Europäisches Patentamt
European Patent Office
Office européen des brevets

SQL Self-Study Course

For MS SQL Server

- all examples executable in PATSTAT Online
- with exercises

Version 2.3



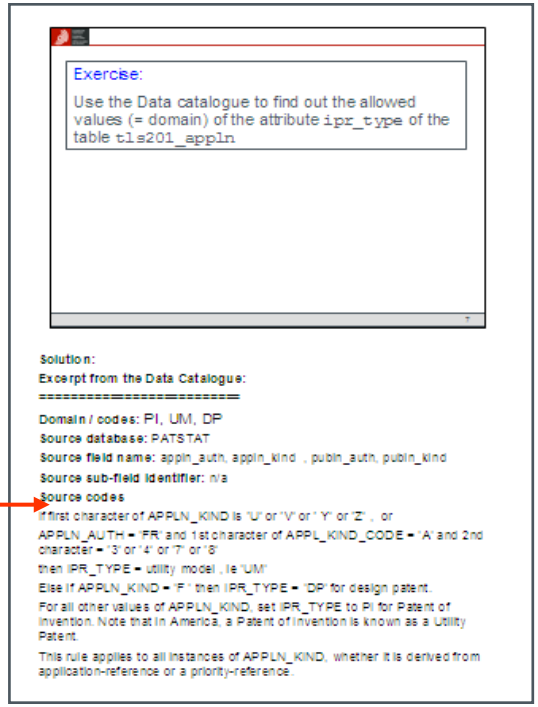
Purpose of this document

- This document should enable you to understand and to write your own SQL (MS SQL Server) queries in PATSTAT Online.

Tip:

Have a look at and also print the Notes View of each slide because it contains additional essential information and the solution to the exercises

Notes View



Additional Resources

You will find more information on PATSTAT data, the PATSTAT Online search application, etc. in the “Downloads" tab on EPO’s PATSTAT pages http://www.epo.org/searching-for-patents/business/patstat.html

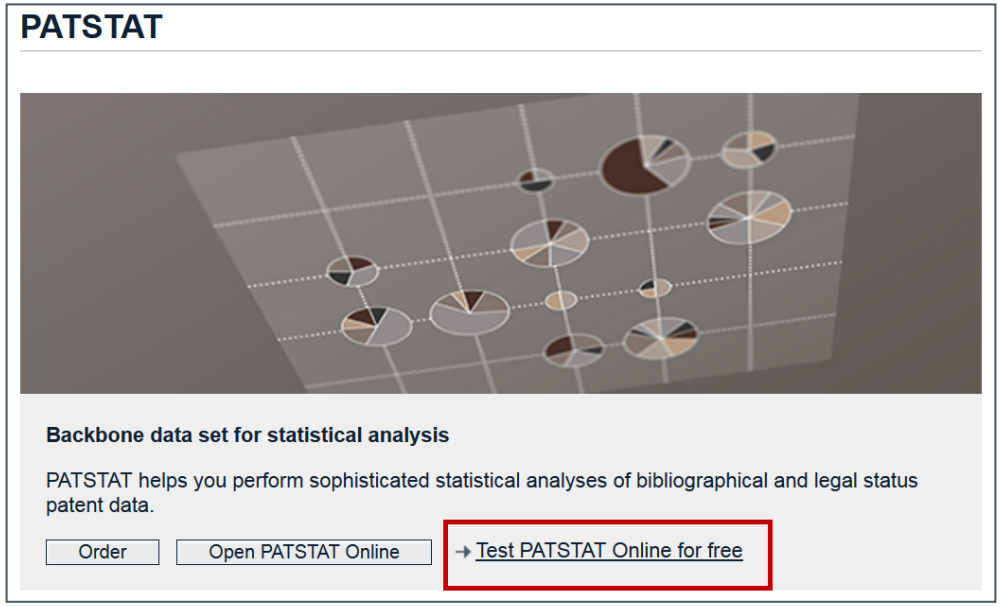
Getting started	Conditions	Downloads

Download	File type	Size
Presentations from the PATSTAT User Day in Tokyo in November 2014	ZIP	3 MB
An introduction to the PATSTAT database with example queries	PDF	292 KB
PATSTAT - Data Catalog 2015 Autumn Edition	PDF	2 MB
→ Sample set - contains bibliographical and legal status data for patent applications in the field of wind energy	Access 2010 format	49.4 MB
EP Register for PATSTAT - Data Catalog 2015 Autumn Edition	PDF	1.5 MB
PATSTAT data elements	PDF	220 KB
SQL self-study course	PDF	7.9 MB
PATSTAT Online user manual	PDF	2.8 MB
Sample queries and tips	PDF	204 KB

Prerequisites

To effectively follow this exercise and to do the proposed exercises, you need access to a PATSTAT database.

You can ask for a free trial for PATSTAT Online on the EPO homepage: http://www.epo.org/searching-for-patents/business/patstat.html



It will also help if you are ...

- familiar with the patent system in general (priorities, classification, families)
- experienced in doing boolean searches (AND, OR, NOT)

Content

Basics of relational databases

Data Storage

Pre-computer age data storage

Today: Relational Databases

- Ideal for structured data
- It's all in tables
- SQL query language

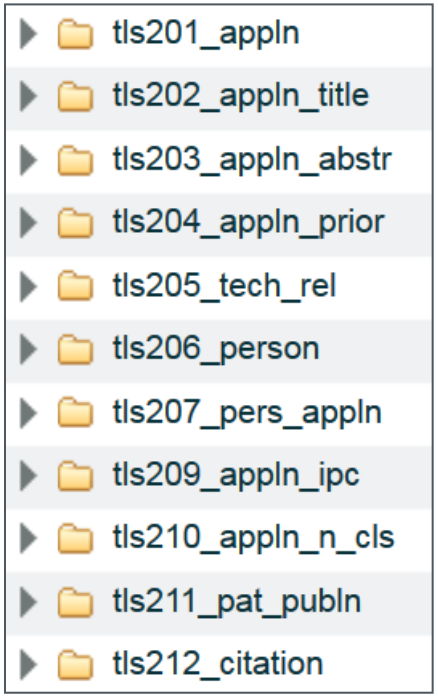
Tables, Rows, Columns

A table consists of **columns** and **rows** (also called **record**).

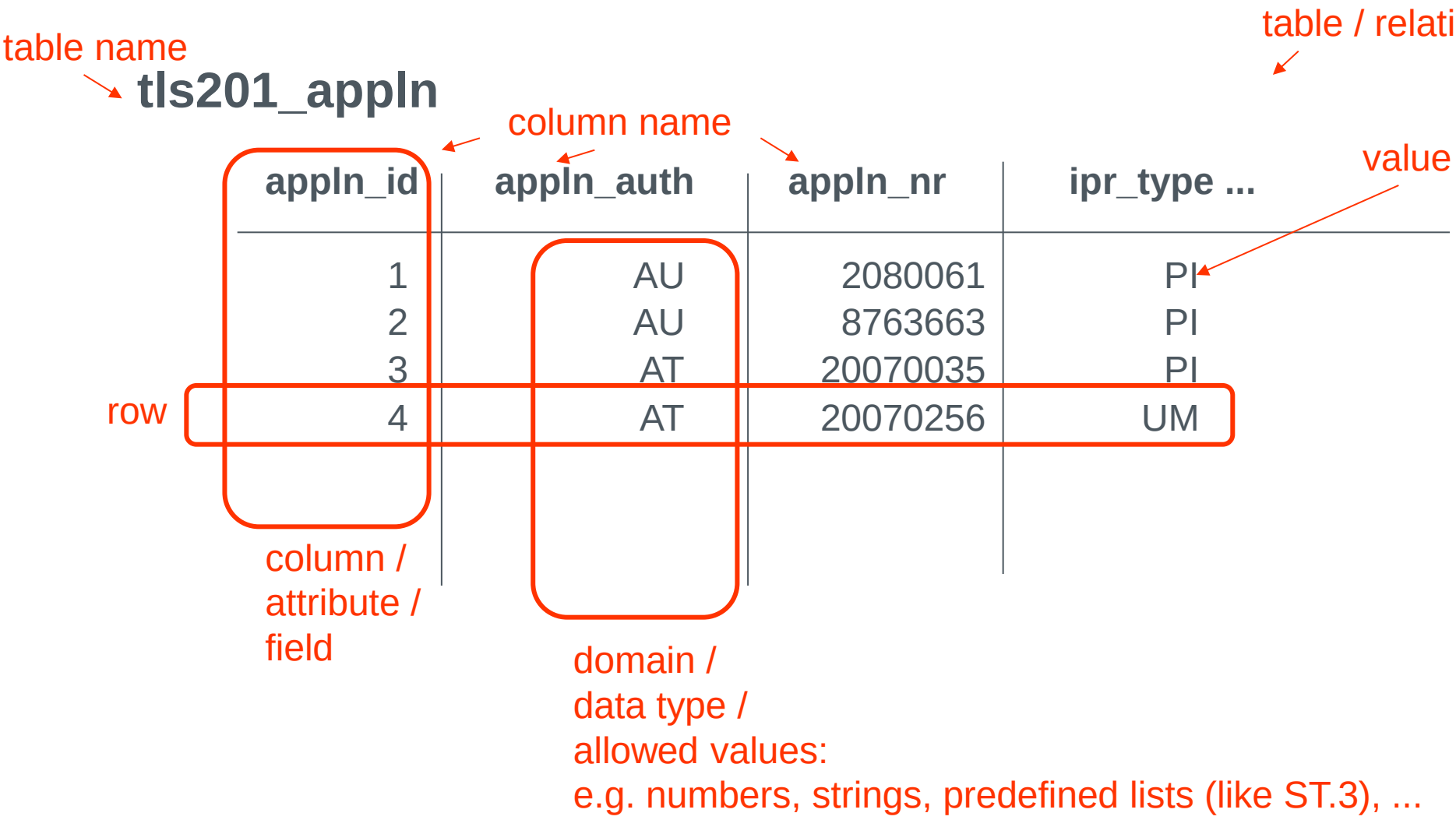
A cell contains 1 (!) data item of a certain **type**, which can be a string, a number or a date.

