Lecture 4:
SQL2, Queries on Multiple Tables

Instructor: Shel Finkelstein

Reference:
A First Course in Database Systems,
3rd edition, Chapter 6.2, 6.3, 6.4

Important Notices

- Lecture slides and other class information will be posted on Piazza (not Canvas).
  - Slides are posted under Resources→Lectures
  - Lecture Capture recordings are available to all students under Yuja.
    - Recording for CSE 180 lecture for Wednesday, January 17 was posted directly on Yuja, since I was at a conference on that date.
  - There's no Lecture Capture for Lab Sections.
- All Lab Sections (and Office Hours) normally meet In-Person, with rare exceptions announced on Piazza.
  - Some slides for Lab Sections will be posted on Piazza under Resources→Lab Section Notes
- Office Hours for TAs and me are posted in Syllabus and Lecture1, as well as in Piazza notice @7.

Important Notices

- Lab1 solution was posted on Piazza under Resources→Lab1 on Wed, January 24.
- Lab2 assignment was posted on Piazza under Resources→Lab2 on Wed, January 24. General Information about appears under Resources-->General Resources.
  - See Piazza announcement about Lab2, which is much more difficult than Lab1.
  - Lab2 is due **Tuesday, February 6**, by 11:59pm.
    - Late Lab Assignments will not be accepted; be sure that you post the correct file!
    - Be sure to do Lab assignments individually, but you can get help on concepts.
  - Your solution should be submitted via Canvas as a zip file.
    - Canvas will be used for both Lab Assignment submission and grading.
  - You'll probably have enough information to complete Lab2 after the Lecture on Fri, January 26
  - A load file to help you test your Lab2 solution was posted under Resources→Lab2
    - But testing can only prove that a solution is incorrect, not that it is correct.
    - We won't provide answers for your Lab2 queries on that test data.
  - Using ChatGPT to help understand CSE 180 concepts (or even ask what SQL queries you wrote actually do) is great …
    - … but if you do use systems such at ChatGPT to do Lab Assignments, you're mostlt cheating yourself …
    - … because you can't use such systems on the Midterms or Final!
  - Lab2 will be discussed at Lab Sections and Tutoring.

Important Notices

- First Gradiance Assignment, CSE 180 Fall 2023, was posted on Thursday, January 18, and is due (on Gradiance) on **Friday, January 26**, by 11:59pm.
  - The day of the week on which Gradiance Assignments are due will vary.
  - For Gradiance this quarter, the token is **EC7184D3**
  - There are 9 questions in this Gradiance Assignment.
  - You may take each Gradiance Assignment as many time as you like.
    - Questions stay the same, but answers change.
    - Your Gradiance Assignment score is the score on your **Final** Gradiance submission before the deadline.
    - Gradiance does not allow late submissions, so no extensions are possible.
  - Suggest that you use your cruzid (or something similar) as your Gradiance login.
    - Be sure to enter your UCSC email address as your email.
    - If you forget your Gradiance password, you can recover it by providing your Gradiance login and email.
  - After Friday, January 19 Lecture, you should have enough info to complete this Gradiance Assignment.

Group Tutor-1

Alex Asch
alasch@ucsc.edu

Tutoring Times:

- Tuesday: 1:00pm – 2:00pm
- Friday: 9:20am – 10:25am
- Friday: 1:30pm – 2:30pm

Tutoring will be in Jack's Lounge, on the first floor of Baskin Engineering.

Group Tutor-2

Yash Raj Singh
ysingh5@ucsc.edu

Tutoring Times:

- Monday: 100pm – 2:00pm
- Tuesday: 2:00pm – 3:00pm
- Wednesday: 1pm – 2:00pm

Tutoring will be in Jack's Lounge, on the first floor of Baskin Engineering.

