Relational Databases and SQL

Introduction

RDBMS & SQL

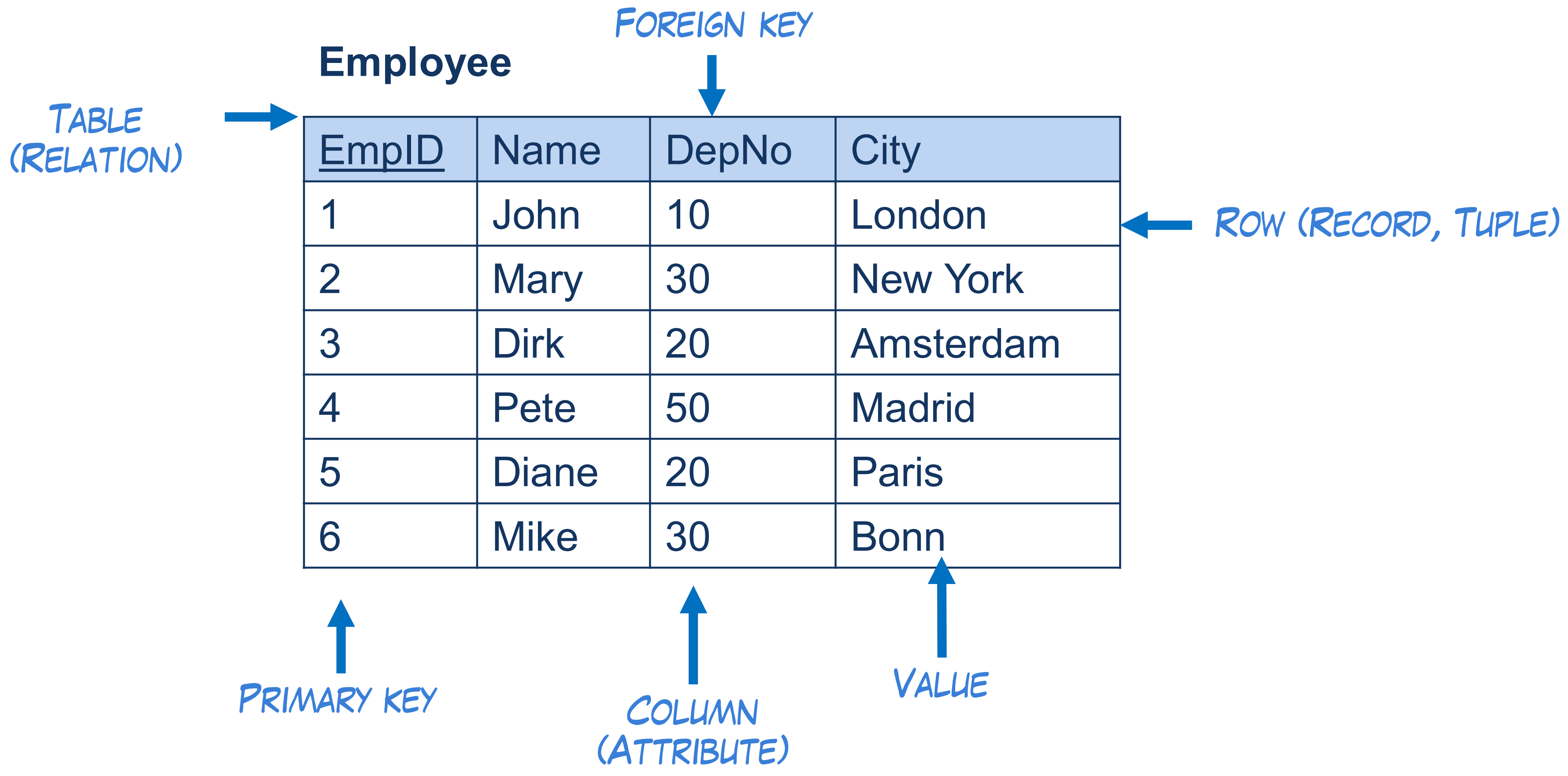
Relational database

- A relational database is a database that is perceived as a collection of relations and nothing but relations.

EmpID	Name	DepNo	City
1	John	10	London
2	Mary	30	New York
3	Dirk	20	Amsterdam
4	Pete	50	Madrid
5	Diane	20	Paris
6	Mike	30	Bonn

DepNo	Name
10	Sales
20	Marketing
30	Management
40	Office
50	R&D

The table



Employee

EmpID	Name	DepNo	City
1	John	10	London
2	Mary	30	New York
3	Dirk	20	Amsterdam
4	Pete	50	Madrid
5	Diane	20	Paris
6	Mike	30	Bonn

RDBMS

- **R**elational **D**ata**B**ase **M**anagement **S**ystem
  - Application on a network (host/port)
  - Manages databases
  - Executes queries
  - Guarantees relational integrity
  - Manages transactions
  - Security
  - Authentication, authorization
  - Backup/restore
  - …

RDBMS examples

- Berkeley DB
- Caché
- Clipper
- **DB2**
- IDMS
- IDS (hiërarchisch)
- dBase
- FileMaker
- Firebird
- FoxPro
- Informix
- **MariaDB**
- msSQL
- Microsoft Access
- **Microsoft SQL Server**
- **MySQL**
- **Oracle Database**
- Paradox
- **PostgreSQL**
- SmallSQL
- SQLite
- Sybase
- Turboveg