appln_id	appln_auth	appln_nr	appln_kind	appln_filing_da...	ipr_type
0				9999-12-31	
1	EP	00103094	A	2000-02-15	PI
2	EP	00107845	A	1992-12-02	PI
3	EP	00202556	A	2000-07-17	PI
4	EP	00300208	A	2000-01-13	PI
5	EP	00310305	A	2000-11-20	PI

A relational database is a collection of linked tables.



Relational Database concepts



A relational database

- ... is a collection of tables
- Each attribute of a row contains an elementary data element, e. g.
  - one IPC symbol
  - one application number
  - one abstract (regarded as one string, not as a sequence of words!)
- ... is well suited for structured data, but not for text or images (because e. g. text is usually treated as a complex data element, containing paragraphs, sentences, words, ...)

Exercise:

Use the Data Catalog to find out the allowed values (= the domain) of the attribute `ipr_type` of the table `tls201_appln`

A key uniquely identifies a row

- A key is an attribute (or a set of attributes) which can uniquely identify a single row of its own table
- Examples
  - `tls201_appln.appln_id`
    (short for: column `appln_id` in `tls201_appln`)
  - in table `tls204_appln_prior`:
    columns `appln_id` and `prior_appln_id` or
    columns `appln_id` and `prior_appln_seq_nr`
- One key (the shortest one) is the PRIMARY KEY;
  other keys may exist: ALTERNATIVE KEYs
- By convention, keys are usually the leftmost columns of a table

Exercise:

Use the Data catalogue to find the keys of table tls211_pat_publn

Moving from MySQL to SQL Server

Use TOP instead of LIMIT

Purpose: Retrieve only some rows for testing a query.

Example: Retrieve 500 random rows

- SELECT *
  FROM tls201_appln
  LIMIT 500

MySQL

- SELECT TOP 500 *
  FROM tls201_appln

SQL Server

Use OFFSET and FETCH instead of LIMIT

Example: Retrieve 500 rows, skipping the first 10 000

- SELECT *
  FROM tls201_appln
  ORDER BY appln_filing_date
  LIMIT 10000, 500

MySQL

- SELECT *
  FROM tls201_appln
  ORDER BY appln_filing_date
  OFFSET 10000 ROWS
  FETCH NEXT 500 ROWS ONLY

SQL Server

String delimiter

MySQL as well as SQL Server use the single quote (‘) to delimit strings. Not recommended and allowed only in MySQL, is the use of the double quote (“) as string delimiter.

- SELECT *
  FROM tls202_appln_title
  WHERE appln_title = 'Milk pump'

MySQL SQL Server

- SELECT *
  FROM tls202_appln_title
  WHERE appln_title = “Milk pump”

not recommended

MySQL

Functions

Every SQL dialect has its own set of functions. So frequently function names are different or require different parameters.

- SELECT *
  FROM tls202_appln_title
  WHERE LENGTH(appln_title) < 10

MySQL

- SELECT *
  FROM tls202_appln_title
  WHERE LEN(appln_title) < 10

SQL Server

NATURAL JOIN

NATURAL JOINs are **not supported by SQL Server**.

```
SELECT *
FROM tls202_appln_title
NATURAL JOIN tls201_appln
WHERE appln_title LIKE 'bicycle transmission%'
```

MySQL

This can be re-written:

```
SELECT *
FROM tls202_appln_title
JOIN tls201_appln ON tls202_appln_title.appln_id = tls201_appln.appln_id
WHERE appln_title LIKE 'bicycle transmission%'
```

SQL Server

Restrictions of GROUP BY clause

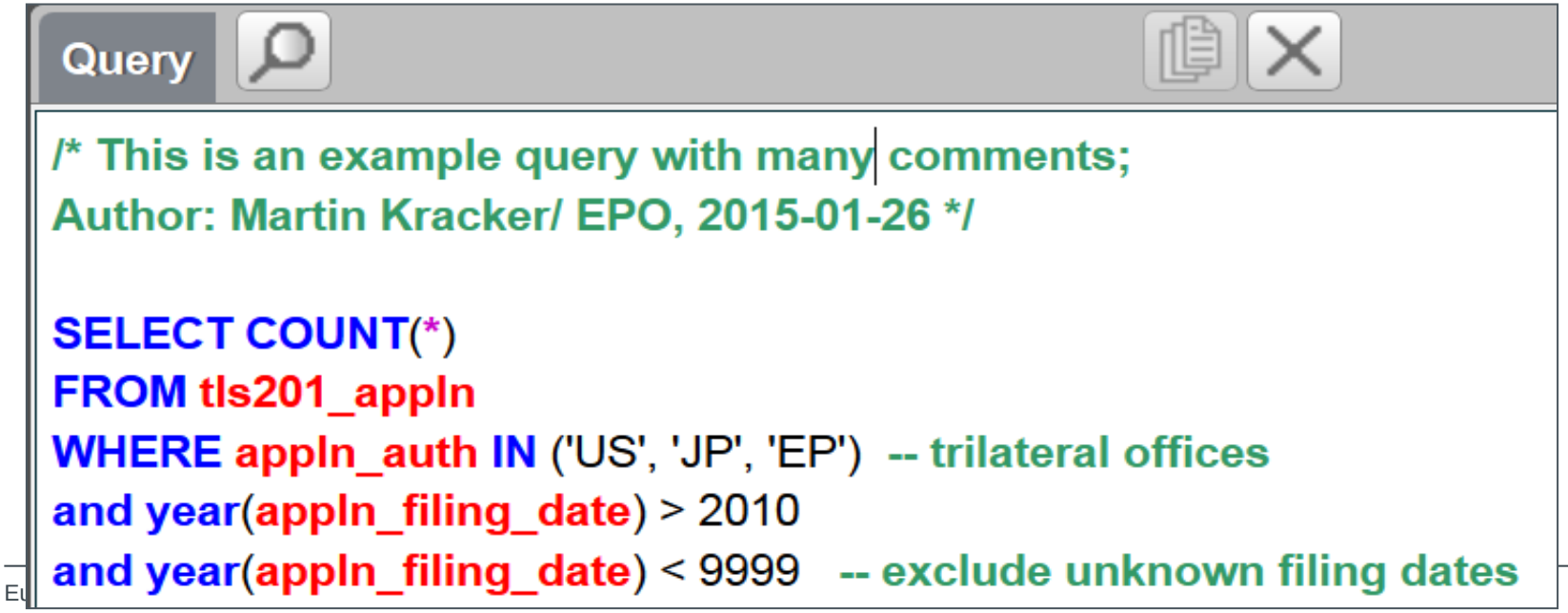
Restrictions:

- In SQL Server, when using GROUP BY the SELECT clause must only contain
  - attributes listed in the GROUP BY clause or
  - aggregated attributes, like COUNT()

- In contrast, MySQL allows in the SELECT clause also attributes which are neither grouped nor aggregated, but it may return any value of that attribute which occurs in the grouped rows.
  In short, MySQL is more liberal here, on the cost of retrieving unexpected results.

