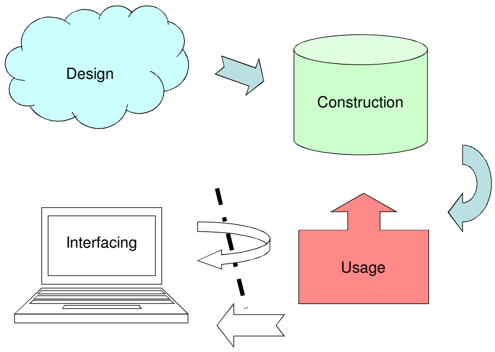
Course Objectives



Database Construction and Usage

SQL DDL and DML
Relational Algebra

Course Objectives – Construction

When the course is through, you should

- Given a database schema with related constraints, implement the database in a relational (SQL) DBMS

SQL Data Definition Language

Case convention

- SQL is completely case insensitive. Upper-case or Lower-case makes no difference. We will use case in the following way:
  - **UPPERCASE** marks keywords of the SQL language.
  - **lowercase** marks the name of an attribute.
  - **Capitalized** marks the name of a table.

Creating and dropping tables

- Relations become tables, attributes become columns.

```
CREATE TABLE tablename (
  <list of table elements>
);
```

- Get all info about a created table:

```
DESCRIBE tablename;
```

Oracle specific!

- Remove a created table:

```
DROP TABLE tablename;
```

Table declaration elements

- The basic elements are pairs consisting of a column name and a type.
- Most common SQL types:
  - INT or INTEGER (synonyms)
  - REAL or FLOAT (synonyms)
  - CHAR(*n*) = fixed-size string of size *n*.
  - VARCHAR(n) = variable-size string of up to size *n*.

Example

Example:

```
CREATE TABLE Courses (
  code CHAR(6),
  name VARCHAR(50)
);
```

Created the table courses:

code	name

Declaring keys

- An attribute or a list of attributes can be declared PRIMARY KEY or UNIQUE
  - PRIMARY KEY: At most one per table, never NULL. Efficient lookups in all DBMS.
  - UNIQUE: Any number per table, can be NULL. Could give efficient lookups (may vary in different DBMS).
- Both declarations state that all other attributes of the table are functionally determined by the given attribute(s).

Example

```
CREATE TABLE Courses(
  code CHAR(6),
  name VARCHAR(50),
  PRIMARY KEY (code)
);
Or
CREATE TABLE Courses(
  code CHAR(6),
  name VARCHAR(50),
CONSTRAINT CoursesPK PRIMARY KEY (code)
);
```

Foreign keys

- Referential constraints are handled with references, called *foreign keys*:

  FOREIGN KEY *attribute*
  REFERENCES *table(attribute)*

Foreign keys

- General:

```
FOREIGN KEY course REFERENCES Courses(code)
```

- If course is Primary Key in Courses:

```
FOREIGN KEY course
    REFERENCES Courses
```

- Give a name to the foreign key:

```
CONSTRAINT ExistsCourse
FOREIGN KEY course
REFERENCES Courses
```

Example

```
CREATE TABLE GivenCourses (
    code            CHAR(6),
    period          INT,
    numStudents     INT,
    teacher         VARCHAR(50),
    PRIMARY KEY (code, period),
    FOREIGN KEY (code) REFERENCES  Courses(code)
);

CREATE TABLE GivenCourses (
    code            CHAR(6) REFERENCES Courses(code),
    period          INT,
    numStudents     INT,
    teacher         VARCHAR(50),
    PRIMARY KEY (code, period)
);
```

Value constraints

- Use CHECK to insert simple value constraints.
  - CHECK (*some test on attributes*)

```
CREATE TABLE GivenCourses (
    code            CHAR(6),
    period          INT CHECK (period IN (1,2,3,4)),
    numStudents     INT,
    teacher         VARCHAR(50),
    FOREIGN KEY (code) REFERENCES Courses(code),
    PRIMARY KEY (code, period)
);
```

Naming constraints

- Default error messages are horrible.
- Naming constraints makes them a lot easier to read and understand.

