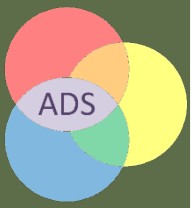
Data Science & Society

Lecture 07:
SQL & Spark

INFOMDSS 2018 :: Dr. Marco Spruit



Agenda

- Book review grades
- Tweaked Course Schedule
  - http://bit.ly/infomdss-schedule
- This Thursday 14:00-14:30: Office hour Midterm exam review
- Extra assignment Big Spatial Data

- SQL in context and examples
  - NoSQL, RDD, ...
- SQL in Spark
  - CH 10 intro

Course Schedule 2018

http://bit.ly/infomdss-schedule

infomdss2018	Date	Monday C3	Tuesday C4 Workshops	Tuesday C5 Lectures	Thursday C7 lectures	Weekly assignments	Weekly readings	DevTest Lab
week 1 (37)	2018-09-03		N/A	N/A	Regular lecture (MS): Course introduction	1. Book review: - Submit Top 3 Books	1. Davenport & Patil (2012) 2. Stair & Reynolds (2012) - CH 1,3 3. Pritzker, P., and May, W. (2015) - CH 2, App. A 4. Chapman et al. (2000) - CH 1,2 5. Spruit & Lytras (2018)	N/A
week 2	2018-09-10		Required attendance: - Tutorial 1A: Azure VMs - Tutorial 1B: Linux Bash	Regular lecture (MS): Big Data Engineering with Hadoop	Regular lecture (MS): Hadoop MapReduce and HDFS	1. Complete Tutorial 1: - Bash in Ubuntu 2. Start Tutorial 2: - Wordcount in Hadoop	- White (2015) - CH 1,2 - Dean & Ghemawat (2008) - Get & Read your selected book to review	LabA: Ubuntu 18 1 core, 2 MB
week 3	2018-09-17		Hands-on Tutorial 2: Wordcount in Hadoop	Required: Guest lecture (DV): UMCU/Neonatology on Big Data for Small Babies	Regular lecture (MS): Spark Architecture & Transformations	1. Complete Tutorial 2: - Wordcount in Hadoop 2. Start Tutorial 3: - Neonatology Part I	- White (2015) - CH 3,19 - Complete reading your selected book to review	LabB: Ubuntu 18 + Hadoop artifact 2 cores, 4 MB
week 4	2018-09-24	Python tutorial & Q/A	1. Complete Tutorial 3: - Neonatology Part I 2. Start Tutorial 3: - Neonatology Part II	Guest lecture (SM): ORTEC on Big Data Knowledge Discovery	Wrap-up lecture (MS): - Wide Transformations - Walkthrough of Tutorial 3 in Hadoop & Spark - Midterm Q/A	1. Submit Book 2-pager 2. Complete Tutorial 3: - Neonatology Parts I,II	- Complete readings above - Review tutorials to understand at the command level - Review _all_ lecture slides	LabB: Ubuntu 18 + Hadoop artifact 2 cores, 4 MB
week 5	2018-10-01	MIDTERM EXAM	NO LAB	Student pich session: - The TOP-20 Data Science & Society books	Regular lecture (MB): Methods & Statistics I	1. Complete Tutorial 4 - Neonatology in Spark 2. Start Tutorial 5 - Statistics in Jupyter	- Lazer et al. (2014, March 28) - Broniatowski et al. (2014, July 28) - Chambers & Zaharia (2018) - CH 1	LabA: Ubuntu 18 DSVM 2 cores, 4 MB
week 6 (42)	2018-10-08		1. Complete Tutorial 5 - Statistics in Jupyter	Regular lecture (MB): - Methods & Statistics II	Required: Guest lecture (COM): UMCU/Epidemiology on Menopause and Cardiometabolic Disease Risk	1. Continue Tutorial 5 - Statistics in Jupyter 2. Start Tutorial 6 - Epidemiology Analytics in Jupyter Part I	- Chambers & Zaharia (2018) - CH 2	LabA: Ubuntu 18 DSVM 2 cores, 4 MB
week 7	2018-10-15		1. Explain assignment Big Spatial Data 2. Complete Tutorial 6 Part I - Epidemiology Analytics in Jupyter Part I	Regular lecture (MB+MS): - Methods & Statistics Wrap-up - SQL in Spark	Guest lecture (TB,JWvE): ESRI NL on Big Data in Geographical Information Systems	1. Complete Tutorial 6 Part I - Epidemiology Data Preprocessing 2. Start Tutorial 6 Part II - Epidemiology Analytics	- Chambers & Zaharia (2018) - CH 10	LabA: Ubuntu 18 DSVM 2 cores, 4 MB
week 8	2018-10-22		1. Complete Tutorial 6 Part II - Epidemiology Analytics	Guest lecture (WO): - CoreLifeAnalytics on Machine Learning in Cell Screening	Guest lecture (MM): UMCU/Julius on Big Data Ethics in Research, Privacy and Data Protection	1. Complete Tutorial 6 Part II - Epidemiology Data Analytics	- Complete ALL readings above - Review ALL tutorials to understand at the command level - Review ALL lecture slides	LabB: HDInsight VMs on Azure DataBricks cluster
week 9	2018-10-29		1. Complete Tutorial 6 Part II - Epidemiology Data Analytics 2. Work on Tutorial 6 Part III - Epidemiology Advanced Analytics	Guest lecture (FS,VM): UMCU/Psychiatry on Big Data in Psychiatry	FINAL LECTURE (MS): - "Towards Tomorrow: Trends in Data Science & Society" - Endterm Q/A	1. Complete Tutorial 6 Part III - Epidemiology Advanced Analytics	- Complete ALL readings above - Review ALL tutorials to understand at the command level - Review ALL lecture slides	LabB: HDInsight VMs on Azure DataBricks cluster
week 10	2018-11-05				ENDTERM EXAM			
week 1	2010-01-03				2ND CHANCE EXAM			