Comments

- SQL Server does not support
  - comment after the #-character
- SQL Server (like MySQL) supports these types of comments:
  - comment between ***/* .. */***
  - comment after two minus signs **--** till the end of the line



Your first query

SQL: Your first query

- Example:

```
SELECT appln_id
FROM tls201_appln
WHERE appln_auth = 'IE'
```

retrieves all application IDs which were filed in Ireland.

SELECT <attribute name> ← define result fields
FROM <table name> ← define tables used
WHERE <condition> ← optional; condition (which rows?)

Note: SQL language is not case-sensitive

Exercise:

Type the query of the last slide in the query window and execute it

Hint:

Use <Enter> for a new line,
<Tab> for indentation

Press <Ctrl + Enter> to execute the query

The 3 main SQL keywords: SELECT - FROM - WHERE

Query

Every query starts with SELECT

- SELECT lists the **columns** (attribute) you want to see in your result
- FROM identifies the **table** you want to query
- WHERE defines the **rows** you want to retrieve

Result

The result is always another table, which contains

- all the rows which fit the condition of the WHERE clause
- exactly the columns you listed in your SELECT clause

One more example

```
SELECT appln_auth, appln_nr,  appln_kind, appln_filing_date
FROM tls201_appln
WHERE appln_auth = 'IE'
```

Result table (based on table tls201_appln)

FROM

SELECT

appln_auth	appln_nr	appln_kind	appln_filing_date
IE	87	D2	9999-12-31
IE	142	A	1942-01-02
IE	144	A	1944-01-03
IE	145	A	1944-12-07
IE	147	A	1947-01-02
IE	148	A	1948-01-02
IE	150	A	1950-01-03
IE	151	A	1950-12-27
IE	156	A	1956-01-03

WHERE

Querying a single table

SELECT clause

The select clause may contain

- a single attribute
- a list of attributes, separated by comma
- an asterisk (*), which stands for "all attributes of the table(s)"

Examples

- SELECT appln_id
- SELECT appln_auth, appln_nr, appln_kind, appln_filing_date
- SELECT *

Exercise:

Use the previous query to retrieve

- several attributes of table tls201_appln
- all attributes of this table

Note: The previous query was

```
SELECT _________________
FROM tls201_appln
WHERE appln_auth = 'IE'
```

WHERE clause (1)

The WHERE clause contains one or more conditions.

The general form of a condition is:

<attribute or constant> <comparison operator> <attribute or constant>

The most common **comparison operators** and their meanings are

- = equal
- <> or != not equal
- >, >= larger, larger or equal
- <, <= smaller, smaller or equal

Typically, an attribute is compared to a constant. Examples:

- appln_auth = 'IE' ← Strings must be enclosed by single quotes (‘)
- appln_id <> 1234711 ← Number are not enclosed in quotes
- appln_filing_date >= '2001-01-01' ← Dates are enclosed by single quotes (‘) and have the form YYYY-MM-DD

You can also compare 2 attributes:

- appln_filing_year = earliest_publn_year ← = Applications which are filed and published in the same year

Comparison operator: IN

Some of these following operators can be prefixed with the keyword NOT, which reverses its meaning.

IN, NOT IN

- Tests if the value of an attribute is contained in a list of values.

Examples:

- appln_auth IN ('EP', 'JP', 'US') ← limit to trilateral offices
- appln_auth NOT IN ('US', 'CA') ← all offices except US and CA

Comparison operator: BETWEEN

BETWEEN, NOT BETWEEN

- Tests if the value of an attribute is between 2 values

Example:

- appln_filing_date BETWEEN '2001-04-01' AND '2001-09-30'

Instead BETWEEN, you could use the operators >= and <=

- appln_filing_date >= '2001-04-01' AND appln_filing_date <= '2001-09-30'

Comparison operator: LIKE

LIKE, NOT LIKE

- Tests whether 2 strings are similar.
  The %-wildcard represents zero, one or more characters (any characters)

Examples:

- person_name LIKE 'SIEMENS%'
- publn_kind LIKE 'A%'
- ipc_class_symbol like 'B60K%'
- appln_title LIKE '%hybrid%motor%'

WHERE clause (3)

The WHERE clause can contain more than 1 condition.

AND, OR and brackets can be applied as usual

Example:

- WHERE (appln_auth = 'EP'
  AND appln_filing_year = 2008)
  OR (appln_auth <> 'EP'
  AND appln_filing_year = 2010)

Exercise:

Create a query to retrieve all Austrian patent applications which where filed since 2005.

Hints:
- It is sufficient to use table `tls201_appln` only.
- Do not forget to exclude Austrian utility models
- Exclude the filing year 9999
- When in doubt, check the Data Catalog

TOP clause

You can limit the number of rows which would be retrieved.

F**ormat:** TOP immediately follows the SELECT

- TOP <number>

Example:

```
SELECT TOP 100 appln_id     ← retrieve the first 100 rows
FROM tls201_appln
WHERE appln_auth = 'IE'
```

Note: Without explicit sorting (ORDER BY clause) the sample you will retrieve is not defined.

Exercise:

Retrieve (any) 1000 rows of table `tls209_appln_ipc .`

Hint:
You do not need the WHERE clause.

ORDER BY clause (1)

You can order the result table by one or more columns.

The ORDER BY clause is always at the end of a query.

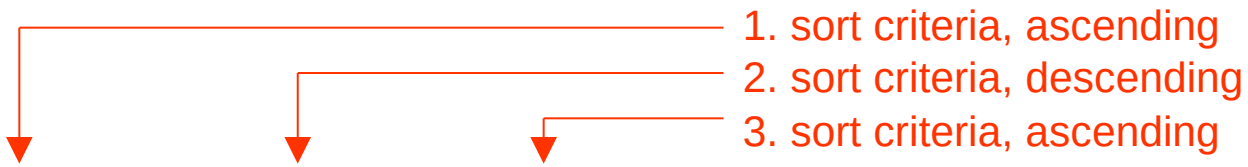
Allowed **formats:**

- ORDER BY <attribute> ← ascending order (default)
- ORDER BY <attribute> asc ← ascending order
- ORDER BY <attribute> desc ← descending order
- list of attributes with any sort order
  ORDER BY <attribute> desc, <attribute> asc, ...

ORDER BY clause (2)

Example:

```
SELECT publn_date, publn_auth, publn_nr, publn_kind, pat_publn_id, appln_id
FROM tls211_pat_publn
WHERE publn_date BETWEEN '2011-05-01' AND '2011-05-14'
ORDER BY  publn_date, publn_auth desc, publn_nr
```



Row	publn_date	publn_auth	publn_nr	publn_kind	pat_publn_id	appln_id
1	2011-05-01	TW	201114360	A	78856451	339052344
2	2011-05-01	TW	201114361	A	78856452	339052345
3	2011-05-01	TW	201114362	A	78856453	339052346
4	2011-05-01	TW	201114363	A	78856454	339052347
5	2011-05-01	TW	201114364	A	78856455	339052348
6	2011-05-01	TW	201114365	A	78856456	339052349

Exercise:

Take any query and sort the result

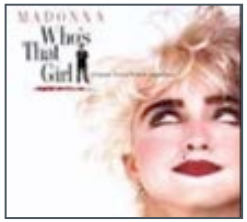
- Sort according to multiple columns

- Try ascending or descending sort order

Querying multiple tables

Why do we need more than 1 table? (1)

My CD collection:




Madonna

Rolling Stones

Shakira

Artist	Born	Grammy's	Album	Issued
Madonna	USA	7	Who's That Girl	1987
Madonna	USA	7	True Blue	1986
Madonna	USA	7	Rebel Heart	2015
Rolling Stones	n/a	2	Forty Licks	2002
Rolling Stones	n/a	2	Hyde Park Live	2013
Shakira	Colombia	2	Shakira	2014

Problem of data redundancy:

- Waste of storage space
- Risk of inconsistency when data must be updated (Grammy's)

Why do we need more than 1 table? (2)

Artists and albums are 2 different “things” (entities) which should be stored in separate tables.

Table ARTIST

Name	**Born**	**Grammy's**
Madonna	USA	7
Rolling Stones	n/a	2
Shakira	Colombia	2

Table ALBUM

Title	**Issued**	**Artist**
Who's That Girl	1987	Madonna
True Blue	1986	Madonna
Rebel Heart	2015	Madonna
Forty Licks	2002	Rolling Stones
Hyde Park Live	2013	Rolling Stones
Shakira	2014	Shakira

The **key** (see underlined column) uniquely identifies 1 entry in a table.
It is also called Primary Key (PK).