Creating and modifying tables

CREATE, ALTER TABLE

Creating and modifying tables

- **Introduction to SQL**
  - history
  - language elements : DDL - DML - DCL
  - standard : ANSI SQL
  - extensions
- Creating and modifying tables

History

- Structured Query **Language**
  - So, a ***language***
  - Functional
- First publication in 1974
- Originates from SEQUEL

Language elements

DDL Data Definition Language	**DML** Data Manipulation Language	**DCL** Data Control Language
• create objects • modify objects • delete objects	• query data • insert data • modify data • delete data	• assign authorizations

Standards

- ANSI SQL (1986)
  - based on original IBM SQL (SEQUEL) - an existing implementation
  - represents a basic minimum
- ANSI SQL-89, SQL-92
  - results from the commercialisation of SQL
  - enhanced integrity rules
  - new data types
  - more manipulation possibilities
- ANSI SQL 1999, 2003 ... 2016
  - added functionalities like XML, JSON, temporal tables, INSTEAD OF triggers, Window functions, identity columns

SQL Extensions

- Extensions on the SQL standard language, offering extra programming features like conditional logic

Sybase &
Microsoft SQL Server
- Transact-SQL (T-SQL)

Oracle
- PL/SQL

IBM
- DB2 UDB SQL

Creating and modifying tables

- Introduction to SQL
  - history
  - language elements : DDL - DML - DCL
  - standard : ANSI SQL
  - extensions
- **Creating and modifying tables**

Creating and modifying tables

- Creating a table
  - `CREATE TABLE` command
- Deleting a table
  - `DROP TABLE` command
- Modifying a table structure
  - `ALTER TABLE` command

Creating table

- Name the table
- Name each column
- Specify the data type of each column
- Specify the null status for each column (optional, but best practise)

```
CREATE TABLE invoice
(
      invoicenr     int           NOT NULL,
      clientnr      int           NOT NULL,
      invoicedate   datetime2     NOT NULL,
      invoiceamount decimal(5,2)  NULL
)
```

AVAILABLE DATA TYPES
DIFFER PER DBMS

Deleting table

- Provide table name

```
DROP TABLE invoice
```

Modifying table structure

- Add column(s)

```
ALTER TABLE invoice
  ADD customnr  int   NOT NULL
```

```
ALTER TABLE invoice
  ADD customnr  int       NOT NULL,
      status    char(1)   NULL
```

- Modify column(s)

```
ALTER TABLE invoice
  ALTER COLUMN status char(3) NULL
```

- Drop column(s)

```
ALTER TABLE invoice
  DROP COLUMN status
```

Assignments

- 3.1
- 3.2
- 3.3

Reading data from a table

SELECT

SELECT statement

(5) SELECT *select list*

(1) FROM *table name*

(2) WHERE *predicate*

(3) GROUP BY *grouping element list*

(4) HAVING *predicate*

(6) ORDER BY sort_key

Basic form of a query

SELECT form

- format of the result set returned by the query

FROM source

- which table contains the data source

WHERE (or HAVING) condition

- conditions that a row must meet to qualify for selection

```sql
SELECT name, age, salary
FROM Employees
WHERE age < 50
```

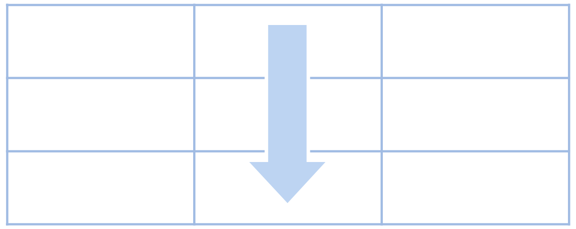
Selecting an entire table

```
SELECT *
FROM Publishers
```

pub_id	pub_name	city
1389	Algodata Info	Berlin
736	New Moon Books	Madrid
…	…	…

Select list

- SELECT clause specifies the columns that appear in the result set
- The result forms a vertical subset of the table(s)



- Example:

```
SELECT  au_fname
       ,au_lname
       ,city
FROM Authors
```

ALL vs DISTINCT

```
CA
CA
CA
KS
KS
CA
MI
NY
CA
TN
CA
OR
OR
```

```sql
SELECT ALL state
FROM Authors;
```

```sql
SELECT DISTINCT state
FROM Authors;
```

```
CA
KS
MI
NY
TN
OR
```

Reading data from a table

- Basic form of a query
- Select list
- $\boxed{\textbf{Sorting}}$
- Conditions for selecting data
- Functions and calculations
- Grouping data