All multiple choice

Learning objectives in midterm exam

	Litera-ture	Work-shops	Guest talks	Regular lectures	TOTAL
Understand the role of data science and its societal impact	9		1		10
Recognise the knowledge discovery processes in applied data science	6				6
Identify trends and developments in big data engineering & analytics			1	15	16
Apply selected big data technologies to solve real-world problems		37		26	63
TOTAL	15	37	2	41	**95**

Extra assignment Big Spatial Data

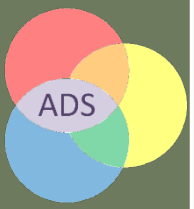
- Playful Preparation for Guest lecture:
  - ESRI NL on Big Data in Geographical Information Systems
  - PDF available on Teams: url

Materials from Dr. Sergio España

SQL in context and examples

Refreshing your memory
& CH3 of Fundamentals of IS (Stair & Reynolds)

Structured Query Language (SQL)'s DDL & DML



Structured Query Language

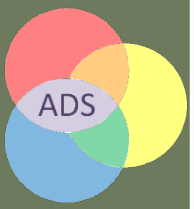
- Abbreviation: SQL
- Pronounced as: ‘S-Q-L’ [‘Ess-Que-El’]
- Initially introduced as ‘SEQUEL’ in the 70s
- SQL-92 is a 1992 ANSI standard
- SQL:2011 is the current standard

- Apache Spark implements subset of SQL:2003.

- *“Structured Query Language* is a domain-specific language for expressing relational operations over data” (CH 10)

SQL vs NoSQL

Quotes from: *Cattell, R. (2011). Scalable SQL and NoSQL data stores. ACM Sigmod Record, 39(4), 12-27.*



"Not Only SQL" or "Not Relational"

SQL vs NoSQL: *the* survey

› Cattell, R. (2011). Scalable SQL and NoSQL data stores. *ACM Sigmod Record*, *39*(4), 12-27.

› *NoSQL systems generally have six key features:*

1. the ability to horizontally scale "simple operation" throughput over many servers,
2. the ability to replicate and to distribute (partition) data over many servers,
3. a simple call level interface or protocol (in contrast to a SQL binding),
4. a weaker concurrency model than the ACID transactions of most relational (SQL) database systems,
5. efficient use of distributed indexes and RAM for data storage, and
6. the ability to dynamically add new attributes to data records.

SQL vs NoSQL: *the* survey

- A key feature of NoSQL systems is "shared nothing" horizontal scaling – replicating and partitioning data over many servers
- No ACID transactional properties: updates are eventually propagated, but there are limited guarantees on the consistency of reads. Some authors suggest a "BASE" acronym in contrast to the "ACID" acronym:
  - **BASE** = Basically Available, Soft state, Eventually consistent
  - **ACID** = Atomicity, Consistency, Isolation, and Durability
- The idea is that by giving up ACID constraints, one can achieve much higher performance and scalability.

- Sounds a lot like… Hadoop??

SQL vs NoSQL: The argument for relational

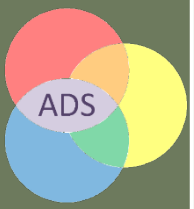
1. If new relational systems can do everything that a NoSQL system can, with analogous performance and scalability, and with the convenience of transactions and SQL, why would you choose a NoSQL system?
2. Relational DBMSs have taken and retained majority market share over other competitors in the past 30 years: network, object, and XML DBMSs.
3. Successful relational DBMSs have been built to handle other specific application loads in the past: read-only or read-mostly data warehousing, OLTP on multi-core multi-disk CPUs, in-memory databases, distributed databases, and now horizontally scaled databases.
4. While we don't see "one size fits all" in the SQL products themselves, we do see a common interface with SQL, transactions, and relational schema that give advantages in training, continuity, and data interchange.