Why do we need more than 1 table? (3)

In real lives names and titles often are potentially not unique.

- A key could also be a combination of columns (e. g. first name, family name, birth date)
- An ID can be added to guarantee uniqueness

Table ARTIST

Artist_ID	Name	Born	Grammy's
1	Madonna	USA	7
2	Rolling Stones	n/a	2
3	Shakira	Colombia	2

Table ALBUM

Album_ID	Title	Issued	Artist_ID
1	Who's That Girl	1987	1
2	True Blue	1986	1
3	Rebel Heart	2015	1
4	Forty Licks	2002	2
5	Hyde Park Live	2013	2
6	Shakira	2014	3

A column which can be used to combine 2 tables is called a **Foreign Key (FK).** It is matched with the Primary key of the other table.

Why do we need more than 1 table? (4)

Gedankenexperiment: Reconstruct the original table (6 rows):

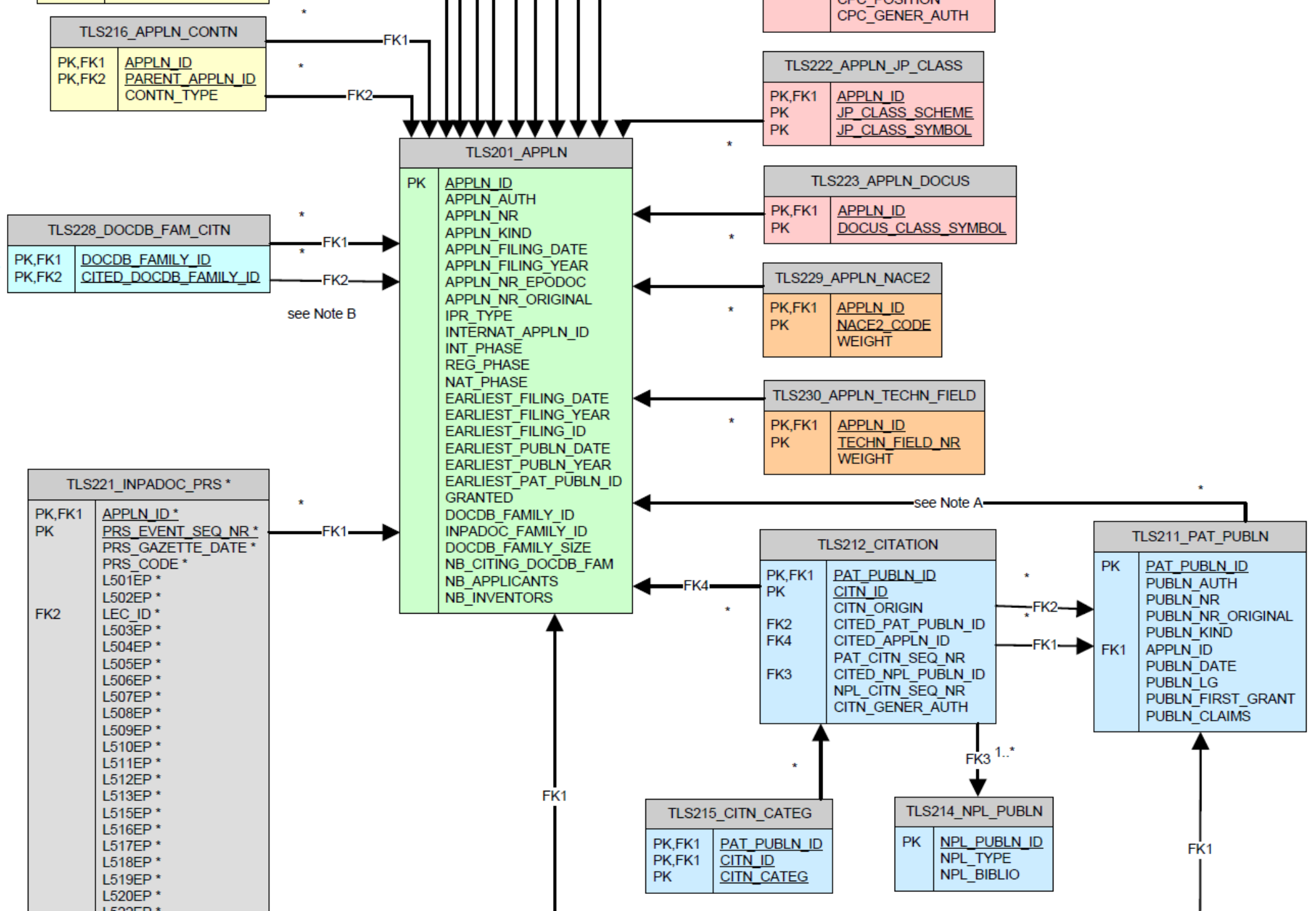
1) Combine every row of table 1 with every row of table 2 → 24 rows

2) Filter on all rows where the FK equals the PK → 6 original rows

Artist_ID (PK)	Name	Born	Grammy's	Album_ID	Title	Issued	Artist_ID (FK)
1	Madonna	USA	7	1	Who's That Girl	1987	1
1	Madonna	USA	7	2	True Blue	1986	1
1	Madonna	USA	7	3	Rebel Heart	2015	1
1	Madonna	USA	7	4	Forty Licks	2002	2
1	Madonna	USA	7	5	Hyde Park Live	2013	2
1	Madonna	USA	7	6	Shakira	2014	3
2	Rolling Stones	n/a	2	1	Who's That Girl	1987	1
2	Rolling Stones	n/a	2	2	True Blue	1986	1
2	Rolling Stones	n/a	2	3	Rebel Heart	2015	1
2	Rolling Stones	n/a	2	4	Forty Licks	2002	2
2	Rolling Stones	n/a	2	5	Hyde Park Live	2013	2
2	Rolling Stones	n/a	2	6	Shakira	2014	3
3	Shakira	Colombia	2	1	Who's That Girl	1987	1
3	Shakira	Colombia	2	2	True Blue	1986	1
3	Shakira	Colombia	2	3	Rebel Heart	2015	1
3	Shakira	Colombia	2	4	Forty Licks	2002	2
3	Shakira	Colombia	2	5	Hyde Park Live	2013	2
3	Shakira	Colombia	2	6	Shakira	2014	3

Links between tables (1)

- In PATSTAT, patent data is broken into multiple tables. So, typically you have to query multiple tables to get useful answers.
- Luckily, tables can be joined. The JOIN is a very powerful feature of SQL and you will use it for most queries.
- One JOIN clause can connect 2 tables.
  To connect multiple tables, multiple JOIN clauses must be used.


TLS216_APPLN_CONTN
PK,FK1 APPLN_ID
PK,FK2 PARENT_APPLN_ID
CONTN_TYPE
FK1
FK2
TLS222_APPLN_JP_CLASS
PK,FK1 APPLN_ID
PK JP_CLASS_SCHEME
PK JP_CLASS_SYMBOL
CPC_GENER_AUTH
TLS201_APPLN
PK APPLN_ID
APPLN_AUTH
APPLN_NR
APPLN_KIND
APPLN_FILING_DATE
APPLN_FILING_YEAR
APPLN_NR_EPODOC
APPLN_NR_ORIGINAL
IPR_TYPE
INTERNAT_APPLN_ID
INT_PHASE
REG_PHASE
NAT_PHASE
EARLIEST_FILING_DATE
EARLIEST_FILING_YEAR
EARLIEST_FILING_ID
EARLIEST_PUBLN_DATE
EARLIEST_PUBLN_YEAR
EARLIEST_PAT_PUBLN_ID
GRANTED
DOCDB_FAMILY_ID
INPADOC_FAMILY_ID
DOCDB_FAMILY_SIZE
NB_CITING_DOCDB_FAM
NB_APPLICANTS
NB_INVENTORS
TLS223_APPLN_DOCUS
PK,FK1 APPLN_ID
PK DOCUS_CLASS_SYMBOL
TLS228_DOCDB_FAM_CITN
PK,FK1 DOCDB_FAMILY_ID
PK,FK2 CITED_DOCDB_FAMILY_ID
FK1
FK2
see Note B
TLS229_APPLN_NACE2
PK,FK1 APPLN_ID
PK NACE2_CODE
WEIGHT
TLS230_APPLN_TECHN_FIELD
PK,FK1 APPLN_ID
PK TECHN_FIELD_NR
WEIGHT
TLS221_INPADOC_PRS *
PK,FK1 APPLN_ID *
PK PRS_EVENT_SEQ_NR *
PRS_GAZETTE_DATE *
PRS_CODE *
L501EP *
L502EP *
FK2 LEC_ID *
L503EP *
L504EP *
L505EP *
L506EP *
L507EP *
L508EP *
L509EP *
L510EP *
L511EP *
L512EP *
L513EP *
L515EP *
L516EP *
L517EP *
L518EP *
L519EP *
L520EP *
see Note A
TLS212_CITATION
PK,FK1 PAT_PUBLN_ID
PK CITN_ID
CITN_ORIGIN
FK2 CITED_PAT_PUBLN_ID
FK4 CITED_APPLN_ID
PAT_CITN_SEQ_NR
FK3 CITED_NPL_PUBLN_ID
NPL_CITN_SEQ_NR
CITN_GENER_AUTH
FK4
FK2
FK1
FK3 1..*
TLS211_PAT_PUBLN
PK PAT_PUBLN_ID
PUBLN_AUTH
PUBLN_NR
PUBLN_NR_ORIGINAL
PUBLN_KIND
FK1 APPLN_ID
PUBLN_DATE
PUBLN_LG
PUBLN_FIRST_GRANT
PUBLN_CLAIMS
FK1
TLS215_CITN_CATEG
PK,FK1 PAT_PUBLN_ID
PK,FK1 CITN_ID
PK CITN_CATEG
TLS214_NPL_PUBLN
PK NPL_PUBLN_ID
NPL_TYPE
NPL_BIBLIO
FK1

