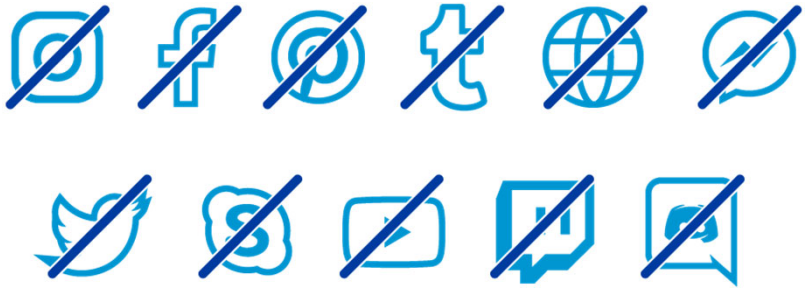
DATABASES

Source: https://en.itpedia.nl/2017/11/26/wat-is-een-database/

Prof. Dr. Ulrike Herster
Hamburg University of Applied Sciences

HAW HAMBURG

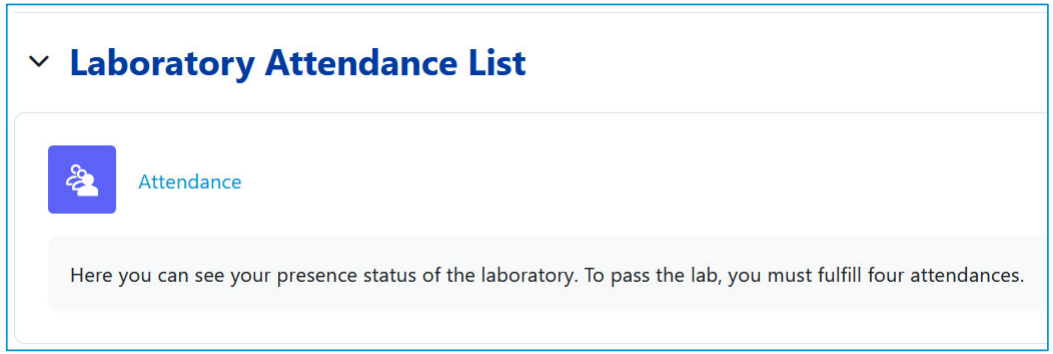
COPYRIGHT



The publication and sharing of
slides, images and sound recordings of this
course is not permitted

LABORATORY ATTENDANCE LIST

- In our moodle room you find a *Laboratory Attendance List*



- This list documents
  - Your attendance for the four labs
  - Comments, e.g. about a presentation you did within a laboratory
- After each laboratory you have time until the following Friday to report incorrect comments / absences by e-mail to the lecturer of that lab (Moldenhauer, Yildirim, or Herster), e.g., for the first laboratory on 29.04.2024 you have time until Friday, 03.05.2024

ORGANIZATION
OUR JOURNEY IN THIS SEMESTER

- Integrity, Trigger & Security
- Database Applications
- Transactions
- Subqueries & Views
- **More SQL**
- Notations & Guidelines
- Constraints
- Relationships
- Simple Entities and Attributes
- Basics

Source: Foto von Justin Kauffman auf Unsplash

MORE SQL
DATA DEFINITION: CREATE TABLE


Repetition

Syntax for creating an empty table:

```
CREATE TABLE <relationname>
       (<column> <type> [ DEFAULT expr]
              [ [NOT] NULL ] [ colconstraint ] *
       [,{<column> <type> [ DEFAULT expr ]
              [ [NOT] NULL ] [ colconstraint ] *
        | <tableconstraint> } ] *
       ) ;
```

Source: Elmasri, Fundamentals of Database Systems, Page 88ff

MORE SQL
DATA DEFINITION: CREATE TABLE

- **`DEFAULT`** expr → default value
- Default for expr → NULL
- expr may be literal, system variable, or function
- Example:
  - **`DEFAULT`** `0`
  - **`DEFAULT`** `current timestamp`

MORE SQL
DATA DEFINITION: CREATE TABLE

Example:

```
CREATE TABLE Employee
       ( . . . ,
       Dno             INT             NOT NULL      DEFAULT 1,
       ) ;

CREATE TABLE Department
       ( . . . ,
       Mgr_ssn         CHAR(9)         NOT NULL      DEFAULT‘001’,
       . . . ,
       ) ;
```

Source: Elmasri, Fundamentals of Database Systems, Page 88ff

MORE SQL
DATA DEFINITION: CREATE TABLE

- The **AUTO_INCREMENT** attribute can be used to generate a unique identity for new rows

- Example:

```
CREATE TABLE Department
( Dname     VARCHAR(15)   NOT NULL,
Dnumber     INT           AUTO_INCREMENT,
…
PRIMARY KEY (Dnumber) ,
UNIQUE (Dname),
…);
```

Source: https://dev.mysql.com/doc/refman/8.0/en/example-auto-increment.html

MORE SQL
DATA DEFINITION: CREATE TABLE

- The **AUTO_INCREMENT** attribute can be used to generate a unique identity for new rows
- When you insert any other value into an **AUTO_INCREMENT** column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the largest column value
- Example:

```
INSERT INTO animals (id,name) VALUES(100,'rabbit');
INSERT INTO animals (id,name) VALUES(NULL,'mouse');
SELECT * FROM animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | dog       |
|   2 | cat       |
|   3 | penguin   |
|   4 | lax       |
|   5 | whale     |
|   6 | ostrich   |
|   7 | groundhog |
|   8 | squirrel  |
| 100 | rabbit    |
| 101 | mouse     |
+-----+-----------+
```

Source: https://dev.mysql.com/doc/refman/8.0/en/example-auto-increment.html

MORE SQL
DATA DEFINITION: SEQUENCES

Not supported by mySQL!