SQL vs NoSQL: The counter-argument for NoSQL

1. We haven't yet seen good benchmarks showing that RDBMSs can achieve scaling comparable with NoSQL systems like Google's BigTable.
2. If you only require a lookup of objects based on a single key, then a key-value store is adequate and probably easier to understand than a relational DBMS. Likewise for a document store on a simple application: you only pay the learning curve for the level of complexity you require.
3. Some applications require a flexible schema, allowing each object in a collection to have different attributes. While some RDBMSs allow efficient "packing" of tuples with missing attributes, and some allow adding new attributes at runtime, this is uncommon.
4. A relational DBMS makes "expensive" (multinode multi-table) operations "too easy". NoSQL systems make them impossible or obviously expensive for programmers. [un-ease of use??]
5. While RDBMSs have maintained majority market share over the years, other products have established smaller but non-trivial markets in areas where there is a need for particular capabilities, e.g. indexed objects with products like BerkeleyDB, or graph-following operations with object-oriented DBMSs.

SQL DDL

Data Definition Language



SQL data definitions

- CREATE
  - Creation of database objects
- ALTER
  - Modify the structure and/or the characteristics of database objects
- DROP
  - Deletion of database objects
- TRUNCATE
  - Deletion of data in tables without altering the structure

SQL DDL and SQL DML

- SQL is no ‘classical’ programming language, it is more a ‘data sub language’
- Core parts of SQL:
  - **Data definition language (DDL)**: used to define database structures
  - **Data manipulation language (DML)**: define, update, and request data (queries)

The CREATE TABLE statement

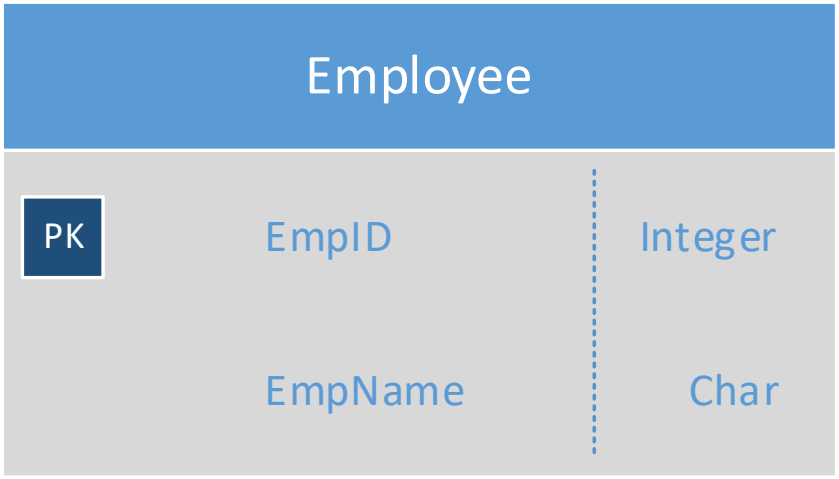
- Creation of tables
  - The SQL CREATE TABLE statement

  - CREATE TABLE EMPLOYEE(

    EmpID Integer PRIMARY KEY,

    EmpName Char(25) NOT NULL

    );

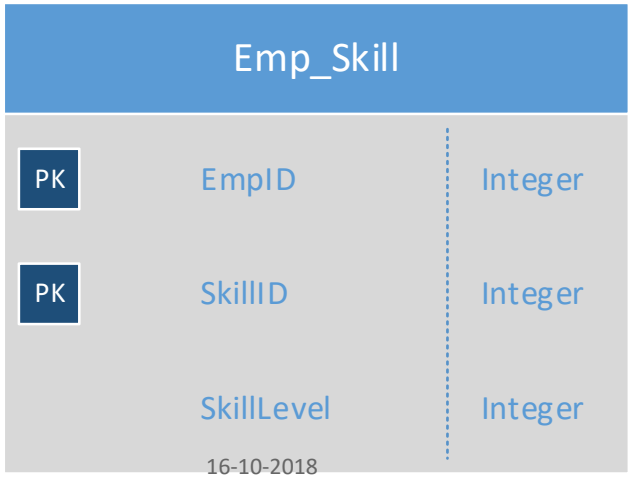


Composite PRIMARY KEYS

› Creation of database tables with composite keys

– CREATE TABLE EMP_SKILL(

```
EmpID        Integer      NOT NULL,
SkillID      Integer      NOT NULL,
SkillLevel   Integer      NULL,
CONSTRAINT   EmpSkill_PK  PRIMARY KEY (EmpID, SkillID)
);
```