Links between tables (2)

- How tables should be joined in a meaningful way, is defined in the Data Catalog.

TLS202_APPLN_TITLE				
	APPLN_ID			
	APPLN_TITLE			
	PRIMARY KEY	APPLN_ID		
	FOREIGN KEY	APPLN_ID	REFERENCES	TLS201_APPLN (APPLN_ID)

TLS204_APPLN_PRIOR				
	APPLN_ID			
	PRIOR_APPLN_ID			
	PRIOR_APPLN_SEQ_NR			
	PRIMARY KEY	(APPLN_ID, PRIOR_APPLN_ID)		
	FOREIGN KEY	APPLN_ID	REFERENCES	TLS201_APPLN (APPLN_ID)
	FOREIGN KEY	PRIOR_APPLN_ID	REFERENCES	TLS201_APPLN (APPLN_ID)

Links between tables (3): person tables

Fragment of the real database (Oct 2011 version)

tls201_appln

appln_id	appln_auth	appln_nr	ipr_type ...
56	EP	040005863	PI

tls207_pers_appln

person_id	appln_id	applt_seq_nr	invt_seq_nr
27177351	56	0	3
33799676	56	1	0

tls206_person

person_id	person_ctry_code	person_name
27177351	DE	Pompe, Wolfgang
33799676	DE	Technische Uni..

Join possibility

JOIN clause (1)

```
SELECT *
FROM tls202_appln_title
JOIN tls201_appln  ON tls202_appln_title.appln_id = tls201_appln.appln_id
WHERE appln_title LIKE 'bicycle transmission%'
```

- The JOIN clause specifies the **table to be joined** with the other tables in the queries.
- The **join condition** (ON part of the JOIN clause) specifies the attributes which should be joined.

Result table 5/143

Row	appln_id	appln_title	appln_id	appln_auth	appln_nr	appln_kind	appln_filing_da..
1	1783717	Bicycle transmi...	1783717	AU	3744201	D	2001-02-27
2	2416396	Bicycle transmi...	2416396	AU	8843691	A	1991-09-24
3	2416397	BICYCLE TRAN...	2416397	AU	8843691	D	1991-09-24

Table Alias - Giving tables a short name

- Table names are quite long
- You can assign a short **alias** name to tables
- Aliases are specified in the query and only valid within the scope of the query
- Aliases have to be written directly after the table name
- → Queries are easier to read

```
SELECT *
FROM tls202_appln_title t
JOIN tls201_appln a ON t.appln_id = a.appln_id
WHERE appln_title LIKE 'bicycle transmission%'
```

- Note: Once a table alias is used, you cannot refer to the table by its full name within the same query.

Identification of attributes (1)

- If an attribute is unique within all tables of a query, it is sufficient to use only the (unqualified) attribute name.

  Examples:
  `appln_id`
  `ipc_class_symbol`

- If the same attribute name is used in multiple tables of a query, you must use the qualified form.
  Table name and attribute name are connected with a dot:
  `<table name>.<attribute name>`

  Examples:
  `tls201_appln.appln_id`
  `tls211_appln_ipc.ipc_class_symbol`

Identification of attributes (2)

- If you have defined a table alias, you must use this table alias and not the table name for qualified attribute names.

  Examples:
  ```
  a.appln_id
  i.ipc_class_symbol
  ```

- This will also make your queries easier to read

JOIN clause (2)

```
SELECT *
FROM tls209_appln_ipc
JOIN tls201_appln ON tls209_appln_ipc.appln_id = tls201_appln.appln_id
WHERE tls209_appln_ipc.appln_id = 456789
```

- 1 applications may have 0, 1 or many IPC symbols (1:n relationship)
- Restricting the query to 1 application, still retrieves several rows, with an identical application part

Result table 1/3

Row	appln_id	ipc_class_sym...	ip...	ipc_version	i...	i...	ip...	ipc_...	t...	appln_id	appln_auth	appln_nr	appln_kind
1	456789	G06F 13/00	A	2006-01-01	I	L	KR	G06F	6	456789	KR	20070006214	A
2	456789	G06F 15/00	A	2006-01-01	I	L	KR	G06F	6	456789	KR	20070006214	A
3	456789	H04L 12/28	A	2006-01-01	I	F	KR	H04L	4	456789	KR	20070006214	A

table tls209_appln_ipc

table tls201

Exercise:

Run this query which is similar to the previous one. It retrieves the IPC symbols of 3 applications:

```
SELECT *
FROM tls209_appln_ipc i
JOIN tls201_appln a  ON i.appln_id = a.appln_id
WHERE a.appln_id IN (456789, 456790, 456791)
```

How many rows do you get?
How many applications?
How many IPC symbols per application?

Exercise:

Write and run a query which joins the tables `tls201_appln, tls207_pers_appln, tls206_person`.

The Data Catalog tells you how to join the tables.

Use table aliases.

Restrict the result to
- applications with Russian (RU) application authority and
- with applicants / inventors from Spain (ES)

Exercise:

Re-write the previous query (4 table join) but retrieve only these columns:

`person_name` and `doc_std_name` of table `tls206_person` and all columns of table `tls201_appln`

INNER JOIN vs. OUTER JOIN

Up to now we only discussed INNER JOINs.

- The **INNER JOIN** returns corresponding rows in 2 tables. A row in the left table with no corresponding row in the right table (or vice versa) is not retrieved.
- In **OUTER JOIN** returns also rows which do not have a corresponding row in the other table.
  The attributes in the replenished rows of the other table contain the value NULL.

```
SELECT i.*, a.*
FROM tls201_appln a
JOIN tls209_appln_ipc i ON a.appln_id = i.appln_id
WHERE a.appln_id = 364137181
```

This query returns no rows, because the selected application does not have IPC symbols.

OUTER JOIN

```
SELECT i.*, a.*
FROM tls201_appln a
LEFT OUTER JOIN tls209_appln_ipc i ON a.appln_id = i.appln_id
WHERE a.appln_id = 364137181  -- same application as on the previous slide
```

- tls201_appln is the left table, tls209_appl_ipc is the right table.
- The LEFT OUTER JOIN retrieves for all rows of the left table the matching rows of the right table and - in case there are no matching rows - NULL values.

Result table 1 / 1

Row	appln_id	ipc_cl...	ipc_cl...	ipc_ve...	ipc_va...	ipc_p...	ipc_ge...	ipc_su...	techn_...	appln_id	appln_auth	appln_nr	appln_kind
1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	364137181	AR	P110101612	A

table tls209_appln_ipc | table tls201_appln

Although the application has no IPCs, it is retrieved.