ORDER BY

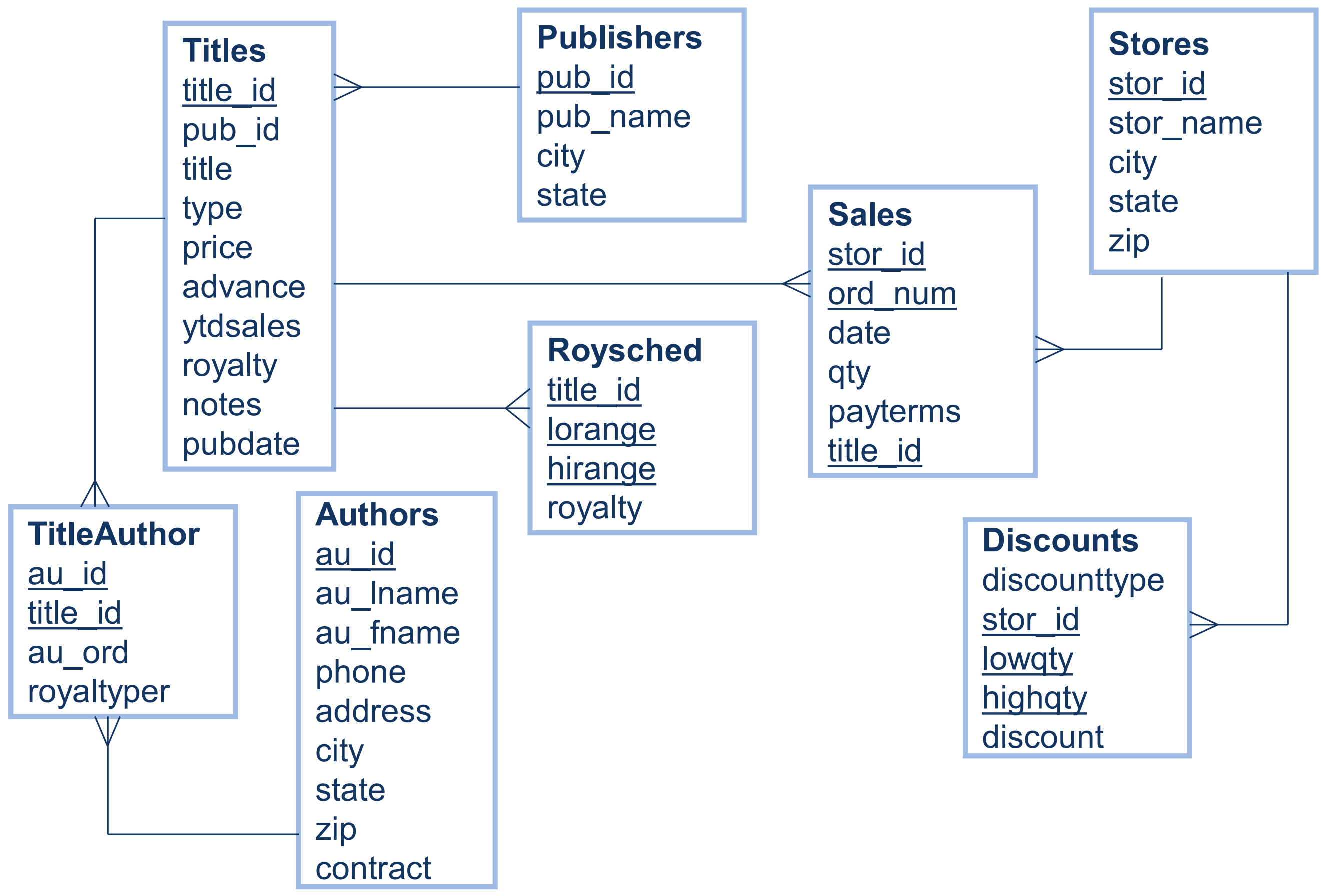
```
SELECT  title
        ,price
FROM    Titles
ORDER BY price
```

```
SELECT  city AS City
        ,au_fname AS [First Name]
FROM    Authors
ORDER BY city, au_fname DESC
```

title	price
The Psychology of Computer Cooking	NULL
Net Etiquette	NULL
The Gourmet Microwave	2.99
You Can Combat Computer Stress!	2.99
Life Without Fear	7
...	...

City	First Name
Ann Arbor	Innes
Berkeley	Cheryl
Berkeley	Abraham
...	...

Pubs: Data Structure Diagram



Demo

- SELECT assignment 1

```
SELECT *
FROM Publishers
```

- Specifying column names offers greater control over the result set of the query (order of columns, headers, etc.)

```
SELECT pub_id, pub_name, city, state
FROM Publishers
```

Demo

- SELECT assignment 2

a.

```
SELECT au_fname, au_lname
FROM Authors
```

b. Using the order by clause

```
SELECT au_fname, au_lname
FROM Authors
ORDER BY au_lname
```

c. Using aliases

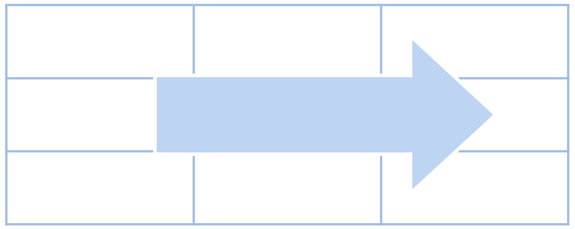
```
SELECT  au_fname AS "First Name"
       ,au_lname AS [Last Name]
FROM Authors
ORDER BY au_lname
```

Reading data from a table

- Basic form of a query
- Select list
- Sorting
- **Conditions for selecting data**
- Functions and calculations
- Grouping data

WHERE clause

- `WHERE` clause specifies condition(s) for a row to be selected
- Result forms a horizontal subset



```
SELECT title, type, price
FROM Titles
WHERE type = 'business'
```

Operators

- Comparison operators
- BETWEEN operator
- IN operator
- LIKE operator
- IS NULL operator