- Used to create unique, increasing numbers
- Can be used for generating artificial keys
- Available in many DBMS and in SQL2003

- Syntax:

```
CREATE SEQUENCE <seqname>
       [ INCREMENT BY < integer >]
       [ START WITH < integer > ]
       [...];
```

MORE SQL
DATA DEFINITION: SEQUENCES

Not supported by mySQL!

- Default increment: 1
- Default start: 1
- Sequences can be used as DEFAULT value

- Example PostgreSQL:

```
CREATE SEQUENCE nr_seq START WITH 5 INCREMENT BY 5 ;
CREATE SEQUENCE matr_seq;
CREATE TABLE Student (
        matrno INTEGER DEFAULT nextval ( 'matr_seq' ),
        matr_name VARCHAR( 30 ) ) ;
```

MORE SQL
DATA DEFINITION: ALTER TABLE


Repetition

- For modifying an existing relation
  - COLUMN: **`ADD, DROP, MODIFY`**
  - CONSTRAINT: **`ADD, DROP`**
  - TABLE: **`RENAME`**
  - Vendor-specific extensions

MORE SQL
DATA DEFINITION: DROP

- Delete named schema elements,
  e.g., tables, constraints, schema, indexes, triggers

- Need to Observe Referential Integrity
  ... so, need to drop tables in correct order

- Two drop behavior options:
  - **`CASCADE`**
  - **`RESTRICT`**

- **`DROP`** deletes all data AND the data definition
  → if you want to delete only the data then use **`DELETE`**

Source: Elmasri, Fundamentals of Database Systems, Page 88ff

MORE SQL
DATA DEFINITION: DROP

Syntax:

```
DROP TABLE < relationname > [( CASCADE | RESTRICT )]
```

Oracle:

```
DROP TABLE < relationname > [ CASCADE CONSTRAINTS ]
```

MORE SQL
DATA DEFINITION: DROP

- Example schema:

```
DROP SCHEMA COMPANY CASCADE;
```

- Example table:

```
DROP TABLE Dependent RESTRICT;
```

→ The table is dropped only if it is not referenced in any constraint or view or by another element

```
DROP TABLE Dependent CASCADE;
```

→ The table is dropped even if there are references

Source: Elmasri, Fundamentals of Database Systems, Page 88ff

MORE SQL
DATA DEFINITION: DROP

- Specifiy the drop behavior
  - **CASCADE**
  - **RESTRICT**

- Example:

```
ALTER TABLE COMPANY.Employee DROP COLUMN Address CASCADE;
```

Source: Elmasri, Fundamentals of Database Systems, Page 88ff

MORE SQL
DATA DEFINITION: DROP

Example:

```
ALTER TABLE COMPANY.Department ALTER COLUMN Mgr_ssn
DROP DEFAULT;

ALTER TABLE COMPANY.Department ALTER COLUMN Mgr_ssn
SET DEFAULT ‘333445555’;

ALTER TABLE COMPANY.Employee
DROP CONSTRAINT EMPSUPERFK CASCADE;
```

Source: Elmasri, Fundamentals of Database Systems, Page 88ff

MORE SQL
DATA DEFINITION: INDEX

- Internal structure to increase speed of queries
  → speed up the search for and retrieval of records (access paths)
  - But slow down inserts and updates
  - Memory consumption!

- Earlier versions of SQL had commands for creating indexes, but these were removed because they were not at the conceptual schema level
- Many systems still have the **`CREATE INDEX`** commands.

- Syntax:

```
CREATE [ UNIQUE ] INDEX <name>
       ON < table > ( < column > [ , . . . ] )
```

Source: Elmasri, Fundamentals of Database Systems, Page 88ff

MORE SQL
DATA DEFINITION: INDEX

- Column is used often for searches or sorting
- Many different values, not many NULLs
- Many rows in table
- More reads than writes on data
- Might be used as join condition
- RDBMS must check value for referential integrity
- Column is an FK
- Referenced column (PK) usually already has an index

MORE SQL
DATA DEFINITION: OTHER OBJECTS

- Can be **CREATE**d, **ALTER**ed, **DROP**ped
- **USER, ROLE**
  - DB users and groups
- **VIEW**
  - User view on table (external layer)
- Syntax:

```
CREATE USER user [ IDENTIFIED BY [PASSWORD] ’ passwd ’ ]
       [ , user [ IDENTIFIED BY [PASSWORD] ’ passwd ’ ] ]
```

MORE SQL
DATA DEFINITION: OTHER OBJECTS

- Example: User
  - Either owner of a relation or the DBA can grant (or revoke) selected users the privileges to use a SQL statement
    (e.g., **SELECT, INSERT, DELETE**)

```
CREATE USER ‘student' IDENTIFIED BY ‘123’;
GRANT ALL PRIVILEGES ON COMPANY.Employee TO ‘student’;

REVOKE DROP ON COMPANY.Employee FROM ‘student’;
SHOW GRANTS FOR student ;
```

MORE SQL
DATA DEFINITION: OTHER OBJECTS

- TABLESPACE
  - Grouping of tables based on physical storage
- SYNONYM
  - Alias name for tables, views, sequences
- FUNCTION, PROCEDURE
  - Stored Procedure, Persistent Stored Module (PSM)
- TRIGGER
  - Active rule for certain events

MORE SQL
DATA MANIPULATION

- **`INSERT`, `UPDATE`, `DELETE`**
  - All operations work on a set of tuples
    → Special case(!): work on one tuple
  - Example for modifications of sets of tuples:
    - Increase the wage of all employees by 10
    - Delete stock with price below 1€
    - Set the academic title of some students to ’BSc’

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
DATA MANIPULATION: TRANSACTIONS IN A NUTSHELL

- Start a transaction
  - Some DBMS (e.g., PostgreSQL, but not on Oracle): need to explicitly start a transaction
  - begin or start
    → Autocommit mode?
- Commit to a group of changes
  - **`commit ;`**
  - Until **`commit ;`** changes are local to your session
  - If you forget to commit, your changes will be lost
- Undo last changes
  - **`rollback ;`**