SELECT clause revisited

The select clause may contain

- a single attribute
- a list of attributes, separated by comma:
  The attributes may be qualified or unqualified
- an asterisk (*), which stands for "all attributes of the table(s)"
- **an asterisk preceded by a table name / table alias which stands for "all attributes of this table"**
- **a list of anything from above, separated by comma**

Examples

- SELECT appln_id
- SELECT appln_auth, appln_nr, appln_kind, appln_filing_date
- **SELECT a.appln_id, tls212_citation.citn_origin**
- SELECT *
- **SELECT p.*, tls201_appln.*, i.ipc_class_symbol**

Grouping, counting and aggregating rows

What is a grouping? (1)

Given the table from a previous slide, we **define groups** e. g. **by the ARTIST** column:

Artist	Born	Grammy's	Album	Issued
Madonna	USA	7	Who's That Girl	1987
Madonna	USA	7	True Blue	1986
Madonna	USA	7	Rebel Heart	2015
Rolling Stones	n/a	2	Forty Licks	2002
Rolling Stones	n/a	2	Hyde Park Live	2013
Shakira	Colombia	2	Shakira	2014

There are 3 groups (here: colour coded) which can be described:

- by their size: “Madonna” has 3 rows, Shakira has 1 row
- by their group column
- but NOT by any other column

GROUP BY clause (1)

```
SELECT appln_auth, appln_filing_year
FROM tls201_appln
WHERE appln_filing_year BETWEEN 2005 AND 2008
GROUP BY appln_auth, appln_filing_year
ORDER BY appln_filing_year, appln_auth
```

- The GROUP BY is followed by one or more attributes.
- All rows which have the same values for these attributes are compressed into 1 row

Result table 12/477

Row	appln_auth	appln_filing_year
1	AE	2005
2	AE	2006
3	AE	2007
4	AE	2008
5	AM	2005
6	AM	2006
7	AM	2007

GROUP BY clause (2)

```
SELECT appln_auth, appln_filing_year, COUNT(*)
FROM tls201_appln
WHERE appln_filing_year BETWEEN 2005 AND 2008
GROUP BY appln_auth, appln_filing_year
ORDER BY appln_auth, appln_filing_year
```

The count(*) returns the number of rows (here: applications) which are compressed into a single row by the GROUP BY clause (here: filing year and office)

Result table 1/477

Row	appln_auth	appln_filing_year	count(*)
1	AE	2005	2
2	AE	2006	4
3	AE	2007	5
4	AE	2008	5
5	AM	2005	17
6	AM	2006	14
7	AM	2007	13

In plain words:
This query computes the number of applications per office and filing year.

GROUP BY clause (3)

- When using GROUP BY the SELECT clause may only contain
  - attributes listed in the GROUP BY clause or
  - aggregated attributes, like COUNT ()

- Example of an invalid query:
  SELECT appln_auth, appln_filing_year, COUNT(*),
  appln_id, appln_filing_date
  FROM tls201_appln
  WHERE appln_filing_year BETWEEN 2005 AND 2008
  GROUP BY appln_auth, appln_filing_year
  ORDER BY appln_auth, appln_filing_year

Exercise:

Write a query which, for Danish (DK) applications, returns the number of applications per publication year of the first publication.
The order should be chronologically reversed (most recent on top).

Hint: Consider attribute

```
tls201_appln.publn_earliest_year
```

Attribute Aliases

```
SELECT appln_auth AS office, appln_filing_year, COUNT(*) AS nbOfAppln
FROM tls201_appln
WHERE appln_filing_year BETWEEN 2005 AND 2008
GROUP BY appln_auth, appln_filing_year
ORDER BY nbOfAppln desc
```

- You can rename any column in the SELECT clause by adding **AS <alias>** after each attribute / aggregated attribute
- This is often used for renaming computed attributes
- The alias can be utilized in the ORDER BY clause, but not in the WHERE clause

Result table 15/477

Row	office	appln_filing_year	nbOfAppln
1	US	2007	495717
2	US	2005	494518
3	US	2006	488045
4	US	2008	480633
5	CN	2008	465773
6	JP	2005	462611
7	JP	2006	443554

Aggregate functions (1)

Aggregate functions work on a set of values and return a single value.

- When there is no GROUP BY clause, the function works on the complete result table
- When there is a GROUP BY clause, the function works on each group separately

- **COUNT()** Number of rows containing a Non-NULL value
- **SUM()** Sum
- **AVG()** Average
- **MIN()** Minimum
- **MAX()** Maximum
- **STD()** Std. deviation

Aggregate functions (2)

An aggregate function requires an argument which may be:

- an **asterisk (*):**
  This is the most common form, like COUNT(*)
  → the function is applied on all rows (of a group)
- an **attribute**:
  E. g. SUM(publn_claims)
  → the function is applied on all rows (of a group) which have a value other than NULL[1] in that attribute
- **DISTINCT <attribute>**
  E. g. COUNT(DISTINCT appln_auth)
  → the function is applied on all rows (of a group) but considers each value of the attributes only once[2]

[1] NULL is a special value which is used to express "do not know", "not applicable" or the like, but which is neither zero nor empty. See also the section about OUTER JOINs.

[2] See also slide for the DISTINCT keyword

Exercise:

Write a query which retrieves all Swiss (CH) applications per filing year:
- the number of applications
- the average number of applicants per application
- the maximum number of applicants per application

Give all attributes descriptive aliases.

Hint: Make use of the available pre-computed attributes in PATSTAT.
Hint: Use “AVG(nb_applicants ***1.0**)” in the SELECT clause to return the average as decimal number (without rounding).

Filtering on groups

Example: Groups defined by Artist

Artist	Born	Grammy's	Album	Issued
Madonna	USA	7	Who's That Girl	1987
Madonna	USA	7	True Blue	1986
Madonna	USA	7	Rebel Heart	2015
Rolling Stones	n/a	2	Forty Licks	2002
Rolling Stones	n/a	2	Hyde Park Live	2013
Shakira	Colombia	2	Shakira	2014

Besides the data in the group columns, groups have certain characteristics which can be used to filter groups.

Examples:

- size of group
- latest album release year

HAVING clause

```
SELECT appln_auth, appln_filing_year, COUNT(*) as NbOfApplns
FROM tls201_appln
WHERE appln_filing_year BETWEEN 2005 AND 2008
GROUP BY appln_auth, appln_filing_year
HAVING COUNT(*) > 100000
ORDER BY appln_auth, appln_filing_year
```

- Purpose is to restrict aggregated values of a group
- Accepts conditions like in the WHERE clause
- To be used together with a GROUP BY clause
- HAVING clause is positioned directly after the GROUP BY clause

Result table 13 / 22

Row	appln_auth	appln_filing_year	NbOfApplns
1	CN	2005	291455
2	CN	2006	346826
3	CN	2007	391414
4	CN	2008	469529
5	DE	2005	112432
6	DE	2006	104068
7	EP	2005	159562

Exercise:

Write a query which retrieves all Austrian (AT) applications filed in the year 2006 which have been published more than 2 times.

Hints:
- Join the application table with the publication table.
- use GROUP BY and HAVING

Subqueries

Subqueries

A *subquery* is a query which is *nested* within an *outer query*.

```
SELECT *
FROM tls211_pat_publn
WHERE publn_claims =
    (SELECT MAX(publn_claims)
    FROM tls211_pat_publn
    )
```

- Subqueries are always put in brackets
- Subqueries may or may not be correlated to the outer query
- Subqueries often are a simple, alternative way to formulate a query.

Subquery in the SELECT clause

A *subquery* in the SELECT clause must always return a single value of a single row.

```
SELECT publn_nr, publn_date,
   (SELECT appln_filing_date
   FROM tls201_appln a
   WHERE p.appln_id = a.appln_id -- correlates outer and nested query
   ) as filingDate
FROM tls211_pat_publn p
WHERE publn_auth = 'NL'
```

- The subquery is correlated with the outer query by "**WHERE p.appln_id = a.appln_id**"

Subquery in the WHERE ... IN clause

The IN operator checks whether a value falls within the set of values retrieved by the subquery.

```
SELECT appln_auth, appln_nr, appln_kind
FROM tls201_appln a
JOIN tls209_appln_ipc i ON a.appln_id = i.appln_id
WHERE ipc_class_symbol IN
    (SELECT ipc_class_symbol
    FROM tls209_appln_ipc
    WHERE appln_id = 100072)
```

The query above retrieves all applications which have at least one IPC symbol in common with a specific other application.

Subquery in the WHERE ... EXISTS clause

The EXISTS operator checks whether the subquery retrieves any row at all. The subquery may return any number of rows or values.

```
SELECT appln_auth, appln_nr, appln_kind
FROM tls201_appln a
WHERE EXISTS
   (SELECT * FROM tls209_appln_ipc i
   WHERE a.appln_id = i.appln_id
   AND ipc_class_symbol = 'B60K   1/00')
AND EXISTS
   (SELECT * FROM tls209_appln_ipc i
   WHERE a.appln_id = i.appln_id
   AND ipc_class_symbol = 'B60W  30/18')
```

The query above retrieves all applications which have **at least these two** IPC symbols.

