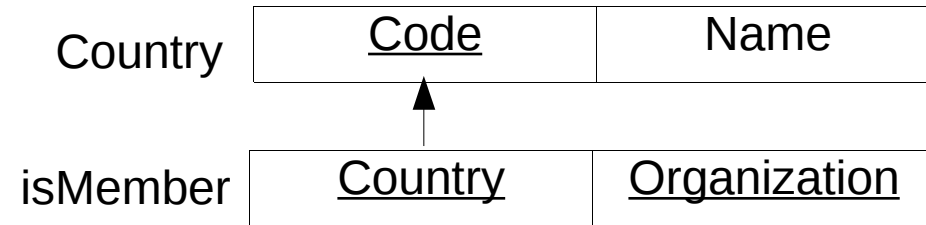
**SELECT** LNAME  
**FROM** EMPLOYEE  
**WHERE** SSN **NOT IN** (**SELECT** ESSN **FROM** WORKS\_ON  
                          **WHERE** HOURS > 10.0);

Or

**SELECT** LNAME  
**FROM** EMPLOYEE  
**WHERE** **NOT EXISTS** (**SELECT** \* **FROM** WORKS\_ON  
                          **WHERE** SSN = ESSN **AND** HOURS > 10.0);

**EXISTS**

# Quiz



- Consider the following two SQL queries:

```
SELECT Name
FROM Country
WHERE Code IN ( SELECT Country
                 FROM IsMember
                 WHERE Organization = 'EU' );
```

```
SELECT Name
FROM Country, IsMember
WHERE Code = Country
      AND Organization = 'EU';
```

- Are these queries equivalent?

(yes)

(no)

# Additional Features

# Extended SELECT Syntax

**SELECT** *<attribute-list and function-list>*  
**FROM** *<table-list>*  
[ **WHERE** *<condition>* ]  
[ **GROUP BY** *<grouping attribute-list>* ]  
[ **HAVING** *<group condition>* ]  
[ **ORDER BY** *<attribute-list>* ];

# Aggregate Functions

- Used to accumulate information from multiple tuples, forming a single-tuple summary
- Built-in aggregate functions: SUM, MAX, MIN, AVG, COUNT
- Example: *What is the average budget of all movies ?*  
`SELECT AVG(budget) FROM Film;`
- Used in the SELECT clause and the HAVING clause
  - Hence, cannot be used in the WHERE clause!
- NULL values are not considered in the computations; e.g.,:

	50	50
	100	100
	NULL	0
AVG:	75	50

# Aggregate Functions (cont'd)

- Example

*How many movies were directed by Steven Spielberg?*

```
SELECT COUNT(*) FROM Film  
WHERE director='Steven Spielberg';
```

- All tuples in the result are counted, *with duplicates!*
  - i.e., COUNT(title) or COUNT(director) give same result
- COUNT(DISTINCT year) would include each year only once

# Grouping Before Aggregation

- How can we answer a query such as  
*“How many films were directed by each director after 2001?”*
- Need to produce a result with one tuple per director
  1. Partition relation into subsets based on **grouping column(s)**
  2. Apply aggregate function to each such group independently
  3. Produce one tuple per group



# Grouping Before Aggregation

- How can we answer a query such as  
*“How many films were directed by each director after 2001?”*
- **GROUP BY** clause to specify grouping attributes

```
SELECT director, COUNT(*)  
FROM Film  
WHERE year > 2001  
GROUP BY director;
```
- **Important:** Every element in SELECT clause must be a grouping column or an aggregation function
  - e.g., `SELECT director, year, COUNT(*)` would not be allowed (in the query above) unless *also* grouping by year:  
i.e., `GROUP BY director, year`

# Filtering Out Whole Groups

- After partitioning into groups, whole partitions can be discarded
  - i.e., HAVING clause specifies a condition on the grouped tuples

```
SELECT DNO, COUNT(*), AVG(SALARY)  
FROM EMPLOYEE  
GROUP BY DNO  
HAVING COUNT(*) > 2;
```

- HAVING clause cannot reference individual tuples within a group
  - Instead, can reference grouping column(s) and aggregates only
- Contrast WHERE clause to HAVING clause

*Note:* As for aggregation, no GROUP BY clause means relation treated as one group

