1. CREATE TABLE

```
CREATE TABLE <name of table>
(
        <name of attribute> <type of attribute> <integrity constraints>,
        <name of attribute> <type of attribute> <integrity constraints>, …
);
```

<placeholder>

Example:

```
CREATE TABLE Courses (
    CourseNo INTEGER,
    Name VARCHAR(10),
    Room CHARACTER(5)
);
```

Integrity Constraints

- NOT NULL (data cannot be empty)
- UNIQUE (cannot be null, unique value in the table)

- PRIMARY KEY (identifier of the entity and unique)
- FOREIGN KEY (identifier of an entity of another table)

- DEFAULT (default value of a column)
- AUTO_INCREMENT (automatically set to 'last value + 1')

- Combinations possible

```
CREATE TABLE Lectures (
CourseNo INTEGER PRIMARY KEY AUTO_INCREMENT
);
```

2. ALTER TABLE

ALTER TABLE <table name> <specification>

Specification:

- ADD <col name> <col definition>
- CHANGE <old col name> <new col name> <col definition>
- MODIFY <col name> <col definition>
- DROP <col name>

- Example:
  ALTER TABLE courses ADD moodle boolean not null;

3. RENAME

- RENAME TABLE <table_name> TO <new_table_name>

4. DROP

- DROP TABLE <table_name>,...

5. INSERT INTO

INSERT INTO <table_name> VALUES (<value1>,<value2>,<value3>,...);

- Values in order of columns

INSERT INTO <table_name> (<column1>,<column2>,<column3>,...)
VALUES (<value1>,<value2>,<value3>,...);

- Values in desired order

Example:

```
INSERT INTO Courses VALUES(1,'EngDB','N0199');
```

6. SELECT

SELECT <columns> FROM <data source> WHERE <condition> […]

Examples:

```
SELECT name FROM Courses
SELECT * FROM Courses
SELECT * FROM Courses WHERE Room='N0199'
SELECT * FROM Courses WHERE Room='M0199'
```

7. UPDATE

- UPDATE <table name>
  SET <col name>=<col value> [,<col name>=<col value>,…]
  [WHERE <condition>]

Example:

- UPDATE instructors SET department='Cms' WHERE name='Bob'

- DELETE FROM <table_name> [WHERE …]
- WHERE <columnName>=<value> [and | or <columnName>=<value>]

Example

- DELETE FROM instructors WHERE name='Bob' and department='Cms'
- DELETE FROM instructors

Relational Algebra

1. Projection Π

In general: $\Pi_{columns}$ (Relation)

Example: Π_{name} (instructors)

SELECT name FROM instructors

id	name	department
1	Tom	CMS
2	Alice	CiE
3	Bob	CMS

Π_{name} (instructors)

name
Tom
Bob
Alice

Some rules of the $\Pi_{columns}$ (Relation) operator

- Π_a(Relation) = Π_a(Π_a(Relation))
- Π_a(Π_b(Relation)) = Π_a(Relation) if $a \subseteq b$

id	name	department
1	Tom	CMS
2	Alice	CiE
3	Bob	CMS

- Example of a nested statement:
  SELECT department from (select department from instructors) as instructorAlias
  The result of (…) is named instructorAlias

Exercise (Students → 7 mins):
$\Pi_{department}$(instructors),
Π_{name} (Π_{name} (instructors)),
Π_{name} ($\Pi_{name, department}$ (instructors)),
$\Pi_{name, department}$ ($\Pi_{department}$ (instructors)) (error! why?)
$\Pi_{name, department}$ (Π_{id} (instructors)) (error! why?)

```
select department from lec3__instructors;
select name from (select name from lec3__instructors) as instructorAlias;
select name from (select name, department from lec3__instructors) as instructorAlias;
select name, department from (select department from lec3__instructors) as instructorAlias; #Unknown column 'name' in 'field list'
select name, department from (select id from lec3__instructors) as instructorAlias; #Unknown column 'name' in 'field list'
```