More useful SQL features

Comments

- A comment is a text which is ignored by the query engine. That is, you can write anything you find useful.
- A comment is ...
  - everything between ***/* .. */***
  - everything after two minus signs **--** till the end of the line

Query

```
/* This is an example query with many comments;
Author: Martin Kracker/ EPO, 2015-01-26 */

SELECT COUNT(*)
FROM tls201_appln
WHERE appln_auth IN ('US', 'JP', 'EP')  -- trilateral offices
and year(appln_filing_date) > 2010
and year(appln_filing_date) < 9999   -- exclude unknown filing dates
```

DISTINCT keyword

```
SELECT DISTINCT ipc_class_symbol
FROM tls202_appln_title t
JOIN tls209_appln_ipc i  ON t.appln_id = i.appln_id
WHERE appln_title LIKE 'bicycle transmission%'
ORDER BY ipc_class_symbol
```

- DISTINCT removes duplicate rows
- DISTINCT follows directly after the SELECT keyword

Result table 11 / 116

Row	ipc_class_symbol
1	A63B 22/06
2	B60B 27/02
3	B60B 27/04
4	B60K 17/36
5	B60K 20/00
6	B60K 20/02
7	B62J 13/00

Functions

Functions can be extremely useful.

- A function takes an attribute (and sometimes a list of attributes) as argument.
- A function returns a single value
- In a query, you always can replace an attribute by a function

```
SELECT TOP 100  UPPER(person_name)  +  '; '  +  person_ctry_code
FROM tls206_person
WHERE CHARINDEX('SONY', person_name) = 0
    -- SONY must not be part of the name
AND LEN(RTRIM(LTRIM(person_address))) >= 20
    -- It is assumed that only addresses with more than 20 characters are complete
```

String Functions (1)

String functions work with string attributes.

- person_name **+ '; ' +** person_ctry_code:
  Use the plus sign to concatenate multiple strings into one string
- **LEN(**appln_title**)**
  returns length of a string
- **UPPER(**person_name**)** , **LOWER(**person_name**)**
  returns the string in upper / lower case
- **LTRIM(**person_address**) / RTRIM(**person_address**)**
  returns the string without leading / trailing spaces.

String Functions (2)

- **CHARINDEX(**'SONY‘, person_name**)**
  returns the position where the string SONY is located in person_name. Zero is returned if SONY is not found.
- **SUBSTRING(**ipc_class_symbol, 5, 4**)**
  returns 4 characters starting with the 5th (=main group)
- **LEFT(**ipc_class_symbol, 4**)**
  returns the first 4 characters (= IPC subclass)
- **REPLACE(**person_address, 'Wien', 'Vienna'**)**
  returns a string where all occurrences of 'Wien' are replaced by 'Vienna'

Datetime Functions

These functions work with date attributes.

- **GETDATE**()
  requires no argument and returns the current date
- **DATEADD**(month, 18, appln_filing_date**)**
  DATEADD(month, -18, appln_filing_date**)**
  adds / subtracts 18 month to/from the filing date; can be also used to add years, days, ...
- **YEAR(**appln_filing_date**)**
  MONTH(appln_filing_date**)**
  DAY(appln_filing_date**)**
  returns the specified part of the date

Arithmetic operators

You can do the usual math with number attributes or constants.

```
SELECT top 100 appln_auth, appln_nr, appln_kind, nb_applicants, nb_inventors,
nb_citing_docdb_fam,
earliest_publn_year - appln_filing_year as yearsToPublish,
(nb_applicants * 10 + nb_inventors) * nb_citing_docdb_fam / 2.0
    as myIndicator
FROM tls201_appln
```

+ add	/ divide
- subtract	% modulo (the remainder of a division)
* multiply	

To force a result in decimal format, make sure that at least one argument is in decimal format: e. g. **nb_citations / 2.0**

Frequently used JOINs

Titles and abstracts

Each application has one or none title / abstract. Sometimes the title/abstract is missing, so OUTER JOIN is used.

```
SELECT a.appln_id, appln_auth, appln_nr, appln_kind, t.appln_title
FROM tls201_appln a
LEFT OUTER JOIN tls202_appln_title t ON a.appln_id = t.appln_id
WHERE -- please complete ....
```

Row	appln_id	appln_auth	appln_nr	appln_kind	appln_title
1	599180	AT	12026D	A	Verfahren zur Reinigung vor Zuckersäften.
2	900012445	AT	12026T	A	NULL **there is no title for this application**
3	599181	AT	120260	A	Auf Gleisen fahrbare Maschine, insbesondere Gleisstopfmaschine
4	599182	AT	120261	A	Verfahren und Schaltungsanordnung zur Überprüfung und Korrektur von aus co
5	599183	AT	120262	A	Verfahren zum Kühlen und Wärmeisolieren der Werkzeuge von Strangpressen

Classifications

Each application has none, one or more classifications of a certain type (like IPC, CPC, JP, US, ...). Sometimes classifications are missing, so we use an OUTER JOIN.

```
SELECT a.appln_id, appln_auth, appln_nr, appln_kind, i.ipc_class_symbol
FROM tls201_appln a
LEFT OUTER JOIN tls209_appln_ipc i ON a.appln_id = i.appln_id
WHERE -- please complete ....
```

Row	appln_id	appln_auth	appln_nr	appln_kind	ipc_class_symbol
28	21000020	GB	166139	A	H01M 2/20
29	21000021	GB	166142	A	E04B 2/02
30	21000021	GB	166142	A	E04B 2/28

Applicants and inventors

There is a n:m relationship between applications and applicants. That means, each application may be related to 0, 1 or more applicants, and 1 applicant may have 0, 1 or more applications. This query retrieves applicants.

```
SELECT a.appln_id, appln_auth, appln_nr, appln_kind, p.person_id,
p.person_name
FROM tls201_appln a
LEFT OUTER JOIN tls207_pers_appln pa ON a.appln_id = pa.appln_id
LEFT OUTER JOIN tls206_person p ON pa.person_id = p.person_id
WHERE applt_seq_nr > 0
AND -- please complete ....
```

Row	appln_id	appln...	appln_nr	appln_kind	person_id	person_name
1	21000010	GB	166121	A	39167330	WILHELM CORSALLI
2	21000011	GB	166122	A	16101564	JAMES PHILIP CAMPBELL
3	21000011	GB	166122	A	21102369	LEONARD MILLER
4	21000011	GB	166122	A	23940522	METROPOLITAN-VICKERS ELECTRICAL COMPANY LIMITED

Priorities

Other than you probably expected, the priority table tls204_appln_prior is just a linking table. It links an application with each of its (Paris convention) priorities

```
SELECT a1.appln_id, a1.appln_auth, a1.appln_nr, a1.appln_kind,
a2.appln_id as prio_id, a2.appln_auth as prio_auth, a2.appln_nr as prio_nr,
a2.appln_kind as prio_kind
FROM tls201_appln a1
LEFT OUTER JOIN tls204_appln_prior pr ON a1.appln_id = pr.appln_id
LEFT OUTER JOIN tls201_appln a2 ON pr.prior_appln_id = a2.appln_id
WHERE -- please complete ....
```

Row	appln_id	appln_auth	appln_nr	appln_kind	prio_id	prio_auth	prio_nr	prio_kind
9	21000009	GB	166120	A	NULL	NULL	NULL	NULL
10	21000010	GB	166121	A	901711342	DE	313620T	D
11	21000010	GB	166121	A	901803109	DE	94057X	D
12	21000011	GB	166122	A	NULL	NULL	NULL	NULL

Applications

Priority applications

Families

Each application belongs to exactly one DOCDB and exactly one INPADOC family[1].
The query below retrieves all INPADOC family members of application with ID 21000000.

```
SELECT a2.appln_id, a2.inpadoc_family_id, a2.appln_auth, a2.appln_nr,
a2.appln_kind, a2.appln_filing_date
FROM tls201_appln a1
JOIN tls201_appln a2 ON a1.inpadoc_family_id = a2.inpadoc_family_id
WHERE a1.appln_id = 21000000  -- as an example
```

Row	appln_id	inpadoc_family_id	appln_auth	appln_nr	appln_kind	appln_filing_date
1	3284949	1879637	BE	710170D	A	1968-01-31
2	9588219	1879637	DE	1615691	A	1968-02-10
3	17965995	1879637	ES	349844	A	1968-01-27
4	20873235	1879637	FR	1552002D	A	1968-01-31
5	21000000	1879637	GB	166069	D	1968-01-11
6	41670276	1879637	NL	6801694	A	1968-02-07
7	49980934	1879637	US	3545080D	A	1967-05-16
8	905249276	1879637	US	64263967	A	1967-05-16

The application itself is of course member of its own family

[1] Exception: Artificial applications (appln_id >= 900 000 000) do not belong to any DOCDB simple family.