MORE SQL
DATA MANIPULATION: INSERT

- Syntax:

```
INSERT INTO < table >
      [ ( < column > [ , … ] ) ]
      VALUES ( < expression > [ , ...] )
```

- Column list is optional
  - If omitted, values list must match table's attributes
  - If given, we don't have to specify values for all columns
    → Other columns will get the **DEFAULT** value (or NULL)

MORE SQL
DATA MANIPULATION: INSERT

- There are 2 possibilities for inserting:
  1. Constant tuples
  2. Tuples returned by a query

MORE SQL
DATA MANIPULATION: INSERT


Repetition

- There are 2 possibilities for inserting:
  1. Constant tuples
     - Made from literals
     - ... or from function calls, variables
     - The values must include all attributes with `NOT NULL` specification and no default value
     - E.g., 3+5, current timestamp

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
DATA MANIPULATION: INSERT


Repetition

- There are 2 possibilities for inserting.
  1. Constant tuples
     - Example:

```
INSERT INTO EMPLOYEE
VALUES ( 'Arthur' , 'C', 'Brown' , 323232323,
'1970-12-31', 'London', 'm', 45000, 3334455555, 5 ) ;

INSERT INTO EMPLOYEE ( fname, lname, ssn, super_ssn, dno)
VALUES ( 'Andi' , 'Red', 343434343, 333445555, 5) ;
```

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
DATA MANIPULATION: INSERT


Repetition

Example Constant tuples:

```
INSERT INTO Person
      VALUES ( 1 , 'Miller' , 'Olaf' , 'Hamburg' ) ;
INSERT INTO Person (PNr, Name, Fname, BornIn)
      VALUES ( 2 , 'Meier' , 'Stefan' , 'Berlin' ) ;
INSERT INTO Person (PNr, BornIn, Fname, Name)
      VALUES ( 4 , 'Hamburg' , 'Olaf' , 'Schulz' ) ;
INSERT INTO Person
      VALUES ( 3 , 'Miller' , 'Karina' , 'Wien' ) ;
COMMIT;
```

Person

PNR	Name	Fname	BornIn
1	Miller	Olaf	Hamburg
2	Meier	Stefan	Berlin
4	Schulz	Olaf	Hamburg
3	Miller	Karina	Wien

MORE SQL
DATA MANIPULATION: INSERT


Repetition

Example Constant tuples:

```
INSERT INTO Book
      VALUES ( 1 , 1 , 4712 , 'DBS' ) ;
INSERT INTO Book
      VALUES ( 2 , 2 , 9991 , 'DB1') ;
COMMIT ;
INSERT INTO Book
      VALUES ( 3 , NULL , 4242 , 'Hitch' ) ;
ROLLBACK ;
INSERT INTO Book ( BNr, ISBN , Title )
      VALUES ( 3 , 4242, 'Hitchhiker') ;
```

Book

BNr	PNr	ISBN	Title
1	1	4212	DBS
2	2	9991	DB1
3	NULL	4242	Hitchhiker

MORE SQL
DATA MANIPULATION: INSERT

- There are 2 possibilities for inserting.
  1. Constant tuples
  2. Tuples returned by a query

Example:

```
INSERT INTO  Underpaid ( lname , fname )
       SELECT lname , fname
       FROM Employee
       WHERE salary < 1000 ;
```

→ **WHERE** clause belongs to **SELECT**

MORE SQL
DATA MANIPULATION: INSERT

- A DBMS format can be used (depends on the language of DBMS)
  - Example in MySQL:
    ```
    UPDATE Person SET birthdate = '2008-12-31'
    ```
- A date-function can be used
  - Example in Oracle:

```
INSERT INTO Person (name , birthdate)
    VALUES ('Anna' , '02-FEB-1955') ;
INSERT INTO Person (name , birthdate)
    VALUES ('Anna' , TO_DATE('02.02.1955')) ;
INSERT INTO Person (name , birthdate)
    VALUES ('Anna' , TO_DATE( '02-02-1955' , 'DD-MM-YYYY' )) ;
```

MORE SQL
DATA MANIPULATION: RECAP - CONSTRAINTS

- All modifications need to observe constraints:
  - Domain Constraints
    - Data types must match (or be casted)
    - Type Conversion: implicit vs. explicit
    - Vendor-specific
  - Entity Integrity
    - PK value is not null and unique
  - Referential Integrity (FK)
    - Insert data into master table first
  - Semantical Integrity (check constraints)

MORE SQL
DATA MANIPULATION: UPDATE

- Syntax:

```
UPDATE < table >
SET < column > = < expression >
[ WHERE < condition >]
```

- Used to modify attribute values of one or more selected tuples
- Can modify only tuples of one table at a time
- **WHERE** clause: optional!
  → If left out: Update all tuples
- Note: updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action
  is specified in the referential integrity constraints of the DDL

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
DATA MANIPULATION: UPDATE

Examples:

```
UPDATE Person
SET    lname= 'Brown' , married = TRUE
WHERE  id = 45 ;

UPDATE Employee
SET    salary = salary * 1.1 ;

UPDATE Person
SET    email = NULL
WHERE  email IS NOT NULL ;
```

MORE SQL
DATA MANIPULATION: DELETE

- Removes tuples from a relation, the relation stays in the database
- Syntax:

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

- **WHERE** clause: optional
  → If left out: Delete all tuples!
- Observe referential integrity!!!