2. Union U

Example:

$R \cup S$

instructors ∪ professors

SELECT * FROM professors UNION SELECT * FROM instructors

$\Pi_{name, department}$ (instructors) ∪ $\Pi_{name, department}$ (professors)

instructors relation

id	name	department
1	Tom	CMS
2	Alice	CiE
3	Bob	CMS

professors relation

id	name	department
10	Borrmann	CMS
30	Rank	CiE

Union

id	name	department
10	Borrmann	CMS
30	Rank	CiE
1	Tom	CMS
3	Bob	CMS
2	Alice	CiE

3. Selection σ

Example: $\sigma_{department=CMS}(instructors)$

SELECT * FROM instructors WHERE department='CMS'

id	name	department
1	Tom	CMS
2	Alice	CiE
3	Bob	CMS

$\sigma_{department=CMS}(instructors)$

id	name	department
1	Tom	CMS
3	Bob	CMS

Some rules of the $\sigma_{statement}(Relation)$ operator

id	name	department
1	Tom	CMS
2	Alice	CiE
3	Bob	CMS

- $\sigma_a(Relation) = \sigma_a(\sigma_a(Relation))$
- $\sigma_a(\sigma_b(Relation)) = \sigma_b(\sigma_a(Relation))$
- $\sigma_{a \text{ and } b}(Relation) = \sigma_a(\sigma_b(Relation))$
- $\sigma_{a \text{ or } b}(Relation) = \sigma_a(Relation) \cup \sigma_b(Relation)$

- Example for nested statements:
  SELECT * FROM (SELECT * FROM instructors WHERE department= 'CMS')
  AS instructorsAlias WHERE department='CMS'

Exercise:

1. $\sigma_{department=CMS}(instructors)$, 2. $\sigma_{department=CMS}(\sigma_{department=CMS}(instructors))$
3. $\sigma_{department=CMS}(\sigma_{name=Tom}(instructors))$, 4. $\sigma_{name=Tom}(\sigma_{department=CMS}(instructors))$,
5. $\sigma_{department=CMS \text{ and } name!=Tom}(instructors)$, 6. $\sigma_{department=CMS}(instructors) \cup \sigma_{name!=Tom}(instructors)$, and 7. $\sigma_{department=CMS \text{ or } name=Tom}(instructors)$

```
select * from lec3__instructors WHERE department='CMS'
select * from (select * from lec3__instructors where department='CMS') as l3Instructors where department='CMS';
select * from (select * from lec3__instructors where name='Tom') as l3Instructors where department='CMS';
select * from (select * from lec3__instructors where department='CMS') as l3Instructors where name='Tom';
select * from lec3__instructors where name!='Tom' and department='CMS';
select * from lec3__instructors where name!='Tom'union select * from lec3__instructors wheredepartment='CMS';
select * from lec3__instructors where name='Tom' or department='CMS';
```

Combination of Selection and Projection

In general: $\Pi_{columns}(\sigma_{statement}(Relation))$ or $\sigma_{statement}(\Pi_{columns}(Relation))$

Example:

$\Pi_{name}(\sigma_{department=CMS}(instructors))$

id	name	department
1	Tom	CMS
2	Alice	CiE
3	Bob	CMS

name
Tom
Bob

$\sigma_{department=CMS}(instructors)$

id	name	department
1	Tom	CMS
3	Bob	CMS

name
Tom
Alice
Bob

$\Pi_{name}(instructors)$

$\sigma_{department=CMS}(\Pi_{name}(instructors))$

$\sigma_{department=CMS}(\Pi_{name,department}(instructors))$

name	department
Tom	CMS
Bob	CMS

4. Cartesian product ×

In general: R × S

- Contains | R | · | S | combinations (all) of tuples from R and S.