The comparison operators

=	equal to
>	bigger than
<	smaller than
>=	bigger than or equal to
<=	smaller than or equal to
<> or !=	not equal to

```
SELECT title, type, price
FROM   Titles
WHERE  price >= 10.00
```

```
WHERE  ordernr = 1342
WHERE  orderdate > '1999-01-01'
WHERE  invoiceamount < 1000
WHERE  ordernr <> 1592
```

Condition coupling

- `WHERE` clause can express several conditions
- Conditions are combined with the AND or OR operator
- Combinations are grouped with parenthesis

```
SELECT title, type, price
FROM   Titles
WHERE  price >= 10
AND    price <= 100
AND    (type = 'business' OR type = 'psychology')
```

BETWEEN operator

- `BETWEEN` operator specifies a range
- Upper and lower limits are part of the range
- Can be combined with the NOT operator

```
WHERE amount >= 100 AND amount <= 1000
WHERE amount BETWEEN 100 AND 1000
```

```
WHERE amount < 100 AND amount >1000
WHERE amount NOT BETWEEN 100 AND 1000
```

IN operator

- `IN` operator:
  - replacement for multiple OR operators checking one value
  - specifies a list of constant values

```
SELECT pub_name, city
FROM   Publishers
WHERE  city = 'Boston' OR city = 'Paris' OR city = Chicago')
```

```
SELECT pub_name, city
FROM   Publishers
WHERE  city IN ('Boston', 'Paris', 'Chicago')
```

IN operator

- `IN` operator:
  - can be combined with the NOT operator

```
SELECT title, type
FROM   Titles
WHERE  type NOT IN ('business', 'psychology')
```

LIKE operator

- `LIKE` operator:
  - uses a constant in the form of a pattern or mask
  - can only be applied to character (string) columns
- Use of wildcards is essential

- All titles starting with an 'A':

```
SELECT title
FROM   Titles
WHERE  title LIKE 'A%'
```

LIKE operator

- Wildcards:

```
%           any string of 0 or more characters
_           (underscore) single arbitrary character
[]          arbitrary character inside a given range
[^]         arbitrary character outside a given range
```

`WHERE title LIKE '%5'`	any title ending with a 5
`WHERE title LIKE '__n%'`	any title of which the 3rd character is an n
`WHERE title LIKE '%5%'`	any title containing a 5
`WHERE title LIKE '%[adg]'`	any title ending with a, d or g
`WHERE title LIKE '[035]%'`	any title starting with a 0, 3 or 5
`WHERE title LIKE '[0-9]%'`	any title starting with a number
`WHERE title LIKE '[^a-cf]%'`	any title starting NOT starting with an a, b, c or f

The IS NULL Operator

- = NULL not allowed!
- `IS NULL` operator specifies NULL values
- Can be combined with NOT (`IS NOT NULL`)
- All publishers from outside the US:

```
SELECT pub_name, country
FROM   Publishers
WHERE  state IS NULL
```

Assignments

- 4.3.2
- 4.5.2

Reading data from a table

- Basic form of a query
- Select list
- Sorting
- Conditions for selecting data
- **Functions and calculations**
- Grouping data

Functions and calculations

Calculations in SELECT

Aggregate Functions

Scalar Functions

Calculations in SELECT

- Calculating with constants

```sql
SELECT title, price * 2 AS [Double price]
FROM   Titles
```

- Calculating between mutual columns

```sql
SELECT title, price * ytd_sales AS [Total sales amount]
FROM   Titles
```

- Any mathematical operator can be used

Aggregate functions

- Aggregate functions summarize multiple rows
  - For example: the total amount of sold books

- Common aggregate functions

```
COUNT
MIN
MAX
SUM
AVG
```

Aggregate functions

- COUNT function
  - `SELECT COUNT(*) FROM Titles`
    - counts all the rows of the Titles table
    - `COUNT(*)` = special case - other aggregate functions don't have * option
  - `COUNT(city)`
    - counts all the rows where city has got a value (NULL values are skipped)
  - `COUNT(DISTINCT city)`
    - counts all the rows where city has got a value, duplicates are not counted

Aggregate functions

- `MIN(column)`: smallest value for given column
- `MAX(column)`: highest value for given column
  - MIN and MAX are also applicable to character columns!
- `SUM(column)`: sum of the values for a given column
- `AVG(column)`: average of the values for a given column
  - Careful: only takes NOT NULL values into account!

Aggregate functions

```
SELECT  COUNT(*)
FROM    Titles
```

→ 18

```
SELECT  COUNT(DISTINCT type)
FROM    Titles
```

→ 6

```
SELECT  SUM(ytd_sales)
FROM    Titles
WHERE   type = 'business'
```

→ 30788

Scalar functions

- Operation on one or more values from *one row*

- Common scalar function types:
  - Conversion functions
  - String function
  - Date/Time functions

- Scalar functions can be used anywhere
  - But: known for poor performance!