CONSTRAINT *constraint-name constraint*

```
CONSTRAINT ValidPeriod
  CHECK (period in (1,2,3,4))
```

Example

```
CREATE TABLE GivenCourses (
    code        CHAR(6) REFERENCES Courses(code),
    period      INT,
    numStudents INT,
    teacher     VARCHAR(50),
    PRIMARY KEY (code, period),
    CONSTRAINT ValidPeriod CHECK (period in (1,2,3,4))
);
```

Example

- **Legal:**
  - `INSERT INTO GivenCourses VALUES ('TDA357',4,93,'Rogardt);`
- **Not Legal:**
  - `INSERT INTO GivenCourses VALUES ('TDA357',7,93,'Rogardt);`
  - **ERROR at line 1:**
    - **ORA-02290: check constraint (NIBRO.VALIDPERIOD) violated**

Example: DESCRIBE

```
CREATE TABLE GivenCourses (
    code        CHAR(6) REFERENCES Courses(code),
    period      INT,
    numStudents INT,
    teacher     VARCHAR(50),
    PRIMARY KEY(code,period),
    CONSTRAINT ValidPeriod CHECK (period in (1,2,3,4))
);
```

DESCRIBE GivenCourses;

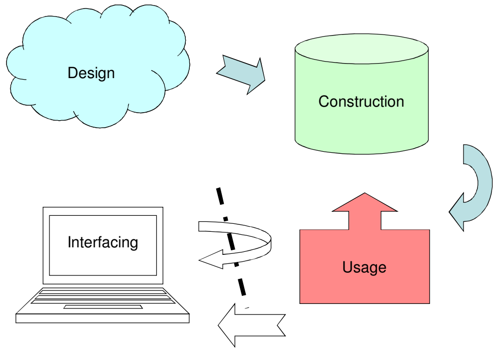
Name	Null?	Type
CODE	NOT NULL	CHAR(6)
PERIOD	NOT NULL	NUMBER(38)
NUMSTUDENTS		NUMBER(38)
TEACHER		VARCHAR2(50)

Exam – SQL DDL

”A grocery store wants a database to store information about products and suppliers. After studying their domain you have come up with the following database schema. …”

- Write SQL statements that create the relations as tables in a DBMS, including all constraints.

Course Objectives



SQL Data Manipulation Language: Modifications

Course Objectives – Usage

When the course is through, you should

- Know how to change the contents of a database using SQL

Inserting data

```
INSERT INTO tablename
  VALUES (values for attributes);

INSERT INTO Courses
  VALUES ('TDA357', 'Databases');
```

code	name
TDA357	Databases

Inserting data (alt.)

```
INSERT INTO tablename
  (some of the attributes)
    VALUES (values for attributes);

INSERT INTO Courses
  (name, code)
    VALUES ('Databases', 'TDA357');
```

code	name
TDA357	Databases

Deletions

```
DELETE FROM tablename
  WHERE test over rows;

DELETE FROM Courses
  WHERE code = 'TDA357';

DELETE FROM Courses;
```

Quiz

code	name
TDA357	Databases
TIN090	Algorithms

↓ DELETE FROM Courses
WHERE code = 'TDA357';

code	name
TIN090	Algorithms

Quiz: What does this statement do?

DELETE FROM Courses;

Updates

```
UPDATE tablename
SET    attribute = ...
WHERE  test over rows

UPDATE GivenCourses
SET    teacher = 'Rogardt Heldal'
WHERE  code = 'TDA357'
  AND  period = 4;
```

Quiz

code	per	#st	teacher
TDA357	2	87	Niklas Broberg
TDA357	4	93	Marcus Björkander
TIN090	1	64	Devdatt Dubhashi

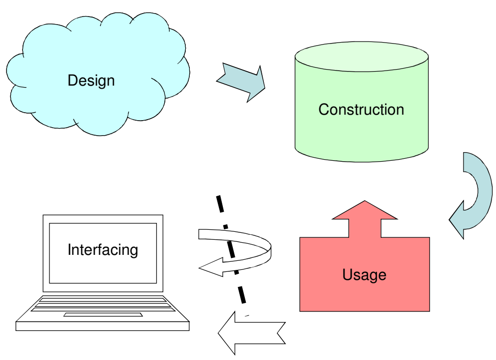
↓ UPDATE GivenCourses
SET teacher = 'Rogardt Heldal'
WHERE code = 'TDA357'
AND period = 4;

code	per	#st	teacher
TDA357	2	87	Niklas Broberg
TDA357	4	93	Rogardt Heldal
TIN090	1	64	Devdatt Dubhashi

Summary

- SQL Data Definition Language
  - **CREATE TABLE**, attributes
  - Constraints
    - **PRIMARY KEY**
    - **FOREIGN KEY … REFERENCES**
    - **CHECK**
- SQL Data Manipulation Language
  - **INSERT, DELETE, UPDATE**

Course Objectives



Course Objectives – Usage

When the course is through, you should

- Know how to query a database for relevant data using SQL

Queries: SQL and Relational Algebra

Querying

- To *query* the database means asking it for information.
  - ”List all courses that have lectures in room VR”
- Unlike a modification, a query leaves the database unchanged.