The SQL syntax for the Cartesian product:

<Relation 1> CROSS JOIN <Relation 2>

or

<Relation 1> JOIN <Relation 2>

- Example:
  instructors × offices
  SELECT * FROM instructors CROSS JOIN offices
  SELECT * FROM instructors JOIN offices

instructors	
name	chair
Bob	CMS
Tom	CIE

×

offices	
room	size
005	14
008	25

=

instructors × offices			
name	chair	room	size
Bob	CMS	005	14
Tom	CIE	005	14
Bob	CMS	008	25
Tom	CIE	008	25

5. Renaming ρ

- Problem occurs when using the same relation twice.
- E.g. the instructors × instructors will fail

~~SELECT * FROM instructors JOIN instructors~~

In general: $\rho_{\ new\ name}$ (Relation)

- The SQL syntax for the renaming is:
  <Relation> AS <New Name>

Example:

$\rho_{\ inst1}(Instructors) \times \rho_{\ inst2}(Instructors)$

SELECT * FROM instructors AS inst1 JOIN instructors AS inst2

6-7.θ-Join and Equi-Join

θ-Join $\bowtie_\theta$ and equi-Join $\bowtie_{\theta\ is\ =}$

In general: $R \bowtie_\theta S$ is a θ-Join. Also: $R \bowtie_\theta S = \sigma_\theta(R \times S)$

The SQL syntax for the Cartesian product:

SELECT * FROM <relation1> JOIN < relation2 >
ON <condition1> [AND|OR more Conditions]

or

SELECT * FROM <table1> INNER JOIN <table2>
ON <condition1> [AND|OR more Conditions]

θ -Join

- Example: instructors $\bowtie_{instructors.name!=students.name}$ students
  SELECT * FROM instructors JOIN students ON instructors.name != students.name

$R \times S$

instructors	
name	chair
Bob	CMS
Tom	CIE

×

students	
name	major
Alice	Math
Bob	Sports

instructors × students			
name	chair	name	major
Bob	CMS	Alice	Math
Tom	CIE	Alice	Math
Bob	CMS	Bob	Sports
Tom	CIE	Bob	Sports

$\sigma_\theta(R \times S)$

instructors × students			
name	chair	name	major
Bob	CMS	Alice	Math
Tom	CIE	Alice	Math
Bob	CMS	Bob	Sports
Tom	CIE	Bob	Sports

σ_θ

σ_θ(instructors × students)			
name	chair	name	major
Bob	CMS	Alice	Math
Tom	CIE	Alice	Math
Tom	CIE	Bob	Sports

Equi-Join

- Example: instructors $\bowtie_{instructors.name=students.name}$ students
  SELECT * FROM instructors JOIN students ON instructors.name = students.name
  or
  SELECT * FROM instructors INNER JOIN students ON
  instructors.name = students.name
  or
  SELECT * FROM instructors CROSS JOIN students ON
  instructors.name = students.name

instructors	
name	chair
Bob	CMS
Tom	CIE

$\bowtie_\theta$

students	
name	major
Alice	Math
Bob	Sports

instructors⋈θstudents		
name	chair	major
Bob	CMS	Sports

- Including renaming: select * from instructors as i join students s on i.name = s.name

8. Natural Join ⋈

In general: $R \bowtie S$

The SQL syntax for the natural join

SELECT * FROM <relation1> NATURAL JOIN <relation2 >

Example:

instructors ⋈ students
SELECT * FROM instructors NATURAL JOIN students

9. Semi Join Right ⋉ and left semi join ⋊

In general: left semi join $R \ltimes S = \Pi_R(R \bowtie S)$, right semi join: $R \rtimes S = \Pi_S(R \bowtie S)$