Conversion functions

- Combine or compare different data types
  - Example: add a number to a string
- Some conversion is automatic (implicit conversion)
  - Works only in one direction, for example text to number (not vice versa)
- When there is no automatic conversion, use explicit conversion

```
SELECT 'The price is ' + CAST(price AS varchar(12)) + ' dollars'
FROM   Titles
```

```
The price is 11.95 dollars
```

String functions

- Manipulate strings
  - Usually output another string

`SELECT CHARINDEX('t','exhibition')`	7
`SELECT SUBSTRING ('exhibition', 4, 4)`	'ibit'
`SELECT RTRIM('exhibition   ' )`	'exhibition'
`SELECT UPPER('exhibition')`	'EXHIBITION'
`SELECT REPLACE('exhibition','tion','t')`	'exhibit'

Date/Time functions

`SELECT SYSDATETIME()`	returns current date and time
`SELECT YEAR('2000-01-01')`	2000
`SELECT DATEADD(day,20,'1999-12-31')`	2000-01-19 00:00:00.000
`SELECT EOMONTH('2012-02-12')`	2012-02-29
`SELECT DATEFROMPARTS(2015,3,30)`	2015-03-30
`SELECT DATENAME(weekday,'1918-10-12')`	Saturday

Reading data from a table

- Basic form of a query
- The select list
- Sorting
- Conditions for selecting data
- Functions and calculations
- **Grouping data**

Group By

- Groups results based on one or more columns
- Often combined with aggregate functions like `AVG` or `SUM`

A	10
B	15
B	25
A	20
C	30
B	5
C	50

A	15
B	15
C	40

Group By

```
SELECT  ordernr AS nr,
        price
FROM    orders
ORDER BY ordernr
```

nr	price
1	15
1	15
2	3
3	7
3	8

```
SELECT  ordernr AS nr,
        SUM(price) AS total
FROM    orders
GROUP BY ordernr
ORDER BY ordernr
```

nr	total
1	30
2	3
3	15

Having

- `HAVING` specifies conditions on the groups included in the result set
- May contain aggregate functions (contrary to the WHERE clause)

```
SELECT ordernr      AS nr,
       SUM(price)   AS total
FROM    orders
GROUP BY ordernr
  HAVING SUM(price) > 3
ORDER BY ordernr
```

RESULT

nr	total
1	30
3	15

WITHOUT HAVING

nr	total
1	30
2	3
3	15

Review

- `SELECT` define columns
  - DISTINCT eliminate double rows
  - calculate, aggregate
- `FROM` specify table(s)
- `WHERE` define conditions (filter)
  - booleans: AND, OR, NOT
  - special operators: [NOT] IN, LIKE, BETWEEN
  - simple comparators: =, <, >, <>
  - special comparators: column IS [NOT] NULL
- `GROUP BY` group rows
- `HAVING` filter the group
- `ORDER BY` sort the result set

Assignment

- 4.5.3
- 4.6.1
- 4.6.4

Reading data from multiple tables

JOIN

Using the join keyword

- Different types of joins:
  - `CROSS JOIN`
  - `INNER JOIN`
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`

Cross join

- **`CROSS JOIN`** logical operator
  - joins each row from the 1st (top) input with each row from the 2nd (bottom) input
  - result is called a *'Cartesian Product'*
  - used for generating test data (rarely used)

Car

Peugeot	306
Peugeot	206
Renault	Scenic

Colour

red
blue

```
SELECT *
FROM Car
CROSS JOIN Colour
```

Result

Peugeot	306	red
Peugeot	306	blue
Peugeot	206	red
Peugeot	206	blue
Renault	Scenic	red
Renault	Scenic	blue

Inner join

- **`INNER JOIN`**
  - Join of two or more tables that return only the rows that satisfy the join condition
  - Example: Overview of all distributors with the products they can deliver

```
SELECT
    d.distributor_nr
    , d.distributor_name
    , p.product_nr
    , p.product_name
FROM          Distributor AS d  ← TABLE ALIAS
  INNER JOIN  Product AS p
    ON        d.distributor_nr = p.distributor_nr  ← JOIN CONDITION
```

ON-clause and WHERE-clause

- Can both be used in one query
  - Use the ON-clause to specify a join-condition
  - Use the WHERE-clause to specify other conditions

- All distributors localized in the USA and their products

```
SELECT
  d.distributor_name
  , p.product_name
FROM Distributor       AS d
  INNER JOIN Product   AS p
    ON                 d.distributor_nr = p.distributor_nr
WHERE                  d.country = 'USA'
```

Preferred Solution

Outer join

- Extension of the Inner join
- Also include rows that would not appear in the result set because their join condition evaluates to false
- Listed with a NULL value in the result set

Outer join

- Two tables:

TableL

l1	l2
A	1
B	2
C	3

TableR

r1	r2
B	10
C	20
D	30

```
SELECT l.l1, l.l2, r.r1, r.r2
FROM TableL AS l
LEFT OUTER JOIN TableR AS r
  ON l.l1 = r.r1
```

- Result:

`LEFT OUTER JOIN`

l1	l2	r1	r2
A	1		
B	2	B	10
C	3	C	20

`RIGHT OUTER JOIN`

l1	l2	r1	r2
B	2	B	10
C	3	C	20
		D	30

`FULL OUTER JOIN`

l1	l2	r1	r2
A	1		
B	2	B	10
C	3	C	20
		D	30

Outer join

- Example: All distributors from Amsterdam with the products that they can deliver PLUS distributors without products at all

```
SELECT  d.distributor_name, p.product_name
FROM Distributor AS d
LEFT OUTER JOIN Product AS p
  ON d.distributor_nr = p.distributor_nr
WHERE d.city = 'Amsterdam'
```

Assignment

- 5.1.3
- 5.2.2

Reading data from multiple tables

Subqueries

Subqueries

- Example: All distributors of cats

Always between parenthesis:

SUBQUERY

SUBQUERY

```sql
SELECT distributor_name
FROM Distributors
WHERE distributor_nr IN
(
        SELECT distributor_nr
        FROM Distributor
        WHERE product_nr IN
        (
                SELECT product_nr
                FROM Product
                WHERE product_name = 'cat'
        )
)
```

Rules and limitations

- `ORDER BY` clause in a subquery not allowed
- When using a simple comparison operator (=, <, >, <>, …) the subquery may only return *one value*

1. Self-Contained Subquery

- Example:
  - Show the information of the store where a highest quantity of a book was sold

```
SELECT stor_id, qty
FROM Sales
WHERE qty = (                  )
```

```
SELECT MAX(qty)
FROM Sales
```

1. Self-Contained Subquery

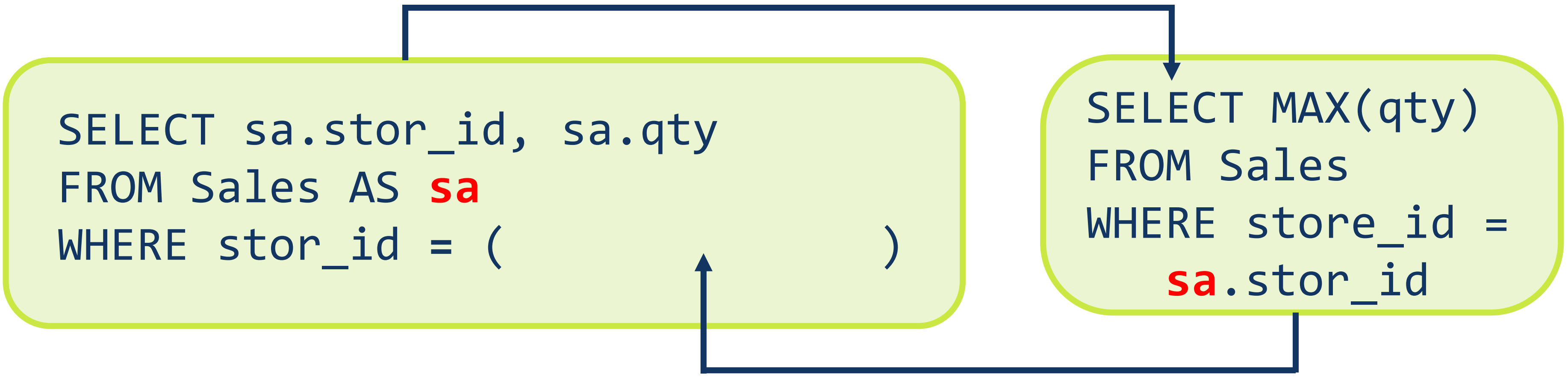
- Example:
  - Show the information of stores where the state is 'California'

```
SELECT stor_id, qty
FROM Sales
WHERE stor_id IN (                   )
```

```
SELECT stor_id
FROM Stores
WHERE state = 'CA'
```

2. Correlated Subquery

- Subquery references a column in the outer statement.
  Inner query is executed for each candidate row in the outer statement.
- Example:
  - For each store the highest quantity of any product sold



```
SELECT sa.stor_id, sa.qty
FROM Sales AS sa
WHERE stor_id = (                    )
```

```
SELECT MAX(qty)
FROM Sales
WHERE store_id =
   sa.stor_id
```

ANY operator

- ANY returns TRUE when:
  - the comparison specified is TRUE for ANY subquery-row
  - Example:
    All books with an advance higher than ANY book of 'Algodata Infosystems'

```
SELECT title
FROM Titles
WHERE advance > ANY
(
    SELECT t.advance
    FROM Titles AS t
    INNER JOIN Publishers AS p
    ON t.pub_id = p.pub_id
    WHERE p.pub_name = 'Algodata Infosystems'
)
```

ALL operator

- ALL returns TRUE when:
  - the comparison specified is TRUE for ALL subquery-rows OR
  - the subquery returns an empty result set

  - Example:
    All books with an advance less than ALL books of 'New Moon Books'

```
SELECT title
FROM Titles
WHERE advance < ALL
(
    SELECT t.advance
    FROM Titles AS t
    INNER JOIN Publishers AS p
    ON t.pub_id = p.pub_id
    WHERE p.pub_name = 'New Moon Books'
)
```

Exists operator

- Is FALSE when the subquery returns an empty result set, else TRUE
- Example:
  - All stores with sales in 1994

JOIN OPERATION

```
SELECT DISTINCT st.stor_name
FROM Sales AS sa
INNER JOIN Stores AS st
ON sa.stor_id = st.stor_id
WHERE YEAR(sa.ord_date) = 1994
```

SUBQUERY AND EXISTS

```
SELECT st. stor_name
FROM Stores AS st
WHERE EXISTS
(
    SELECT 1
    FROM Sales AS sa
    WHERE sa.stor_id = st.stor_id
    AND YEAR(sa.ord_date) = 1994
)
```

Which one is faster?