”Algebra”

- An *algebra* is a mathematical system consisting of:
  - Operands: variables or values to operate on.
  - Operators: symbols denoting functions that operate on variables and values.

Relational Algebra

- An algebra whose operands are relations (or variables representing relations).
- Operators representing the most common operations on relations.
  - Selecting rows
  - Projecting columns
  - Composing (joining) relations

Relational operators (1)

- Selection
  - Choose <u>rows</u> from a relation
  - State condition that rows must satisfy

$$\sigma_{condition}(T)$$

Examples:

$\sigma_{seats>100}(Rooms)$

$\sigma_{(code=\text{"TDA143" AND day="Friday"})}(Lectures)$

Relational operators (2)

- Projection
  - Choose <u>columns</u> from a relation
  - State which columns (attributes)

$\pi_A(T)$

Examples:

π_{code}(Courses)

$\pi_{name,seats}$(Rooms)

Relational operators (3)

$R_1 \times R_2$
- Cartesian product
- Combine each row of R_1 with each row of R_2

$R_1 \bowtie_{condition} R_2$
- join operator
- Combine row of R_1 with each row of R_2 if the condition is true

$R_1 \bowtie_{condition} R_2 = \sigma_{condition}(R_1 \times R_2)$

SQL

- SQL = Structured Query Language
  - The querying parts are really the core of SQL. The DDL and DML parts are secondary.
- Very-high-level language.
  - Specify *what* information you want, not *how* to get that information (like you would in e.g. Java).
- Based on Relational Algebra

The Query Compiler

- SQL query is parsed to produce a parse tree that represents the query.
- Parse tree is transformed to a relational algebra expression tree (or similar).
- Generate a physical query plan.
  - Use algebraic laws to improve query plan by generating many alternative execution plans and estimating their cost.
  - Choose algorithm to perform each step.

Selection

- Selection = Given a relation (table), choose what tuples (rows) to include in the result.

$\sigma_C(T)$

```
SELECT * FROM T WHERE C;
```

- Select the rows from relation T that satisfy condition C.
- σ = sigma = greek letter **s** = **s**election

Example:

GivenCourses =

course	*per*	*teacher*
TDA357	2	Niklas Broberg
TDA357	4	Rogardt Heldal
TIN090	1	Devdatt Dubhashi

```
SELECT *
FROM   GivenCourses
WHERE  course = 'TDA357';
```

Result =

What?

Projection

- Given a relation (table), choose what attributes (columns) to include in the result.

$\pi_X(\sigma_C(T))$ `SELECT X FROM T WHERE C;`

  – Select the rows from table T that satisfy condition C, and project columns X of the result.
  – π = pi = greek letter **p** = **p**rojection

Example:

GivenCourses =

course	*per*	*teacher*
TDA357	2	Niklas Broberg
TDA357	4	Rogardt Heldal
TIN090	1	Devdatt Dubhashi

```
SELECT course, teacher
FROM   GivenCourses
WHERE  course = 'TDA357';
```

Result =

What?

The confusing **SELECT**

Example:

GivenCourses =

course	*per*	*teacher*
TDA357	2	Niklas Broberg
TDA357	4	Rogardt Heldal
TIN090	1	Devdatt Dubhashi

```
SELECT course, teacher
FROM   GivenCourses;
```

Result =

What?

Quiz: **SELECT** is a projection??

Mystery revealed!

```
SELECT code, teacher
  FROM GivenCourses;
```

$\pi_{code,teacher}(\sigma(GivenCourses))$
$= \pi_{code,teacher}(GivenCourses)$

- In general, the SELECT clause could be seen as corresponding to projection, and the WHERE clause to selection (don't confuse the naming though).

Quiz!

- What does the following expression compute?