MORE SQL
DATA MANIPULATION: ASSIGNMENT

- Insert a new student, <‘Johnson’, 25, 1,‘Math’>, in the database.
- Change the class of student ‘Smith’ to 2.
- Insert a new course:
  <‘Knowledge Engineering’, ‘CS4390’, 3, ‘CS’>.
- Delete the record for the student whose name is ‘Smith’ and whose student number is 17.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

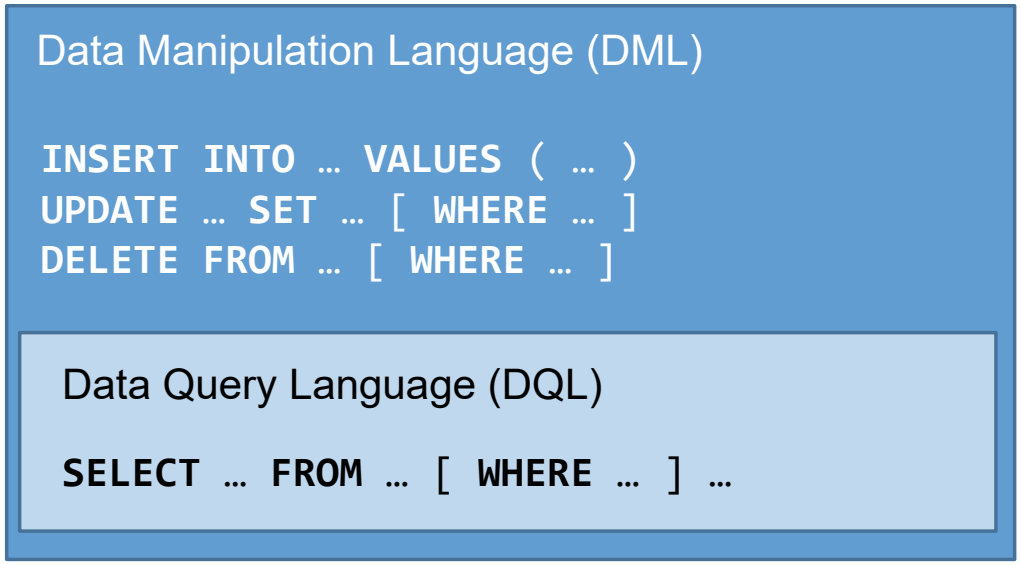
MORE SQL
QUERYS: WHAT DO YOU EXPECT FROM A GOOD QUERY LANGUAGE

- Ad-hoc-formula: No programs, requests!
- Descriptiveness: ”What do I want”, not ”How do I get it”
- Set-orientation: Much data at once, not only a single tuple
- Seclusion: All results are relations again and can be queried again
- Adequate: All data model constructs are supported
- Orthogonal: There are view independent commands that can be combined

MORE SQL
QUERYS: WHAT DO YOU EXPECT FROM A GOOD QUERY LANGUAGE

- Optimizable: The language is transformable, so that the user may use simple queries that are substituted into fast ones (with the same result!)
- Efficiency: Each operation can be executed efficiently
- Security: All queries lead to finite result sets in finite time
- Completeness: Everything that is requestable, can be formulated by a query

MORE SQL QUERYS



- Data Query Language is used to extract data from the database
- It doesn't modify any data in the database
- There is only one basic statement: **SELECT**

Source: Elmasri, Fundamentals of Database Systems, Page 145ff

MORE SQL QUERYS

One big difference between Relational Model and SQL:

- SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values.
  → SQL table is not a set of tuples, it is a multiset

- Some SQL relations are constrained to be sets because
  - a key constraint has been declared or
  - the DISTINCT option has been used with the SELECT statement

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
QUERYS

Syntax:

```
SELECT [ DISTINCT | ALL ] < attribute_list >
FROM < table list >
[ WHERE < condition > ]
[ <group by clause > ]
[ <having clause > ]
[ UNION [ ALL ] < query specification> ]
[ < order by clause > ]
```

MORE SQL QUERYS

SELECT – Basic form

```
SELECT  <attribute list>
FROM    <table list>
WHERE   <condition>
```

- `<attribute list>` is a list of attribute names (columns) whose values are to be retrieved by the query
- `<table list>` is a list of the relation names (e.g., tables) required to process the query
- `<condition>`: optional conditional (Boolean) expression that identifies the tuples to be retrieved by the query

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL QUERYS

Example:

What do these statements mean???

```
SELECT Bdate, Address
FROM Employee
WHERE Fname = 'John' AND Minit = 'B' AND Lname = 'Smith';

SELECT Fname, Lname, Address
FROM Employee, Department
WHERE Dname = 'Research' AND Dnumber = Dno;
```

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
QUERYS: ATTRIBUTE LIST

- <attribute list> is a list of attribute names (columns) whose values are to be retrieved by the query

  Example:

```
SELECT fname, lname, ssn
FROM   Employee ;
```

- Asterisk (*) stands for: all attributes
  Example:

```
SELECT *
FROM   Employee ;
```

- Arithmetic expressions and aggregation functions are possible

MORE SQL
QUERYS: ATTRIBUTE LIST

- Syntax:

```
<attribute list>    ::= [<table>.]* <column list>
<column list>       ::= <select sublist>
                        [, <select sublist> ] *
<select sublist>    ::= <element> [[AS] <column name> ]
<element>           ::= [ <table>.] <column_name>
```

- Attributes of the projection can be given directly, if they are unambiguous
- It is always possible to qualify by relation

MORE SQL
QUERYS: ATTRIBUTE LIST

Example:

What do these statements mean???

```
SELECT fname AS surname              -- AS is optional
FROM   Employee ;

SELECT dname
FROM   Department ;

SELECT Employee.ssn, Department.dname
FROM   Employee, Department ;
```

Even if Department has an attribute ”ssn”, the reference is clear

MORE SQL
QUERYS: ATTRIBUTE LIST

- SQL uses (mainly) multiset semantics
  - No elimination of duplicates
  - No duplicates wanted: use **DISTINCT**

- Example:

```
SELECT     DISTINCT super_ssn
FROM       Employee ;

SELECT     DISTINCT salary
FROM       Employee ;
```