Publications

Publications are all in table tls211_pat_publ.
In many case you would combine the publication with application data. Because every publications belongs to an application, a (INNER) JOIN is sufficient.

```
SELECT publn_nr, publn_kind, publn_date
FROM  tls211_pat_publn p
JOIN tls201_appln a ON p.appln_id = a.appln_id
WHERE appln_filing_date = '2007-07-07'    -- for example
```

Result table 7/593

Row	publn_nr	publn_kind	publn_date
1	2009007664	A1	2009-01-15
2	2047358	A2	2009-04-15
3	1900455	A1	2008-03-19
4	1894761	B1	2009-10-28
5	1894761	A1	2008-03-05

Citations to patent publications (1)

Publications may contain citations (tls212_citation). Citations can refer to other **publications**, **NPL** or **applications**.

This query retrieves citations to patent publications only.

```
SELECT c.citn_origin, c.pat_citn_seq_nr , p2.publn_auth, p2.publn_nr,
p2.publn_kind
FROM tls211_pat_publn p1
LEFT JOIN tls212_citation c ON p1.pat_publn_id = c.pat_publn_id
JOIN tls211_pat_publn p2 ON c.cited_pat_publn_id =  p2.pat_publn_id
        -- citations
WHERE c.pat_citn_seq_nr > 0    -- retrieve only citations to publications
AND p1.publn_auth = 'EP'      -- for example
AND p1.publn_nr = '1674117'
AND p1.publn_kind = 'A1'
ORDER BY citn_origin
```

Citations to patent publications (2)

- One publication may cite several other publications
- One cited publication may be cited several times by different publications

Result table 1 / 10

Row	citn_origin	pat_citn_seq_nr	publn_auth	publn_nr	publn_kind
1	APP	1	US	6299438	B1
2	APP	2	US	6099561	A
3	APP	3	DE	19916086	A1
4	APP	4	US	2001036530	A1
5	APP	5	US	6534197	B2
6	EXA	1	US	6110204	A
7	SEA	1	US	6099561	A
8	SEA	2	DE	19916086	A1
9	SEA	3	US	2001036530	A1
10	SEA	4	US	6299438	B1

NPL (Non Patent Literature)

This query retrieves cited NPL.

```
SELECT c.citn_origin, c.npl_citn_seq_nr , n.npl_biblio
FROM tls211_pat_publn p1
LEFT JOIN tls212_citation c ON p1.pat_publn_id = c.pat_publn_id
JOIN tls214_npl_publn n ON c.npl_publn_id =  n.npl_publn_id
        -- citations
WHERE c.npl_citn_seq_nr > 0    -- retrieve only citations to NPL
AND p1.publn_auth = 'EP'      -- for example
AND p1.publn_nr = '1674117'
AND p1.publn_kind = 'A1'
ORDER BY citn_origin
```

Result table 1 / 1

Row	citn_origin	npl_citn_seq_nr	npl_biblio
1	APP	1	PANTCHOHA ET AL. STOMATOLOGIYA vol. 65, no. 5, 1986, pages 51 - 3

Citation categories (1)

All citations may be categorized (X, Y, A, D, ...) and have 0, 1 or more rows in table tls215_citn_categ.

This query retrieves all categories of all citations of a certain publication,

```
SELECT c.citn_id, c.pat_citn_seq_nr, c.npl_citn_seq_nr, c.citn_origin,
ccat.citn_categ
FROM tls211_pat_publn p
JOIN tls212_citation c ON p.pat_publn_id = c.pat_publn_id
LEFT JOIN tls215_citn_categ ccat ON c.pat_publn_id = ccat.pat_publn_id
                        AND c.citn_id = ccat.citn_id  -- category of citations
WHERE p.publn_auth = 'EP'   -- for example
AND p.publn_nr = '1674117'
AND p.publn_kind = 'A1'
ORDER BY c.citn_id
```

Citation categories (2)

The query retrieved:

Result table 1 / 12

Row	citn_id	pat_citn_seq_nr	npl_citn_seq_nr	citn_origin	citn_categ
1	1	1	0	SEA	X
2	2	2	0	SEA	X
3	3	3	0	SEA	X
4	4	4	0	SEA	X
5	5	1	0	APP	NULL
6	6	2	0	APP	NULL
7	7	3	0	APP	NULL
8	8	4	0	APP	NULL
9	9	5	0	APP	NULL
10	10	0	1	APP	NULL
11	11	1	0	EXA	A
12	11	1	0	EXA	D

Tips for writing queries

1) Clarify the information need

- Write down your information need
  - Choose the right aggregation level: Are you interested in families, applications, publications or any other "units"?
  - What filter conditions have to be applied to limit the sample?
  - What information should the output contain?

- Identify the tables and attributes you need in your query to create the output or define the conditions
  - Consult the Data Catalog on the meaning of tables & content
  - Chose the appropriate date for your analysis: priority date, filing date, publication date, date of grant ...
  - Chose the right classification, family, IP right, ...

2) Check coverage and quality

- Is the needed information available in PATSTAT ? (e.g. the publications of a certain country and period)

- See http://www.epo.org/searching-for-patents/helpful-resources/raw-data/data/tables/weekly.html

- Run test queries to check the actual data content
  - Use TOP clause to limit the output to a sensible amount
  - Use GROUP BY to count the number of rows of certain countries, types, kind codes, number ranges, time periods, ...
  - Is the outcome as expected? If not, consult the Data Catalog and re-check.

3) Query formulation

- Start with a simple query and incrementally extend it with more JOINs and condition as required.

- Between steps: verify that you actually retrieve the data you want. You can check applications in detail:
  - With Espacenet, when searching with APPLN_NR_EPODOC or PUBLN_AUTH & PUBLN_NR
  - In PATSTAT Online: In the Result Window

- Repeat the cycle:
  Run – Check – Enhance query – Run – ....

- When using a locally installed PATSTAT database: avoid complex queries. Creating intermediate tables and check.

4) Empower your queries

Avoid using repetitive queries, where only the year, the office, the classification etc. is manually changed. Use the power of SQL to get your results in a single query, with e. g.

- the IN operator to compare a list of values
- the OR operator, e. g. with a LIKE operator and wildcards
- the GROUP BY when counting rows grouped by some criteria
- Often subqueries or the EXISTS operator are useful

5) Improve performance

Speed up your query when execution time is too long.

- If you use your own database:
  Make sure you have defined indexes for attributes used in JOIN and WHERE clauses. Over indexing decreases query time and does not harm if you do not change the source data.

  In addition, you may precompute & store values which you require frequently and which are costly to compute on the fly.

- In PATSTAT Online all sensible indexes have already been defined. If still necessary, you may have a look at the costs shown in the Estimated Query Plan to fine tune your query:
  Press <Ctrl><lower case L> in the query editor and check the result table.

6) Downloading and post-processing

PATSTAT Online allows you to retrieve large sets of data. This is ideally for further "offline" analysis or visualisation with your own favourite tools. See the PATSTAT Online User Manual for details.
Download the Result Table, as returned by your query

- Download a consistent database (e. g. in MS Access format), as defined by your query with the “Subset Download” feature.

Wrap up

General structure of SELECT

SELECT [DISTINCT] [TOP n] attribute(s) or expression(s)
FROM table
JOIN table **ON** join condition(s) -- JOIN clause may repeat
WHERE condition(s)
GROUP BY attribute(s) or expression(s)
HAVING group condition(s)
ORDER BY attribute(s) or expression(s)

Resources

PATSTAT resources for download

(Data Catalog, sample queries, tips, ..)

- In PATSTAT Online go to menu Help ¦ Database Help
  or
- go directly to tab “Downloads”
  http://www.epo.org/searching-for-patents/business/patstat.html

SQL - The query language

- there are numerous books and documents on the Internet
- The official T-SQL / MS SQL Server resource for the SELECT statement:
  https://msdn.microsoft.com/en-us/library/ms189499.aspx

Human support

- PATSTAT discussion forum: http://forums.epo.org/patstat/
- Questions and feedback patstat@epo.org