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

GivenCourses

course	*per*	*teacher*
TDA357	2	Niklas Broberg
TDA357	4	Rogardt Heldal
TIN090	1	Devdatt Dubhashi

```
SELECT *
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg';
```

FROM Courses, GivenCourses

code	name	course	per	teacher
TDA357	Databases	TDA357	2	Niklas Broberg
TDA357	Databases	TDA357	4	Rogardt Heldal
TDA357	Databases	TIN090	1	Devdatt Dubhashi
TIN090	Algorithms	TDA357	2	Niklas Broberg
TIN090	Algorithms	TDA357	4	Rogardt Heldal
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi

WHERE teacher = 'Niklas Broberg'

code	name	course	per	teacher
TDA357	**Databases**	**TDA357**	**2**	**Niklas Broberg**
TDA357	Databases	TDA357	4	Rogardt Heldal
TDA357	Databases	TIN090	1	Devdatt Dubhashi
TIN090	**Algorithms**	**TDA357**	**2**	**Niklas Broberg**
TIN090	Algorithms	TDA357	4	Rogardt Heldal
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi

Answer:

```
SELECT *
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg';
```

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	2	Niklas Broberg
TIN090	Algorithms	TDA357	2	Niklas Broberg

The result is all rows from `Courses` combined in all possible ways with all rows from `GivenCourses`, and then keep only those where the `teacher` attribute is Niklas Broberg.

Cartesian Products

- The *Cartesian product* of relations R_1 and R_2 is all possible combinations of rows from R_1 and R_2.
  - Written $R_1 \times R_2$
  - Also called *cross-product*, or just *product*

```
SELECT *
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg';
```

Quiz: Translate to a Relational Algebra expression.

Quiz!

List all courses, with names, that Niklas Broberg is responsible for.

```
Courses(code,name)
GivenCourses(course,per,teacher)
   course -> Courses.code
```

```
SELECT *
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg'
  AND  code = course;
```

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	2	Niklas Broberg

code = course

code	name	course	per	teacher
TDA357	**Databases**	**TDA357**	**2**	**Niklas Broberg**
TDA357	Databases	TDA357	4	Rogardt Heldal
TDA357	Databases	TIN090	1	Devdatt Dubhashi
TIN090	**Algorithms**	**TDA357**	**2**	**Niklas Broberg**
TIN090	Algorithms	TDA357	4	Rogardt Heldal
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi

Not equal

Joining relations

- Very often we want to join two relations on the value of some attributes.
  - Typically we join according to some reference, as in:

```
SELECT *
FROM   Courses, GivenCourses
WHERE  code = course;
```

- Special operator $\bowtie_C$ for joining relations.

$$R_1 \bowtie_C R_2 = \sigma_C(R_1 \times R_2)$$

```
SELECT *
FROM   R1 JOIN R2 ON C;
```

Example

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

GivenCourses

course	*per*	*teacher*
TDA357	2	Niklas Broberg
TDA357	4	Rogardt Heldal
TIN090	1	Devdatt Dubhashi

```
SELECT *
FROM   Courses JOIN GivenCourses
  ON   code = course;
```

What?

Natural join

- ”Magic” version of join.
  - Join two relations on the condition that all attributes in the two that share the same name should be equal.
  - Remove all duplicate columns
  - Written $R_1 \bowtie R_2$ (like join with no condition)

Example

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

GivenCourses

code	*per*	*teacher*
TDA357	2	Niklas Broberg
TDA357	4	Rogardt Heldal
TIN090	1	Devdatt Dubhashi

```
SELECT *
FROM   Courses NATURAL JOIN GivenCourses;
```

What?

Outer join

- Compute the join as usual, but retain all tuples that don't fit in from either or both operands, padded with NULLs.

$R_1 \overset{\circ}{\bowtie} R_2$

```
SELECT *
FROM
 R1 NATURAL FULL OUTER JOIN R2;
```

  - FULL means retain all tuples from both operands. LEFT or RIGHT retains only those from one of the operands.
  - Can be used with ordinary join as well.
    - R_1 LEFT OUTER JOIN R_2 ON C;

Quiz!

List all courses and the periods they are given in. Courses that are not scheduled for any period should also be listed, but with NULL in the field for period.

```
SELECT code, period
FROM   Courses LEFT OUTER JOIN GivenCourses
         ON code = course;
```

```
SELECT code, period
FROM   Courses
       LEFT OUTER JOIN
       GivenCourses
       ON code = course;
```

code	*name*
TIN090	Algorithms
TDA590	OOS
TDA357	Databases
TDA100	AI

course	period	teacher	#students
TDA357	2	Niklas Broberg	130
TDA357	4	Rogardt Heldal	135
TIN090	1	Devdatt Dubhashi	95
TDA590	2	Rogardt Heldal	70

```
SELECT code, period
FROM   Courses
       LEFT OUTER JOIN
       GivenCourses
       ON code = course;
```

code	period
TDA357	2
TDA357	4
TIN090	1
TDA590	2
TDA100	NULL

Sets or Bags?