- **SELECT DISTINCT** is the Projection of Relational Algebra

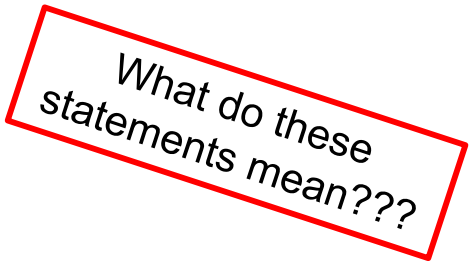
MORE SQL
QUERYS: ATTRIBUTE LIST

- The same name can be used for two (or more) attributes as long as the attributes are in different relations
- If this is the case, and a multi-table query refers to two or more attributes with the same name, we must qualify the attribute name with the relation's name
- This is done by prefixing the relation's name to the attribute name and separating the two by a period
- The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
QUERYS: ATTRIBUTE LIST

Example:




```
SELECT Fname, Employee.lname, Address
FROM   Employee, Department
WHERE  Department.dname = "Research" AND
       Department.Dnumber = Employee.Dno;

SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM   Employee AS E, Employee AS S
WHERE  E.Super_ssn = S.Ssn;
```

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
QUERYS: TABLE LIST

- `<table list>` is a list of the relation names (tables) required to process the query

- Example:

```
SELECT ... FROM Employee ...
SELECT ... FROM Employee , Department ...
```

- More than one table: Cartesian Product
  - Possibly huge result set
  - ... when used without < `condition` >

MORE SQL
QUERYS

Syntax:

```
SELECT [ DISTINCT | ALL ] < attribute_list >
FROM < table list >
[ WHERE < condition > ]
[ <group by clause > ]
[ <having clause > ]
[ UNION [ ALL ] < query specification> ]
[ < order by clause > ]
```

MORE SQL
QUERYS: CONDITION

- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

- Examples:

```
SELECT * FROM Employee WHERE ssn = 333445555 ;
SELECT * FROM Employee WHERE lname IS NULL ;
```

- The **WHERE** clause is optional!
  → If left out: retrieve all tuples
  → If more than one relation is specified in the **FROM** clause and there is no **WHERE** clause, then the Cross Product is selected

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
QUERYS: CONDITION – EXCURSION LOGIC

- Compare two expressions
  - Comparison operators: =, <, <= , >, >=, <> (≠, !=)
  - Expressions could be columns, literals
  - Example:

```
... WHERE age >= 18 ;                -- older than 18
... WHERE last_name <> ’ Miller ’    -- not equals Miller
```

- Check for NULL: **IS NULL**

- **AND**, **OR**, **NOT**
  - Example:

```
… WHERE (age >= 18) AND (last_name <> ’Miller’)
```

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
QUERYS: CONDITION – EXCURSION LOGIC


Repetition

- 42 < NULL ?
  → Comparisons against NULL are never true...

- 42 >= NULL ?
  → ... but they are not false, too!

- So, we need a Ternary Logic
  → Values: TRUE, FALSE, NULL

MORE SQL
QUERYS: CONDITION – EXCURSION LOGIC


Repetition

- NOT:
  → NOT (NULL) = ? **= NULL**

- AND:
  → TRUE AND NULL = ? **= NULL**
  → FALSE AND NULL = ? **= FALSE**

- OR:
  → TRUE OR NULL = ? **= TRUE**
  → FALSE OR NULL = ? **= NULL**

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
QUERYS: CONDITION – EXCURSION LOGIC

a	b	a AND b	a OR b	NOT a
0	0	0	0	1
0	1	0	1	1
0	NULL	0	NULL	1
1	0	0	1	0
1	1	1	1	0
1	NULL	NULL	1	0
NULL	0	0	NULL	NULL
NULL	1	NULL	1	NULL
NULL	NULL	NULL	NULL	NULL

MORE SQL
QUERYS: SET OPERATIONS

- SQL has incorporated some of the set operations:
  - Union (**`UNION`**)
  - Set Difference (**`EXCEPT`**)
  - Intersection (**`INTERSECT`**)

- Note: Set operations apply only to *union-compatible relations:*

  Union compatible means
  - that the two relations have the same number of attributes and
  - each corresponding pair of attributes has the same domain

MORE SQL
QUERYS: SET OPERATIONS

- Example:

What do these statements mean???

```
(SELECT     DISTINCT Pnumber
FROM        PROJECT, DEPARTMENT, EMPLOYEE
WHERE       Dnum=Dno AND Mgr_ssn=Ssn AND Lname="Wong" )
UNION
(SELECT     DISTINCT Pnumber
FROM        PROJECT, WORKS_ON, EMPLOYEE
WHERE       Pnumber= Pno AND Essn= Ssn AND Lname="Wong" );
```

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
QUERYS: SET OPERATIONS

- SQL has also the corresponding multiset operations (keyword **`ALL`**):
  - **`UNION ALL`**
  - **`EXCEPT ALL`**
  - **`INTERSECT ALL`**

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
QUERYS: ASSIGNMENT

- Retrieve the names of all students with Class 2 majoring in "CS" (computer science).
- Retrieve the names of all courses taught by Professor King in 2007 and 2008.
- For each section taught by Professor King, retrieve the course number, semester, year, and name of students who took the section.
- Retrieve the name and transcript of each student with Class 2 majoring in CS.
  A transcript includes course name, course number, credit hours, semester, year, and grade for each course completed by the student.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

MORE SQL
JOIN OF TABLES

- `<tablereference>` is a list of
  - Table names
  - Named subqueries
  - JOINed tables

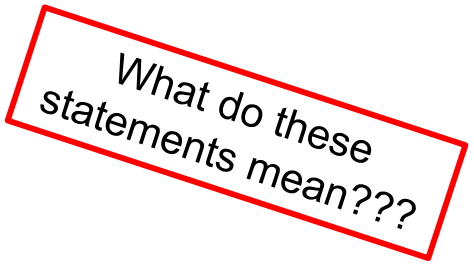
MORE SQL
JOIN OF TABLES

- Example SQL:

```
SELECT *
FROM Employee e , Department d
WHERE e.Dno = d.Dnumber ;
```