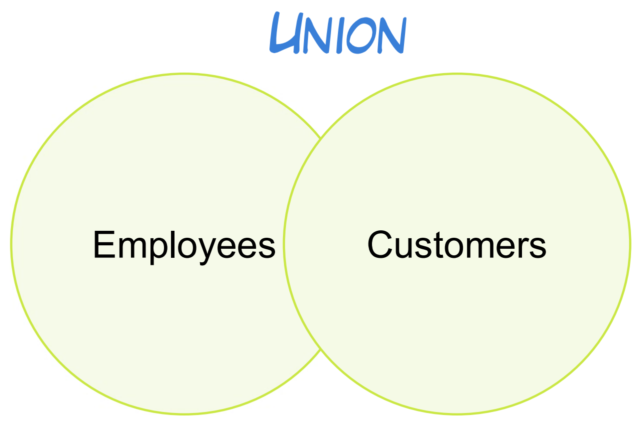
Assignment

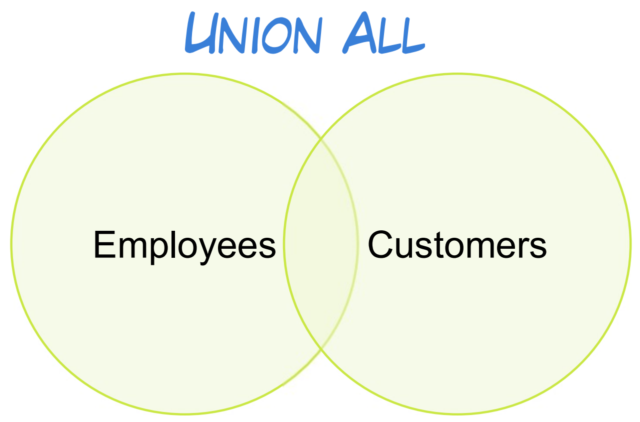
- 6.1.2
- 6.1.3
- 6.2.1

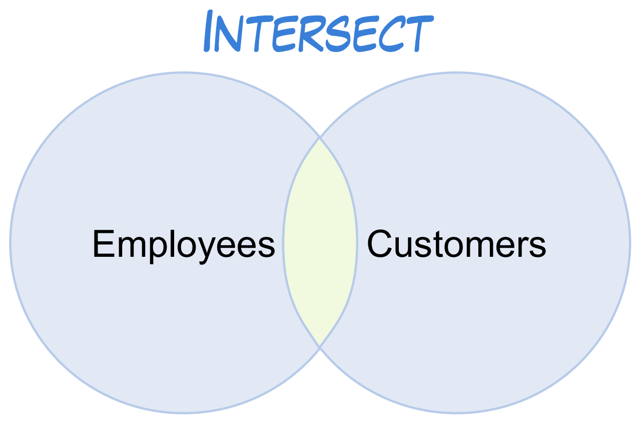
Reading data from multiple tables

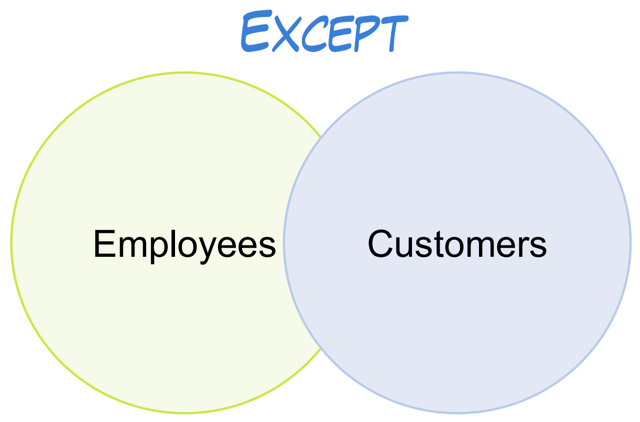
Combining queries

All Set operators

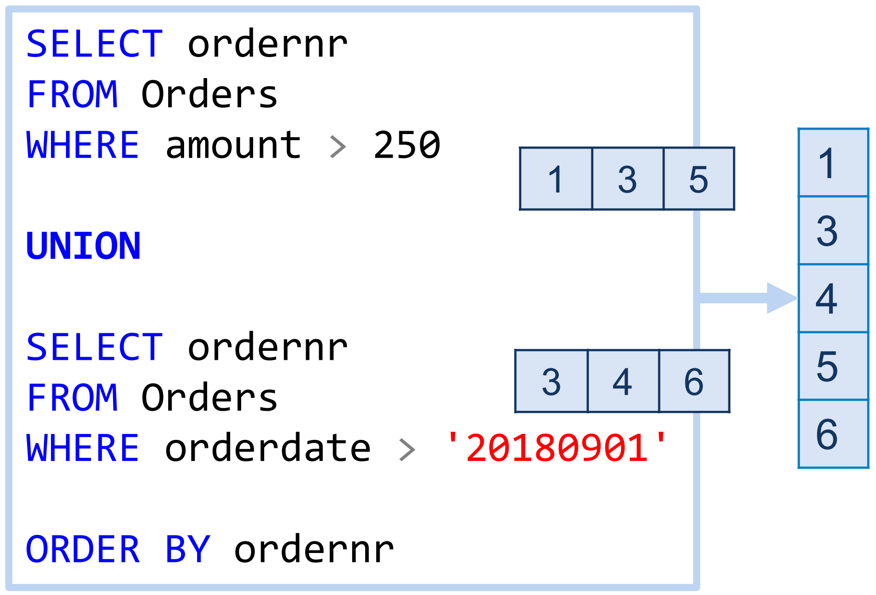








Union and Union All

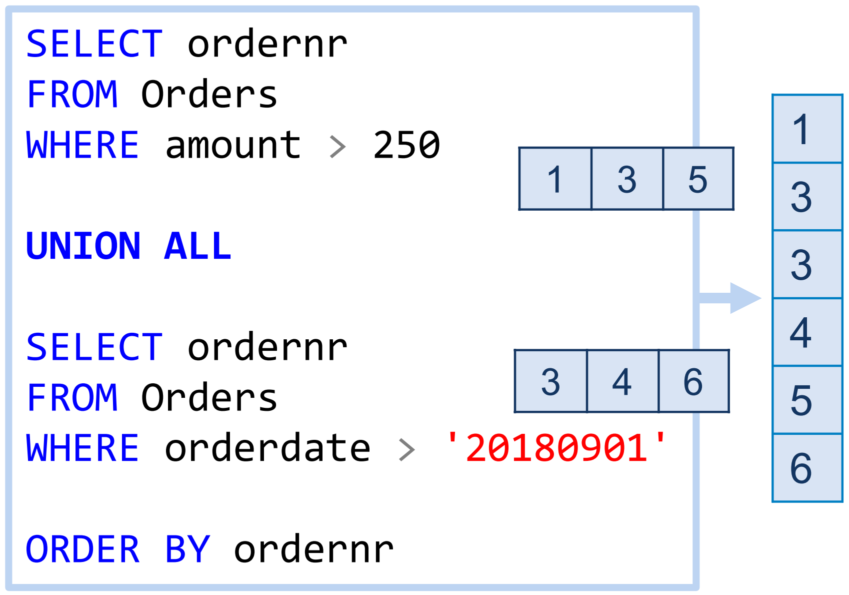


```
SELECT ordernr
FROM Orders
WHERE amount > 250

UNION

SELECT ordernr
FROM Orders
WHERE orderdate > '20180901'

ORDER BY ordernr
```



```
SELECT ordernr
FROM Orders
WHERE amount > 250

UNION ALL

SELECT ordernr
FROM Orders
WHERE orderdate > '20180901'

ORDER BY ordernr
```

INTERSECT and EXCEPT

- `INTERSECT`
  - distinct set of rows which exists in both resultsets

```sql
SELECT au_lname, city
FROM Authors
INTERSECT
SELECT pub_name, city
FROM Publishers
```

- `EXCEPT`
  - distinct set of rows only available in the *left* resultset
  - ideal for calculating deltas

```sql
SELECT au_lname, city
FROM Authors
EXCEPT
SELECT pub_name, city
FROM Publishers
```

SELECT statement revisited

- SELECT *select list*
- FROM *table name* [AS alias]
- [<join type>] JOIN *table name* [AS alias]
- ON <join condition>
- WHERE <predicate>

Manipulating data

INSERT INTO, UPDATE, DELETE

INSERT INTO

- **insert-values** construction: you can state what values must be inserted in a new row
  - multiple rows in one statement is possible

```
INSERT INTO <table_name>
[(column_list)]
VALUES
(contant_expr[, constant_expr]...)
[,(...)]
```

- **insert-select** construction: you can use a SELECT query to create the new row(s)

```
INSERT INTO <table_name>
[(column_list)]
SELECT ...
```

INSERT INTO

- Examples:

```
INSERT INTO Product
(categoryID, productname, price)
VALUES
(27, 'liquorice', 3.00)
```

```
INSERT INTO Product