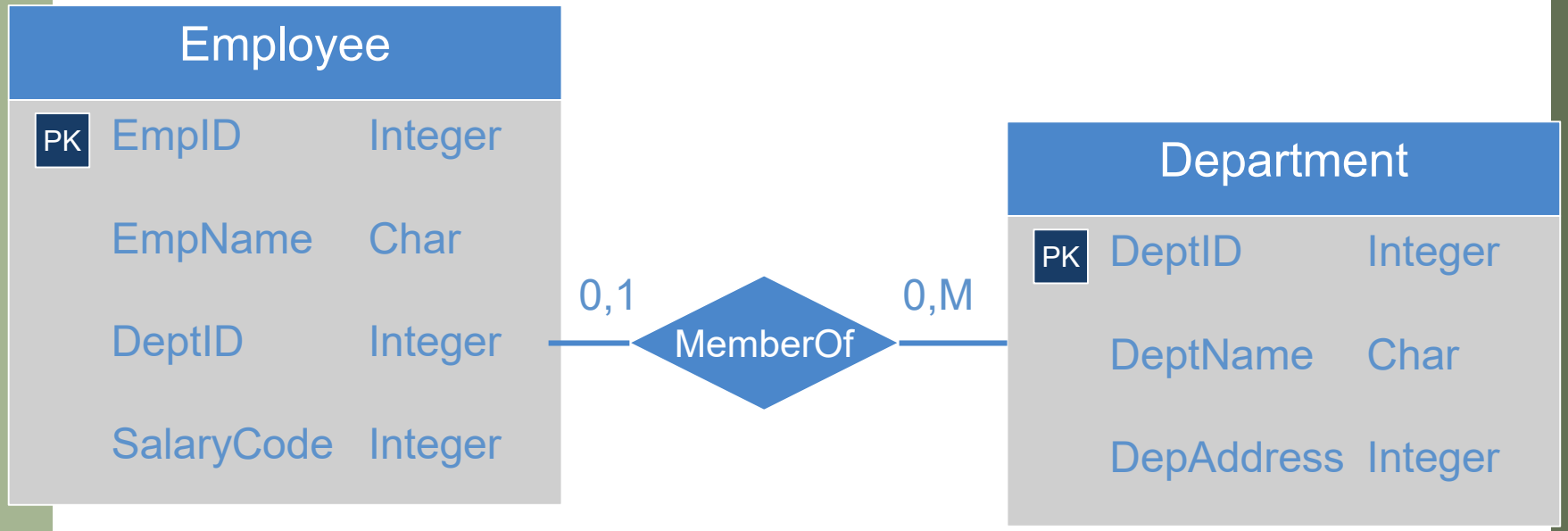
The FOREIGN KEY constraint (1/2)

› Creation of database tables using PRIMARY KEY and FOREGN KEY constraints

```
– CREATE TABLE EMP_SKILL(
    EmpID          Integer        NOT NULL,
    SkillID        Integer        NOT NULL,
    SkillLevel     Integer        NULL,
    CONSTRAINT  EmpSkill_PK  PRIMARY KEY (EmpID, SkillID),
    CONSTRAINT  Emp_FK       FOREIGN KEY(EmpID)
                             REFERENCES EMPLOYEE(EmpID),
    CONSTRAINT  Skill_FK     FOREIGN KEY(SkillID)
                             REFERENCES SKILL(SkillID)
    );
```

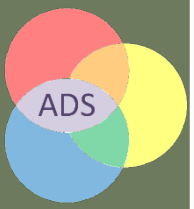
Deletion of database objects

- DROP: Deletion of unwanted database objects
- Warning! The DROP keyword permanently deletes the data object along with the data related to that object
  - DROP TABLE EMPLOYEE;



SQL DML

Data Manipulation Language



The INSERT keyword

- Use the INSERT keyword to add a row in a table
- Non-numerical data needs to be within single quotes (')
- INSERT INTO EMPLOYEE VALUES(91, 'Smither', 12);
- INSERT INTO EMPLOYEE (EmpID, SalaryCode) VALUES (62, 11);

Employee			
EmpID	**EmpName**	**DeptID**	**SalaryCode**
91	Smither	12	
62	*Null!*	*Null!*	11

The UPDATE keyword

› Modify values in an existing row (or a collection of rows)

```
UPDATE  EMPLOYEE
SET     Phone '791-555-123'
WHERE   EmpID = 29;

UPDATE  EMPLOYEE
SET     DeptID = 14
WHERE   EmpName LIKE 'Kr%';
```

Employee				
EmpID	**EmpName**	**DeptID**	**SalaryCode**	**Phone**
91	Smither	12	10	773-334-333
33	Krusty	12	11	445-445-334
29	Homer	15	8	666-666-666

The DELETE keyword

› Delete a row or a set of rows

```
DELETE FROM EMPLOYEE
WHERE  EmpID = 29;

DELETE FROM EMPLOYEE
WHERE  EmpName LIKE 'Kr%';
```

Employee				
EmpID	**EmpName**	**DeptID**	**SalaryCode**	**Phone**
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

Queries

- A query retrieves data from one or more tables and creates a new (temporary) table
- SELECT is the best-known SQL statement

```
SELECT EmpName
FROM  EMPLOYEE
WHERE  EmpID = 91;
```

Results

EmpName
Smither

Employee

EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

Showing all values from a table

› Use an asterisk (*) to show all values in a table that match certain criteria.