The SQL syntax for the left semi join
SELECT <r1>.* FROM <r1> NATURAL JOIN <r2>

The SQL syntax for the right semi join
SELECT <r2>.* FROM <r1> NATURAL JOIN <r2>

- Example:
  instructors ⋉ students (left semi join)
  instructors ⋊ students (right semi join)

instructors	
name	chair
Bob	CMS
Tom	CIE

students	
name	major
Alice	Math
Bob	Sports

→

instructors⋉ students	
name	chair
Bob	CMS

instructors⋊ students	
name	major
Bob	Sports

10. Outer Join

In general: left outer join⟕, right outer join ⟖, and full outer join ⟗

The SQL syntax for the outer joins:
left outer join: SELECT * FROM <r1> LEFT JOIN <r2> ON <Condition>
right outer join: SELECT * FROM <r1> RIGHT JOIN <r2> ON <Condition>
full outer join: <left outer join> UNION [ALL] <right outer join>

- Example: (on name)
  instructors ⟖ students (right outer join)
  instructors ⟕ students (left outer join)
  instructors ⟗ students (full outer join)

instructors	
name	chair
Bob	CMS
Tom	CIE

students	
name	major
Alice	Math
Bob	Sports

instructors⟖ students			
name	chair	name	major
Bob	CMS	Bob	Sports
NULL	NULL	Alice	Math

instructors⟕ students			
name	chair	name	major
Bob	CMS	Bob	Sports
Tom	CIE	NULL	NULL

Instructors ⟗ students			
name	chair	name	major
Bob	CMS	Bob	Sports
Tom	CIE	NULL	NULL
Bob	CMS	Bob	Sports
NULL	NULL	Alice	Math

Explicit Renaming

- Remember: Renaming a relation/table <relation> as <new Name>
- Renaming an attribute/column
  - Operator: $\rho_{new\ name/old\ name}(relation)$
  - The SQL syntax for the renaming an attribute/column:
    SELECT <attribute> AS <new Name> FROM <Relation>
  - Example:
    SELECT number AS roomNumber, building FROM rooms

rooms	
number	building
2	N1
3	N4

$\rho_{roomNumber/number}(rooms)$ →

$\rho_{number/roomNumber}(rooms)$

roomNumber	building
2	N1
3	N4

Aggregate Functions and Grouping

The SQL syntax for the grouping based on an attribute/column:
SELECT <attributes> FROM <Relation>
GROUP BY <attribute>

Example: SELECT firstName FROM persons GROUP BY firstName

firstName	lastName	age	male
Max	Bügler	33	1
Max	Mustermann	20	1
Max	Müller	25	1
Parker	James	28	0
Parker	Miller	24	1
Jennifer	Milan	22	0
Jennifer	Turner	28	0
John	Turner	25	1
Jennifer	Turner	21	0

→

firstName
Jennifer
John
Max
Parker

The SQL syntax for aggregating group attributes:
SELECT <groupAttribute>, <function on aggregated Attributes>
FROM <Relation> GROUP BY <groupAttribute>

Example: SELECT firstName, count(lastName) AS counts
FROM persons GROUP BY firstName

firstName	lastName	age	male
Max	Bügler	33	1
Max	Mustermann	20	1
Max	Müller	25	1
Parker	James	28	0
Parker	Miller	24	1
Jennifer	Milan	22	0
Jennifer	Turner	28	0
John	Turner	25	1
Jennifer	Turner	21	0

→

firstName	counts
Jennifer	3
John	1
Max	3
Parker	2

- Different aggregation functions
- Can be combined by OR, AND, NOT → CONCAT("EngDB", "CMS") → "EngDBCMS"

AVG()	Average Value
COUNT()	Number of (DISTINCT) rows
GROUP_CONCAT()	Concatenate Strings
MAX(), MIN()	Max/min Values
STDDEV()	Standard Deviation of values
SUM()	Sum of values
BIT_AND(), BIT_OR()	Bitwise AND/OR
…	

```
SELECT
  firstName,
  GROUP_CONCAT(lastName) AS lastNames,
  GROUP_CONCAT(DISTINCT lastName) AS distinctLastNames,
  COUNT(lastName) AS count,
  COUNT(DISTINCT lastname) AS distinctCount,
  AVG(age) AS averageAge,
  STDDEV(age) AS ageStDev,
  BIT_OR(male) AS containsMales,
  NOT BIT_AND(male) AS containsFemales,
  BIT_OR(male) AND NOT BIT_AND(male) AS unisexName
FROM Persons
GROUP BY firstName;
```