(categoryID, productname, price, location)
(27, 'liquorice', 3.00, 'London')
```

```
INSERT INTO Product
(categoryID, productname, price)
VALUES
(27, 'liquorice', 3.00),
(18, 'strawberries', 2.49),
(18, 'oranges', 3.99)
```

```
INSERT INTO Product
(categoryID, productname, price)
SELECT catID, productname, price
FROM catalogue
WHERE catDescription = 'flowers'
```

Update

- All rows from the specified table that meet the search condition will receive the specified value

```
UPDATE table_name
SET column_name = expression
WHERE search_condition
```

- Versions of the `UPDATE` command:
  - single row update
  - multiple row update
  - update with a subquery

Update

- Single row update:

```
UPDATE Product
SET price = price * 1.1
WHERE id = 233
```

- Multiple row update:

```
UPDATE Product
SET price = price * 1.1
```

```
UPDATE Product
SET price = price * 1.1
WHERE type = 'luxe'
```

Update

- Update with a subquery:

```
UPDATE Product AS p
SET p.amount = 0
WHERE p.status =
(
    SELECT c.status
    FROM Catalogue AS c
    WHERE p.product_nr = c.p_nr
    AND c.code = 1
)
```

```
UPDATE Product
SET colour =
(
    SELECT color
    FROM Catalogue
    WHERE p_nr = '001'
)
WHERE product_nr > '9000'
```

Delete

- Deletes one, several or all rows from a table

```
DELETE FROM table_name
[WHERE search_condition]
```

- Versions of the `DELETE` command:
  - single row delete
  - multiple row delete
  - delete with a subquery

Delete

- Single row delete:

```
DELETE FROM Product
WHERE id = 233
```

- Multiple row delete:

```
DELETE FROM Product
```

```
DELETE FROM Product
WHERE type = 'luxe'
```

- Delete with a subquery:

```
DELETE FROM Product AS p
WHERE status =
(
    SELECT status
    FROM Catalogue AS c
    WHERE p.product_nr = c.p_nr
    AND code = 1
)
```

Assignment

- 7.1.1
- 7.1.4
- 7.2.2
- 7.3.3