Equivalent Syntax:

```
SELECT *
FROM Employee e JOIN Department d ON e.Dno = d.Dnumber ;
```



Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
JOIN OF TABLES

- Different types of Join:
  - Default: inner Join
    → Only pairs of tuples that match the join condition are retrieved
  - Outer Join
    - Left Outer Join
      → Every tuple in the left table must appear in the result
    - Right Outer Join
      → Every tuple in the right table must appear in the result
    - Full outer Join
  - Natural Join
    → Join attributes have the same name
  - Cross Join
    → Cartesian Product

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
JOIN OF TABLES

Prof

Name	Title
Miller	PR1
Jones	DBS

Location

Title	Room
OO	304
DBS	120

Inner JOIN

Name	Title	Room
Jones	DBS	120

Outer JOIN

Name	Title	Room
Miller	PR1	NULL
Jones	DBS	120
NULL	OO	304

Left Outer JOIN

Name	Title	Room
Miller	PR1	NULL
Jones	DBS	120

Right Outer JOIN

Name	Title	Room
Jones	DBS	120
NULL	OO	304

MORE SQL
JOIN OF TABLES: LEFT OUTER JOIN

- Problem: All persons wanted
  → Also, persons not borrowing a book

```
SELECT * FROM Person AS p
       LEFT OUTER JOIN book AS b ON p.PNr = b.PNr ;
```

- Returns all tuples from the table on the left
- Missing values are filled with NULLs

Person

PNr	Name
123	Miller
456	Smith
789	Brown

Book

BNr	Title	PNr
1234	DB	123
4567	C	456
7894	IT	

MORE SQL
JOIN OF TABLES: RIGHT OUTER JOIN

- Problem: All books wanted
  → Also, books who are not borrowed by anyone

```
SELECT * FROM Person AS p
       RIGHT OUTER JOIN book AS b ON p.PNr = b.PNr ;
```

- Returns all tuples from the table on the right
- Missing values are filled with NULLs

MORE SQL
JOIN OF TABLES: FULL OUTER JOIN

Not supported by mySQL! But you can emulate them by union of left outer join and right outer join

- Problem: All persons and books wanted

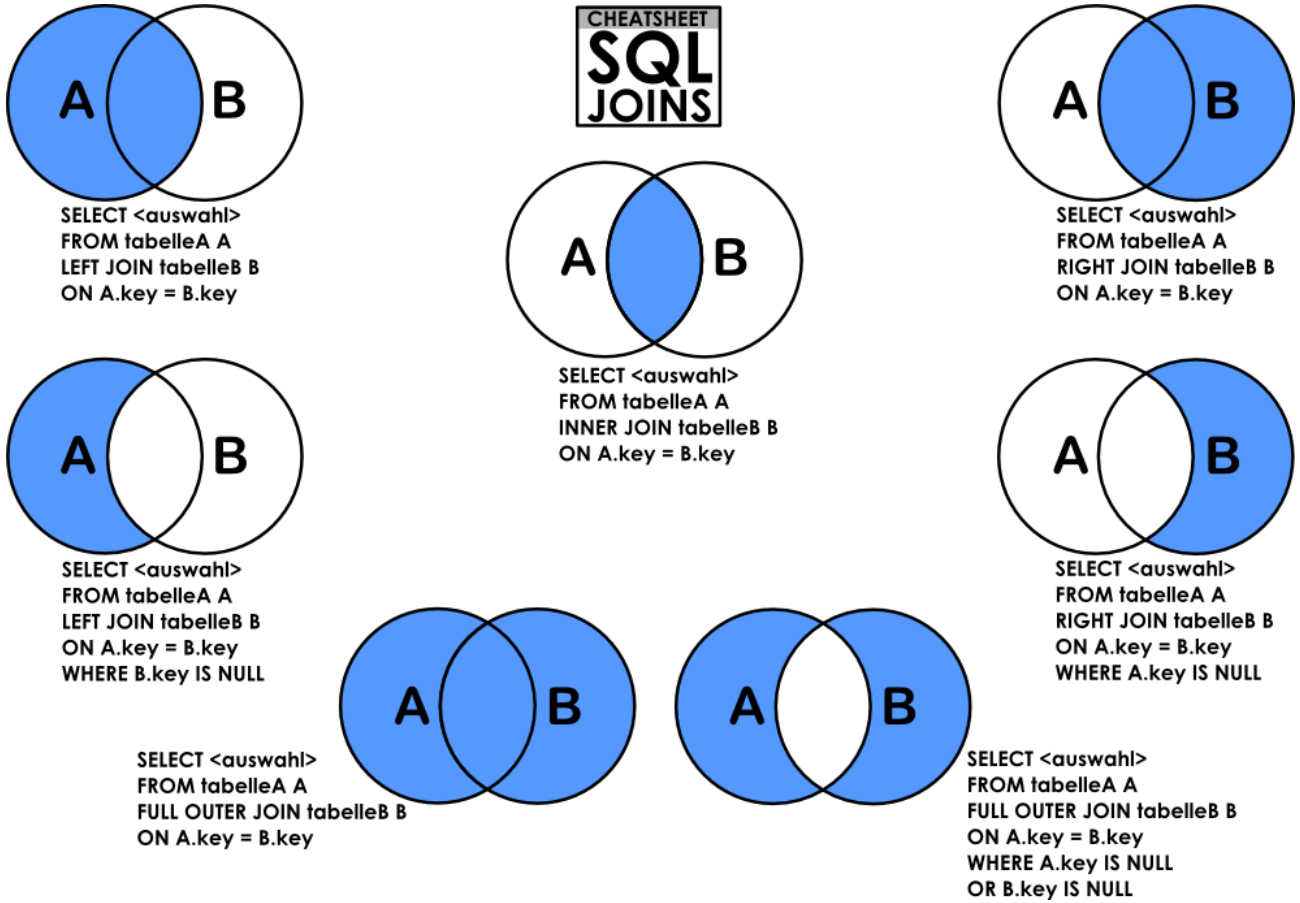
```
SELECT * FROM Person AS p
       FULL OUTER JOIN book AS b ON p.PNr = b.PNr ;
```