Parker
Jennifer
Jennifer
John
Jennifer

+ Seçenekler

firstName	lastNames	distinctLastNames	count	distinctCount	averageAge	ageStDev	containsMales	containsFemales	unisexName
Jennifer	Milan,Turner,Turner	Milan,Turner	3	2	23.6667	3.091206165165235	0	1	0
John	Turner	Turner	1	1	25.0000	0	1	0	0
Max	Bügler,Mustermann,Müller	Bügler,Müller,Mustermann	3	3	26.0000	5.354126134736337	1	0	0
Parker	James,Miller	James,Miller	2	2	26.0000	2	1	1	1

HAVING

The SQL syntax for aggregating group attributes:
SELECT <groupAttribute>
FROM <Relation>
GROUP BY <groupAttribute>
HAVING <condition>

Example:
SELECT firstName FROM Persons GROUP BY firstName HAVING COUNT(lastName)>1;

firstName
Jennifer
Max
Parker

SELECT firstName, GROUP_CONCAT(Age) AS ages
FROM Persons GROUP BY firstName
HAVING SUM(Age)>60

firstName	ages
Jennifer	22,28,21
Max	33,20,25

Nested Queries and Quantors

Example:

Jennifer	Milan	22	0
Jennifer	Turner	28	0
John	Turner	25	1
Jennifer	Turner	21	0

SELECT * FROM Persons **WHERE** age>30

SELECT * FROM Persons **WHERE** age > (SELECT AVG(age) FROM Persons);

Returns a single value

firstName	lastName	age	male
Max	Bügler	33	1
Parker	James	28	0
Jennifer	Turner	28	0

Nested Queries and Quantors-EXISTS

VIPs

lastName
Turner
Milan

firstName	lastName	age	male
Max	Bügler	33	1
Max	Mustermann	20	1
Max	Müller	25	1
Parker	James	28	0
Parker	Miller	24	1
Jennifer	Milan	22	0
Jennifer	Turner	28	0
John	Turner	25	1
Jennifer	Turner	21	0

Nested Queries and Quantors - EXISTS

- List entries for which another query returns something
- **EXISTS** quantor
- Example:

SELECT * FROM Persons WHERE **EXISTS**
(SELECT * FROM VIPs WHERE VIPs.lastName=Persons.lastName);

firstName	lastName	age	male
Jennifer	Milan	22	0
Jennifer	Turner	28	0
John	Turner	25	1
Jennifer	Turner	21	0

SELECT * FROM Persons WHERE **NOT EXISTS**
(SELECT * FROM VIPs WHERE VIPs.lastName=Persons.lastName);

firstName	lastName	age	male
Max	Bügler	33	1
Max	Mustermann	20	1
Max	Müller	25	1
Parker	James	28	0
Parker	Miller	24	1

VIPs

lastName
Turner
Milan

firstName	lastName	age	male
Max	Bügler	33	1
Max	Mustermann	20	1
Max	Müller	25	1
Parker	James	28	0
Parker	Miller	24	1
Jennifer	Milan	22	0
Jennifer	Turner	28	0
John	Turner	25	1
Jennifer	Turner	21	0

Nested Queries and Quantors - EXISTS

- Define whether another query returns something as a boolean column using the EXISTS quantor.
- Example:

SELECT firstName, lastName,
(EXISTS
(SELECT * FROM VIPs WHERE VIPs.lastName=Persons.lastName))
AS isVIP
FROM Persons;

firstName	lastName	isVIP
Max	Bügler	0
Max	Mustermann	0
Max	Müller	0
Parker	James	0
Parker	Miller	0
Jennifer	Milan	1
Jennifer	Turner	1
John	Turner	1
Jennifer	Turner	1