Results

EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333

```
SELECT *
FROM EMPLOYEE
WHERE EmpID = 91;
```

Employee

EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

The DISTINCT keyword

› The DISTINCT keyword can be added to the SELECT statement to prevent that duplicate rows are shown

```
SELECT  DISTINCT DeptID
FROM    EMPLOYEE;
```

Results

DetptID
12
14
15

Employee

EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

The WHERE keyword (1/2)

› With the WHERE keyword criteria can be provided which the shown records should meet

```
SELECT   EmpName
FROM     EMPLOYEE
WHERE    DeptID = 15;
```

Results
EmpName
Homer
Krall

Employee				
EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

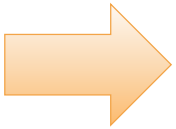
The WHERE keyword (2/2)

- The WHERE clause may include the following symbols:
  - Equals “=”
  - Not Equals “<>”
  - Greater than “>”
  - Less than “<”
  - Greater than or Equal to “>=”
  - Less than or Equal to “<=”

Logical AND- and OR-operators

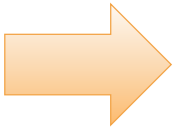
› Multiple matching criteria can be specified by using:
  – AND: represents an intersection of data collections
  – OR: represents a union of data collections

```
SELECT    EmpName
FROM      EMPLOYEE
WHERE     DeptID < 13 OR
                DeptID > 14;
```



Results
EmpName
Smither
Homer
Krall

```
SELECT    EmpName
FROM      EMPLOYEE
WHERE     DeptID = 15 AND
                SalaryCode <= 8;
```



Results
EmpName
Homer

The IN keyword

› The WHERE clause can include the IN keyword to make sure that certain column values are included in the query result

```
SELECT   EmpName
FROM     EMPLOYEE
WHERE    DeptID IN (4, 12, 14);
```

Results

EmpName
Smither
Krusty

Employee

EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

Logical NOT-operator

› The logical NOT-operator is used to prevent records from being part of the resulting set of records

```
SELECT  EmpName
FROM    EMPLOYEE
WHERE   DeptID NOT IN (4, 12, 14);
```

Results
EmpName
Homer
Krall

Employee				
EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

The BETWEEN keyword

› The BETWEEN keyword is used to show records where the values in a column are between a minimum and maximum value

```
SELECT  EmpName
FROM    EMPLOYEE
WHERE   SalaryCode BETWEEN 7 AND 9;
```

Results
EmpName
Homer

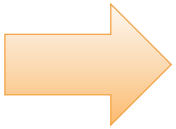
Employee				
EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

The LIKE keyword

- By means of the LIKE keyword partially complete values can be searched for
- Use wildcards to find records that are equal to a certain string value
  - Percentage symbol (%) for multiple arbitrary characters
  - Underscore (_) for a single arbitrary character

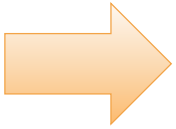
Examples

```
SELECT   EmpID
FROM     EMPLOYEE
WHERE  EmpName LIKE 'Kr%';
```



Results
EmpID
33
32

```
SELECT   EmpID
FROM    EMPLOYEE
WHERE  Phone LIKE '___-___-33_';
```



Results
EmpID
91
33

Employee				
EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

Sorting of results

› Query results are sorted by means of the ORDER BY clause:

```
SELECT EmpID, EmpName
FROM EMPLOYEE
ORDER BY EmpName;
```

Results

EmpID	EmpName
29	Homer
32	Krall
33	Krusty
91	Smither

Employee

EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

Built-in SQL functions

› SQL offers various built-in functions:

- COUNT : counts the number of rows that match the criteria as posed in the query
- MIN: search for the minimum value in a certain column
- MAX: search for the maximum value in a certain column
- SUM: computes the sum of values in a certain column
- AVG: computes the average of the values in a certain column

Examples

```
SELECT COUNT(DeptID)
FROM   EMPLOYEE;
```

Results
COUNT(DeptID)
4

```
SELECT COUNT(DISTINCT DeptID)
FROM   EMPLOYEE;
```

Results
COUNT(DISTINCT DeptID)
3

Employee				
EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

Examples

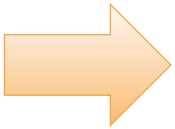
```
SELECT MIN(Hours) AS MinimumHours,
            MAX(Hours) AS MaximumHours,
            AVG(Hours) AS AverageHours
FROM        PROJECT
WHERE ProjID > 7;
```

```
SELECT SUM(Hours) AS TotalHours,
FROM   PROJECT;
```