- Relational algebra formally applies to sets of tuples.
- SQL, the most important query language for relational databases is actually a bag language.
  - SQL will eliminate duplicates, but usually only if you ask it to do so explicitly.
- Some operations, like projection, are much more efficient on bags than sets.

Relational Algebra on Bags

- A *bag* is like a set, but an element may appear more than once.
  - *Multiset* is another name for bag
- Example: {1,2,1,3} is a bag. {1,2,3} is also a bag that happens to be a set.
- Bags also resemble lists, but order in a bag is unimportant.
  - Example: {1,2,1} = {1,1,2} as bags, but [1,2,1] != [1,1,2] as lists.

Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets.
- Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates.
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Quiz

R(A,B)

A	B
1	2
5	6
1	3

SELECT A
FROM R ???

$\pi_A(R)$???

SELECT-FROM-WHERE

- Basic structure of an SQL query:

```
SELECT attributes
FROM   tables
WHERE  tests over rows
```

```
SELECT X
FROM   T
WHERE  C
```

⟺ $\pi_X(\sigma_C(T))$

Example:

```
SELECT code, name, period
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg'
       AND code = course;
```

Courses

code	name
TDA357	Databases
TIN090	Algorithms

GivenCourses

course	per	teacher
TDA357	2	Niklas Broberg
TDA357	4	Rogardt Heldal
TIN090	1	Devdatt Dubhashi

$$\pi_{code,name,period}(\sigma_{teacher='Niklas\ Broberg'\ \&\ code\ =\ course}(Courses\ x\ GivenCourses))$$

Example:

```
SELECT code, name, period
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg'
       AND code = course;
```

code	name	course	per	teacher
TDA357	Databases	TDA357	2	Niklas Broberg
TDA357	Databases	TDA357	4	Rogardt Heldal
TDA357	Databases	TIN090	1	Devdatt Dubhashi
TIN090	Algorithms	TDA357	2	Niklas Broberg
TIN090	Algorithms	TDA357	4	Rogardt Heldal
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi

$\pi_{code,name,period}(\sigma_{teacher='Niklas\ Broberg'\ \&\ code\ =\ course}$(**Courses x GivenCourses**))

Example:

```
SELECT code, name, period
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg'
       AND code = course;
```

code	name	course	per	Teacher
TDA357	Databases	TDA357	2	Niklas Broberg
TDA357	Databases	TDA357	4	Rogardt Heldal
TDA357	Databases	TIN090	1	Devdatt Dubhashi
TIN090	Algorit...			
TIN090	Algorit...			
TIN090	Algorit...			

code	name	course	per	teacher
TDA357	Databases	TDA357	2	Niklas Broberg

$\pi_{code,name,period}$(**$\sigma_{teacher='Niklas\ Broberg'\ \&\ code\ =\ course}$**(Courses x GivenCourses)**)**

Example:

```
SELECT code, name, period
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg'
       AND code = course;
```

code	name	course	per	teacher
TDA357	Databases	TDA357	2	Niklas Broberg

code	name	per
TDA357	Databases	2

$\pi_{code,name,period}$ $(\sigma_{teacher='Niklas\ Broberg'\ \&\ code\ =\ course}(Courses\ x\ GivenCourses))$

Quiz!

What does the following relational algebra expression compute?

$$\sigma_{teacher='Niklas\ Broberg'\ \&\ code\ =\ course}(\pi_{code,name,period}(Courses\ x\ GivenCourses))$$

The expression is invalid, since the result after the projection will not have attributes teacher and course to test.

More complex expressions

- So far we have only examples of the same simple structure: $\pi_X(\sigma_C(T))$
- We can of course combine the operands and operators of relational algebra in (almost) any way imaginable.

$$\sigma_C(R_3 \bowtie_D \pi_X(R_1 \times R_2))$$

```
SELECT *
FROM   R3 JOIN (SELECT X FROM R1,R2) ON D
WHERE  C
```

Summary so far

- SQL is based on relational algebra.
- Operations for:
  - Selection of rows
  - Projection of columns
  - Combining tables
    - Cartesian product
    - Join, natural join
- Bags/Sets semantics

- Much more to come!

Next Lecture

More Relational Algebra and SQL

Assignment Part II – Construction and Usage

- Implement your design from part I by creating tables in Oracle for your relations. Be sure to include all extra constraints.
- Create views and triggers that simplify key operations of the system.
- Fill your tables with data that stress-tests your implementation.