Special Language Elements

SELECT * FROM Persons WHERE age BETWEEN 20 AND 25

firstName	lastName	age	male
Max	Mustermann	20	1
Max	Müller	25	1
Parker	Miller	24	1
Jennifer	Milan	22	0
John	Turner	25	1
Jennifer	Turner	21	0

SELECT * FROM Persons WHERE age IN (11,22,33);

firstName	lastName	age	male
Max	Bügler	33	1
Jennifer	Milan	22	0

SELECT * FROM Persons WHERE lastName LIKE "m%";

firstName	lastName	age	male
Max	Mustermann	20	1
Max	Müller	25	1
Parker	Miller	24	1
Jennifer	Milan	22	0

SELECT * FROM Persons WHERE lastName LIKE "%m%";

firstName	lastName	age	male
Max	Mustermann	20	1
Max	Müller	25	1
Parker	James	28	0
Parker	Miller	24	1
Jennifer	Milan	22	0

SELECT * FROM Persons WHERE lastName LIKE "_____";

firstName	lastName	age	male
Parker	James	28	0

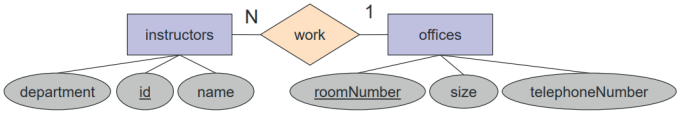
SELECT * FROM Persons WHERE lastName LIKE "M_ller";

firstName	lastName	age	male
Max	Müller	25	1
Parker	Miller	24	1

Foreign key integrity constraint

Foreign key integrity constraint

- Example:



instructors			
id	name	department	room
1	Bob	CMS	10
2	Tom	CIE	5
3	Alice	CMS	8

offices		
roomNumber	size	telephoneNumber
10	10	22678
8	25	35131

CREATE TABLE <name of table> (
[other attribute definitions],
<name of foreign key> <type of foreign key> <integrity constraints>,
[other attributes that are part of the foreign key, comma separated]
FOREIGN KEY (<local foreign key attributes, comma separated>)
REFERENCES <other table>
(<other table primary key attributes, comma separated>));

telephonNumber : varchar(40) #room : i

CREATE TABLE offices (
roomNumber INTEGER PRIMARY KEY,
size INTEGER NOT NULL,
telephonNumber VARCHAR(40) NOT NULL);

CREATE TABLE instructors (
id INTEGER PRIMARY KEY,
name VARCHAR(50) NOT NULL,
department VARCHAR(20),
room INTEGER,
FOREIGN KEY (room) **REFERENCES** offices (roomNumber));

ALTER TABLE <name of table> ADD
FOREIGN KEY (<local foreign key attributes, comma separated>)
REFERENCES <other table>
(<other table primary key attributes, comma separated>);

The syntax for **update and delete** actions in the foreign key definition
FOREIGN KEY (<local foreign>)
REFERENCES <other table> (<other table primary key attributes>)
[ON DELETE [RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT]]
[ON UPDATE [RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT]]

- Example:
  foreign key (room) references offices (roomNumber) on delete cascade
- Restrict is default, means not allowed
- Cascade means that the other table's updates are propagated
- Set Null will set the value to null (if no NOT NULL constraint is given)
- No action will change nothing
- Set default will set the value to a constraint defined default value

- The foreign key ensures data integrity

instructors			
id	name	department	room
1	Bob	CMS	5
2	Tom	CIE	5
3	Alice	CMS	8

offices		
roomNumber	size	telephoneNumber
5	10	22678
8	25	35131

- Example for delete from offices where roomNumber=5
  - Restrict: error
  - Cascade: Bob und Tom are deleted!
  - Set null: room in Bob and Tom are null (if no NOT NULL constrain is given)
  - No action: the room in Bob and Tom is 5
  - Set default: the room in Bob and Tom is set to a predefined default value