Showing sub totals: GROUP BY

› Sub totals are computed by means of the GROUP BY clause

```
SELECT   DeptID,
         COUNT(*) AS NumEmp
FROM     EMPLOYEE
GROUP BY DeptID;
```



Results

DeptID	NumEmp
12	1
14	1
15	2

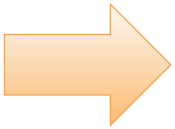
Employee

EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

Showing sub totals: GROUP BY

- Sub totals are computed by means of the GROUP BY clause
- The HAVING clause is used to limit how much data is shown

```
SELECT   DeptID,
         COUNT(*) AS NumEmp
  FROM     EMPLOYEE
  GROUP BY DeptID
  HAVING   COUNT(*) > 1;
```



Results

DeptID	NumEmp
15	2

Employee

EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

Example: “HAVING COUNT(*) > 1” for Privacy

- Example of people with a disease, stored with their postal code and their ethnicity.
- Implements “2-anonymity” when the person is not allowed to see detailed results (SELECT * FROM XXX WHERE allowed = 0)
- And all the detailed results when the person is allowed (SELECT * FROM XXX WHERE allowed = 1).

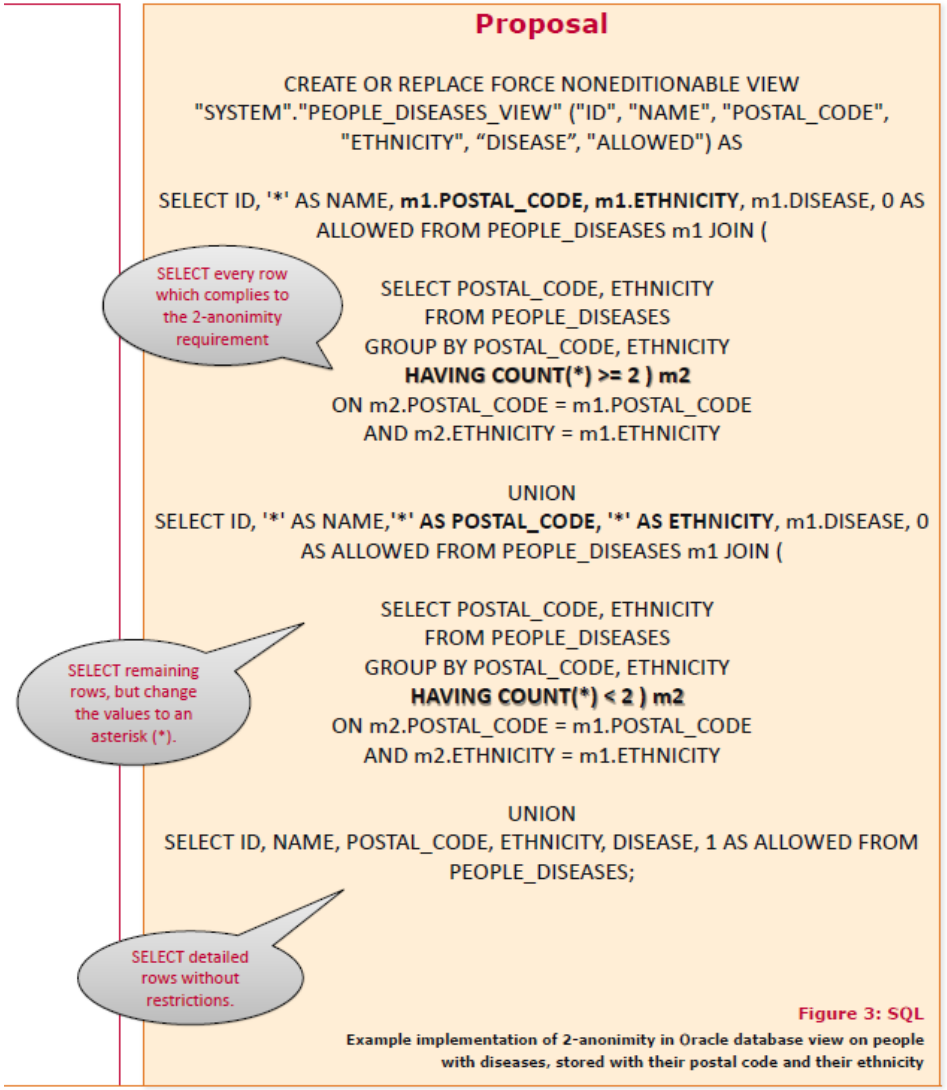


Figure 3: SQL
Example implementation of 2-anonimity in Oracle database view on people with diseases, stored with their postal code and their ethnicity