# Sorting Query Results

- Show the department names and their locations in alphabetical order

```
SELECT DNAME, DLOCATION  
FROM DEPARTMENT D, DEPT_LOCATIONS DL  
WHERE D.DNUMBER = DL.DNUMBER  
ORDER BY DNAME ASC, DLOCATION DESC;
```

<i>DNAME</i>	<i>DLOCATION</i>
Administration	Stafford
Headquarters	Houston
Research	Sugarland
Research	Houston
Research	Bellaire

# SQL Data Manipulation

# Inserting Data

**INSERT INTO** *<table>* (*<attr>*,...) **VALUES** ( *<val>*, ...) ;

**INSERT INTO** *<table>* (*<attr>*, ...) *<subquery>* ;

- Store information about how many hours an employee works for the project '1' into WORKS\_ON.

**INSERT INTO** WORKS\_ON **VALUES** (123456789, 1, 32.5);

Integrity constraint!

Referential integrity constraint!

# Updating Data

**UPDATE** *<table>* **SET** *<attr>* = *<val>* ,...

**WHERE** *<condition>* ;

**UPDATE** *<table>* **SET** (*<attr>*, ....) = ( *<subquery>* )

**WHERE** *<condition>* ;

Integrity constraint!

Referential integrity constraint!

- Give all employees in the 'Research' department a 10% raise in salary

**UPDATE** *EMPLOYEE*

**SET** *SALARY* = *SALARY*\*1.1

**WHERE** *DNO* IN (**SELECT** *DNUMBER*  
**FROM** *DEPARTMENT*

**WHERE** *DNAME* = 'Research');

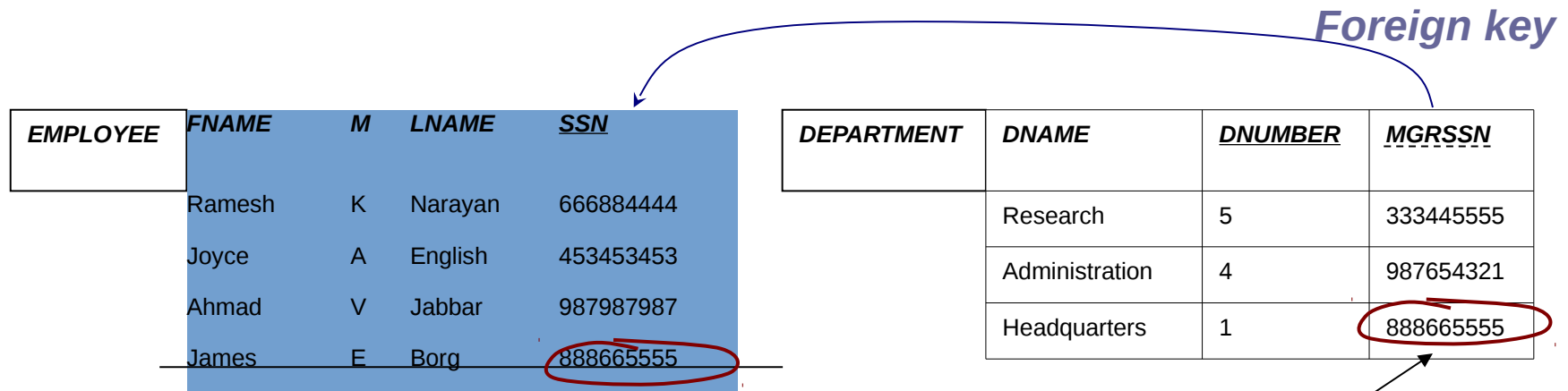
# Deleting Data

**DELETE FROM** *<table>* **WHERE** *<condition>* ;

- Delete the employees having the last name 'Borg' from the EMPLOYEE table.

**DELETE FROM** *EMPLOYEE*

**WHERE** *LNAME* = 'Borg';



ON DELETE SET NULL / DEFAULT / CASCADE ?

**Referential integrity constraint!**

# Views



# What are Views?

- A **virtual** table **derived** from other (possibly virtual) tables, i.e. always up-to-date

```
CREATE VIEW dept_view AS  
SELECT DNO, COUNT(*) AS C, AVG(SALARY) AS S  
FROM EMPLOYEE  
GROUP BY DNO;
```

- Why?
  - ☐ Simplify query commands
  - ☐ Provide data security
  - ☐ Enhance programming productivity

[www.liu.se](http://www.liu.se)