→ Missing values are filled with NULLs

- In mySQL:

```
SELECT * FROM Person AS p
       LEFT JOIN book AS b ON p.PNr = b.PNr
UNION
SELECT * FROM Person AS p
       RIGHT JOIN book AS b ON p.PNr = b.PNr
```

MORE SQL
JOIN OF TABLES: OVERVIEW



Source: https://stackoverflow.com/questions/59590346/trying-to-do-a-left-join-but-ending-up-getting-empty-result

MORE SQL
JOIN OF TABLES

Source: https://www.youtube.com/watch?v=7yvB-tTHRfQ

MORE SQL
JOIN OF TABLES: HTTPS://COMIC.BROWSERLING.COM/23

MORE SQL
SPECIAL FEATURES: SELECT WITHOUT A TABLE

- Sometimes we want to retrieve a value not connected to a table
  - Sequence value
  - System variable: current timestamp

- Example Oracle: special table dual
```
SELECT current timestamp FROM dual ;
```

- Example PostgreSQL: no FROM required
```
SELECT current timestamp ;
```

- Example mySQL: no FROM required
```
SELECT current_timestamp ;
```

MORE SQL
SPECIAL FEATURES: WHERE CLAUSE


Repetition

- Recap:
  - **`WHERE`** clause is optional
  - Contains logical expressions
    → Can be combined using AND, OR, NOT
  - Query for NULL values
    - `… WHERE a IS NULL;`
    - `… WHERE a IS NOT NULL;`

MORE SQL
SPECIAL FEATURES: WHERE CLAUSE

- Using **BETWEEN**

- Example: Search for a range of values
  → Age shall be between 18 and 21

```
... WHERE age >= 18 AND age <= 21 ;
... WHERE age BETWEEN 18 AND 21 ;
```

- BUT: Different DBMS differ on whether boundaries are included or not!

MORE SQL
SPECIAL FEATURES: WHERE CLAUSE

- Using **IN**
  → which compares a value v with a set (or multiset) of values V and evaluates to TRUE if v is one of the elements in V

- Example: Search for a set of values
  → Department ID shall be 4, 5, or 7

```
... WHERE DId = 4 OR DId = 5 OR Did = 7 ;
... WHERE DId IN ( 4 , 5 , 7 ) ;
```

- Also usable for set of values returned by a subquery

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
SPECIAL FEATURES: STRING PATTERN

- Searching for string patterns
  - Search for patterns using **`LIKE`** and wildcards
    - _(underscore): replaces a single character
    - % : replaces an arbitrary number of zero or more characters
  - Escape '\' for literals '%' and '_' in strings
    → E.g., 'AB_CD' represents the string "AB_CD"

```
... WHERE name LIKE 'M%' ;
```

```
'abc' LIKE 'abc'   → TRUE
'abc' LIKE 'a%'    → TRUE
'abc' LIKE '_b_'   → TRUE
'abc' LIKE 'c'     → FALSE
```

MORE SQL
SPECIAL FEATURES: COMPARISON WITH DATES

- The comparison with dates is DBMS dependent

- Example MySQL:

```
... WHERE birthdate = '2011-01-27' ;
-- Compare with year :
-- Cast to String
CAST (birthdate AS CHAR (30)) LIKE '2011%' ;
CONVERT (birthdate , CHAR (30)) LIKE '2011%' ;
-- Or
birthdate BETWEEN '2011-01-01' AND '2011-12-31' ;
-- user of date function
DATE_FORMAT (birthdate , '%Y') = '2011' ;
```

MORE SQL
SPECIAL FEATURES: COMPARISON WITH DATES

- The comparison with dates is DBMS dependent

- Example Oracle:

```
... WHERE birthdate = TO_DATE( '31-DEC-95' , 'DD-MON-YY' )
```

MORE SQL
SPECIAL FEATURES: WHERE CLAUSE → UPDATE, DELETE

- The where clause is the same for
  - **UPDATE**
  - **DELETE**
- Only one table

MORE SQL
SPECIAL FEATURES: SORTING OF RESULTS

- Results are (multi-)sets
  → No defined order!
- Order wanted: use ORDER BY
  - **`ASC`** (default): ascending order
  - **`DESC`**: descending order
  - Precondition: Datatype defines order
    - For **`VARCHAR`** it depends on locale
  - Ordering for more than one column is possible

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
SPECIAL FEATURES: SORTING OF RESULTS

- Example: Employees with same name are ordered by age

```
SELECT * FROM Employee
ORDER BY lname ASC ;
```

- Example: Combine order by function results and renamed attributes

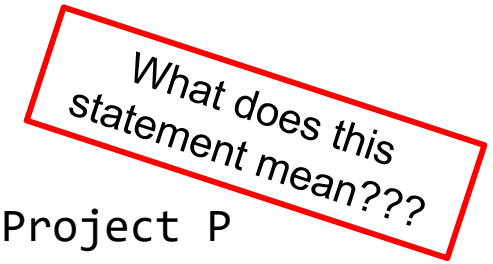
```
SELECT a+b AS sum FROM mytab ORDER BY sum ;
```

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
SPECIAL FEATURES: SORTING OF RESULTS

Example:

```
SELECT      D.Dname, E.Lname, E.Fname, P.Pname
FROM        Department D, Employee E, Works_on W, Project P
WHERE       D.Dnumber = E.Dno AND
            E.Ssn = W.Essn AND
            W.Pno = P.Pnumber
ORDER BY    D.Dname, E.Lname, E.Fname;
```



Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
SPECIAL FEATURES: AGGREGATE FUNCTIONS

- Summarize information from multiple tuples into a single-tuple summary
  - Analyze column values
  - Return one value for many rows (data reduction)
  - NULL values do not count!