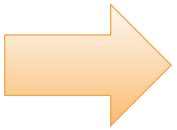
- Toledo,C. van, & Spruit,M. (2016). Adopting privacy regulations in a data warehouse: A case of the anonimity versus utility dilemma. In Fred,A. et al. (Ed.), *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management* (pp. 67–72). KDIR 2016, November 11-13, 2016, Porto, Portugal: ScitePress. [pdf]

Data from multiple tables

- Subqueries
  - The result of a query is a subset of the data
  - Therefore, a query can be used as input for another query
  - This is called a subquery
- Joins
  - A different way to combine data can be realized with a join
    - Join (can also be called Inner Join)
    - Left Outer Join
    - Right Outer Join

Subquery example

```
SELECT EmpName
FROM   EMPLOYEE
WHERE  DeptID IN
      (SELECT DeptID
       FROM   DEPARTMENT
       WHERE  DeptName
              LIKE 'Accou%');
```



Results

EmpName
Smither
Krusty

Department

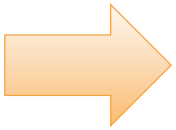
DeptID	DeptName	DeptAddress
12	Accountancy	Floor 3
14	Accounts VIP	Floor 4
15	Production	Warehouse A

Employee

EmpID	EmpName	DeptID	SalaryCode	Phone
91	Smither	12	10	773-334-333
33	Krusty	14	11	445-445-334
29	Homer	15	8	791-555-123
32	Krall	15	10	554-678-889

Join example

```
SELECT  EmpName
FROM    EMPLOYEE AS E,
            DEPARTMENT AS D
WHERE  E.DeptID = D.DeptID
            AND  D.DeptName
                    LIKE 'Accou%';
```



Results
EmpName
Smither
Krusty

Department		
DeptID	DeptName	DeptAddress
12	Accountancy	Floor 3
14	Accounts VIP	Floor 4

Employee JOIN Department							
E.EmpID	E.EmpName	E.DeptID	E.SalaryCode	E.Phone	D.DeptID	D.DeptName	D.DeptAddress
91	Smither	12	10	773-334-333	12	Accountancy	Floor 3
33	Krusty	14	11	445-445-334	14	Accounts VIP	Floor 4
29	Homer	15	8	791-555-123	15	Production	Warehouse A
32	Krall	15	10	554-678-889	15	Production	Warehouse A

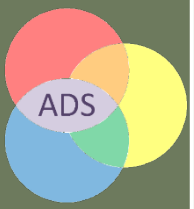
The JOIN…ON syntax

› Moving the JOIN syntax to the FROM clause:

```
SELECT EmpName
FROM   EMPLOYEE AS E JOIN DEPARTMENT AS D
     ON E.DeptID = D.DeptID
WHERE  D.DeptName LIKE 'Account%';
```

SQL in Apache Spark

Best of Both Worlds!