  - **`COUNT`**, **`SUM`**, **`AVG`**, **`MAX`**, **`MIN`**
    - **`COUNT(*)`**: number of rows
    - **`COUNT(DISTINCT a)`**: count different values

  - Can be used in **`SELECT`** clause and **`HAVING`** clause

  - Attention: Not allowed in **`WHERE`** clause! (Exception: MySQL)

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
SPECIAL FEATURES: AGGREGATE FUNCTIONS

- Example

```
SELECT COUNT(*) FROM Book ;          → 4
SELECT COUNT(PNr) FROM Book ;        → 3
SELECT COUNT(DISTINCT PNr) FROM Book ;   → 2
SELECT MIN(Price) , MAX(Price) FROM Book ;      → 9.99 34.89
SELECT SUM(Price) FROM Book ;        → 64.87
SELECT AVG(Price) FROM Book ;        → 16.22
```

Book

PNr	Price	ISBN	Title
001	9.99	4711	DB easy
NULL	19.99	4712	DB Part 2
003	NULL	4714	Hitchhiker
003	34.89	4714	Hitchhiker

MORE SQL
SPECIAL FEATURES: GROUP BY

- Grouping is used to create subgroups of tuples before summarization
  → partition the relation into nonoverlapping subsets (or groups) of tuples
  - Using a grouping attribute
  - Grouping attribute should appear in the **SELECT** clause
  - If NULLs exist in the grouping attribute, then a separate group is created for all tuples with a NULL value

- Example: For each department, retrieve the department number, the number of employees in the department, and their average salary

```
SELECT     Dno, COUNT(*), AVG(Salary)
FROM       Employee
GROUP BY   Dno ;
```

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
SPECIAL FEATURES: GROUP BY

- Example:

Book

PNr	Price	ISBN	Title
001	9.99	4711	DB easy
NULL	19.99	4712	DB Part 2
003	NULL	4714	Hitchhiker
003	34.89	4714	Hitchhiker

How many books per person?

```
SELECT PNr , Count(*) AS BCOUNT
FROM   Book
GROUP BY PNr ;
```

New

PNr	BCOUNT
NULL	1
001	1
003	2

MORE SQL
SPECIAL FEATURES: GROUP BY

- Example:

Book

PNr	Price	ISBN	Title
001	9.99	4711	DB easy
NULL	19.99	4712	DB Part 2
003	NULL	4714	Hitchhiker
003	34.89	4714	Hitchhiker

How many books per person?

```
SELECT     PNr , Count(*) AS BCOUNT
FROM       Book
GROUP BY   PNr
ORDER BY   BCOUNT DESC ;
```

New

PNr	BCOUNT
003	2
001	1
NULL	1

MORE SQL
SPECIAL FEATURES: HAVING

- **HAVING** provides a condition on the summary information regarding the group of tuples associated with each value of the grouping attributes
  - Only the groups that satisfy the condition are retrieved in the result of the query
  - **HAVING** clause appears in conjunction with **GROUP BY** clause

- Note:
  - Selection conditions in **WHERE** clause limit the tuples
  - **HAVING** clause serves to choose whole groups

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
SPECIAL FEATURES: HAVING

- Example: We are interested only in persons who have borrowed more than one book
  → Need condition on groups

```
SELECT     PNr , count(*) AS BCOUNT
FROM       Book
GROUP BY   PNr
HAVING     count(*) > 1;
```

Alternative:

```
SELECT     PNr , count(*) AS BCOUNT
FROM       Book
GROUP BY   PNr
HAVING     BCOUNT > 1;
```

MORE SQL
SPECIAL FEATURES: HAVING

Example: For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project

```
SELECT      Pnumber, Pname, COUNT(*)
FROM        Project, Works_on
WHERE       Pnumber = Pno
GROUP BY    Pnumber, Pname
HAVING      COUNT(*) > 2;
```

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
SPECIAL FEATURES: HAVING

Example: For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project

```
SELECT      Pnumber, Pname, COUNT(*)
FROM        Project, Works_on, Employee
WHERE       Pnumber = Pno AND SSN = ESSN AND Dno = 5
GROUP BY    Pnumber, Pname
```

Source: Elmasri, Fundamentals of Database Systems, Page 97ff

MORE SQL
SPECIAL FEATURES: GROUP BY & HAVING

- Note: When using groups, only 2 types are allowed in **`SELECT`** clause:
  - Aggregate functions
  - Columns contained in **`GROUP BY`** clause

- **`HAVING`**: aggregate functions allowed

- Recall: no aggregate functions in **`WHERE`** clause

MORE SQL
SPECIAL FEATURES

- Specialities not treated in this lecture:
  - **`ANY`** / **`SOME`**
  - **`ALL`** in comparisons of a **`WHERE`** clause
  - **`EXISTS`** in a **`WHERE`** clause
  - **`UNIQUE`** in a **`WHERE`** clause
  - Nested queries

MORE SQL
SPECIAL FEATURES: ASSIGNMENT

- How many students are studying CS?
- List all course names and how often they have been taught.
- For each section taught by Professor Anderson, retrieve the course number, semester, year, and number of students who took the section.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

MORE SQL
QUERY: SUMMARY

- Syntax:

```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```

MORE SQL
QUERY: EXECUTION ORDER

- Order of Execution:

FROM	Cartesian Product, JOIN
WHERE	Selection
GROUP BY	Grouping
HAVING	Condition on groups
ORDER BY	Sorting
SELECT	Projection

- Use same order when you build a query
- Optimizer may choose another exec order