SQL in the Spark Architecture

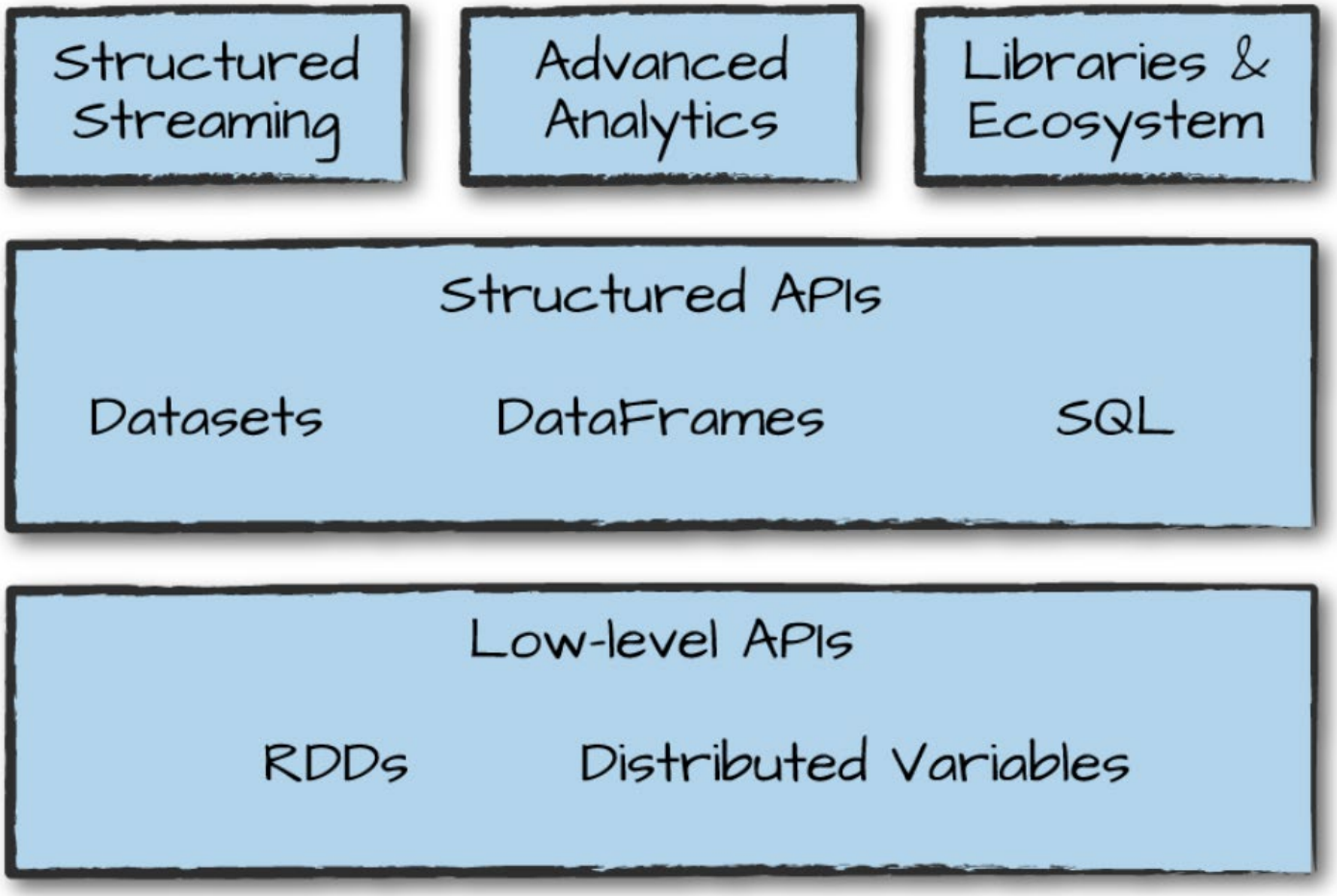


Figure 1-1. Spark's toolkit

Big Data and SQL: How, until Spark 2?

- Apache Hive
  - Before Spark's rise, Hive was the de facto big data SQL access layer
  - Originally developed at Facebook, Hive became an incredibly popular tool across industry for performing SQL operations on big data
  - Why important for Hadoop adoption and cross-industry success?
    - because analysts could run SQL queries
- Why are now many users using Spark SQL, instead of...?
  - Resilient Distributed Datasets (RDDs)
- Compatible with Hive
- Interoperable with DataFrames

Using SQL in Spark

- **Spark SQL CLI:** ./bin/spark-sql
- **Spark's SQL API:**
  - via the method sql on the SparkSession object.
  - returns a DataFrame
  - spark.sql("SELECT 1 + 1").show()
    - Is a *lazy* transformation
- A Table is logically equivalent to a DataFrame

```
CREATE TABLE flights_csv (
  DEST_COUNTRY_NAME STRING,
  ORIGIN_COUNTRY_NAME STRING COMMENT "remember, the US will be most prevalent",
  count LONG)
USING csv OPTIONS (header true, path '/data/flight-data/csv/2015-summary.csv')

CREATE TABLE flights (
  DEST_COUNTRY_NAME STRING, ORIGIN_COUNTRY_NAME STRING, count LONG)
USING JSON OPTIONS (path '/data/flight-data/json/2015-summary.json')

INSERT INTO flights_from_select
  SELECT DEST_COUNTRY_NAME, ORIGIN_COUNTRY_NAME, count FROM flights LIMIT 20
```

SQL Queries in Spark

Select Statements

Queries in Spark support the following ANSI SQL requirements (here we list the layout of the SELECT expression):

```
SELECT [ALL|DISTINCT] named_expression[, named_expression, ...]
    FROM relation[, relation, ...]
    [lateral_view[, lateral_view, ...]]
    [WHERE boolean_expression]
    [aggregation [HAVING boolean_expression]]
    [ORDER BY sort_expressions]
    [CLUSTER BY expressions]
    [DISTRIBUTE BY expressions]
    [SORT BY sort_expressions]
    [WINDOW named_window[, WINDOW named_window, ...]]
    [LIMIT num_rows]

named_expression:
    : expression [AS alias]

relation:
    | join_relation
    | (table_name|query|relation) [sample] [AS alias]
    : VALUES (expressions)[, (expressions), ...]
          [AS (column_name[, column_name, ...])]

expressions:
    : expression[, expression, ...]

sort_expressions:
    : expression [ASC|DESC][, expression [ASC|DESC], ...]
```

A different IF-THEN

› **case…when…then** Statements
  – conditionally replace values in SQL queries
  – essentially the equivalent of programmatic if statements

```
SELECT
  CASE WHEN DEST_COUNTRY_NAME = 'UNITED STATES' THEN 1
       WHEN DEST_COUNTRY_NAME = 'Egypt' THEN 0
       ELSE -1 END
FROM partitioned_flights
```