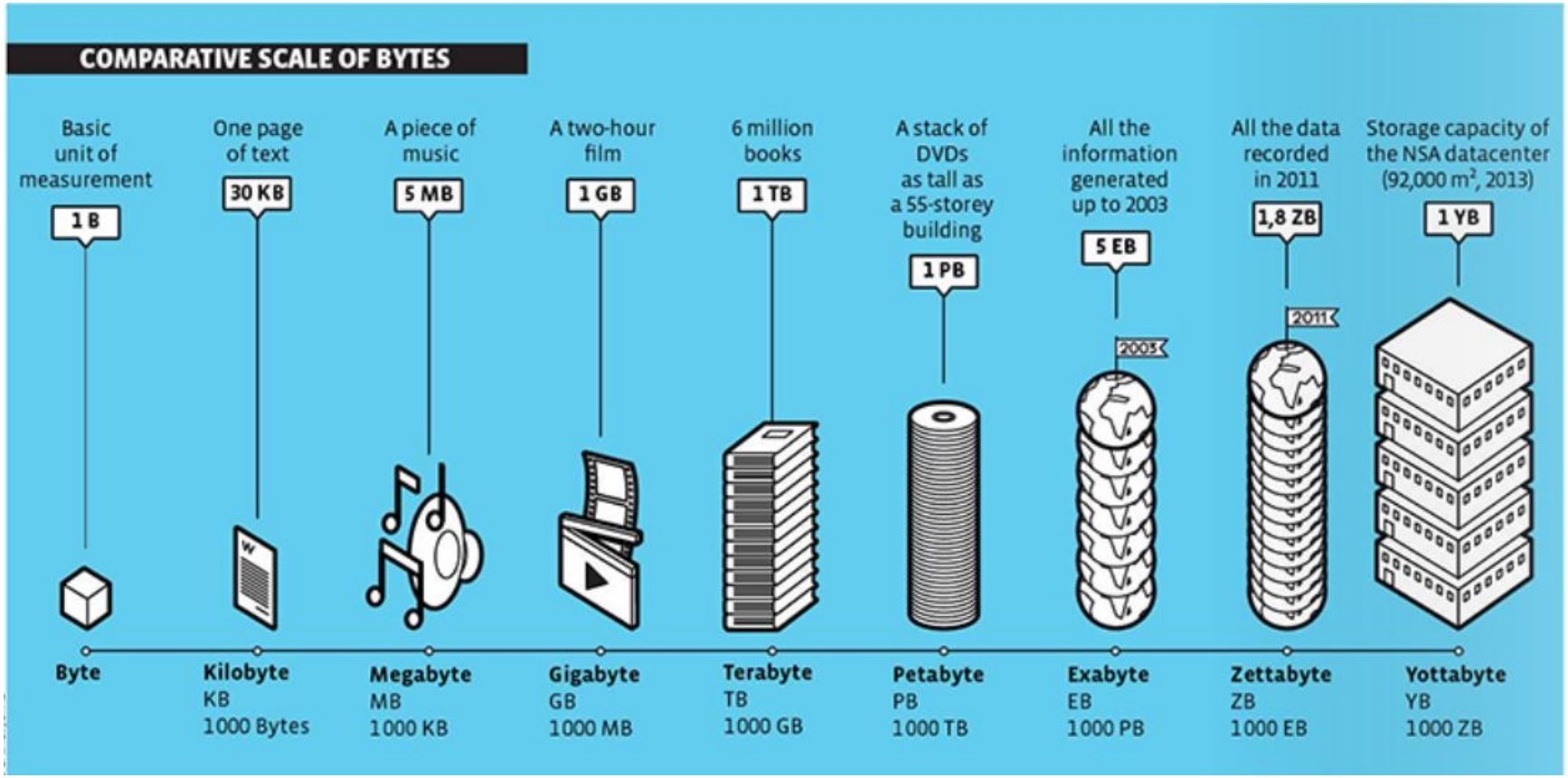
From Bytes to Yottabytes

Multiples of bytes					v · t · e
SI decimal prefixes		Binary usage	IEC binary prefixes		
Name (Symbol)	Value		Name (Symbol)	Value	
kilobyte (kB)	10^3	2^{10}	kibibyte (KiB)	2^{10}	
megabyte (MB)	10^6	2^{20}	mebibyte (MiB)	2^{20}	
gigabyte (GB)	10^9	2^{30}	gibibyte (GiB)	2^{30}	
terabyte (TB)	10^{12}	2^{40}	tebibyte (TiB)	2^{40}	
petabyte (PB)	10^{15}	2^{50}	pebibyte (PiB)	2^{50}	
exabyte (EB)	10^{18}	2^{60}	exbibyte (EiB)	2^{60}	
zettabyte (ZB)	10^{21}	2^{70}	zebibyte (ZiB)	2^{70}	
yottabyte (YB)	10^{24}	2^{80}	yobibyte (YiB)	2^{80}	
See also: Multiples of bits · Orders of magnitude of data					

From Bytes to Yottabytes

Multiples of bytes					v · t · e
SI decimal prefixes		Binary usage	IEC binary prefixes		
Name (Symbol)	Value		Name (Symbol)	Value	
kilobyte (kB)	10^3	2^{10}	kibibyte (KiB)	2^{10}	
megabyte (MB)	10^6	2^{20}	mebibyte (MiB)	2^{20}	
gigabyte (GB)	10^9	2^{30}	gibibyte (GiB)	2^{30}	
terabyte (TB)	10^{12}	2^{40}	tebibyte (TiB)	2^{40}	
petabyte (PB)	10^{15}	2^{50}	pebibyte (PiB)	2^{50}	
exabyte (EB)	10^{18}	2^{60}	exbibyte (EiB)	2^{60}	
zettabyte (ZB)	10^{21}	2^{70}	zebibyte (ZiB)	2^{70}	
yottabyte (YB)	10^{24}	2^{80}	yobibyte (YiB)	2^{80}	
See also: Multiples of bits · Orders of magnitude of data					

Comparative Size of Bytes – credit to Testyotta



Some Supplementary Reading Material

- Database management systems:
  http://en.wikipedia.org/wiki/Database_management_system

- Fifty years of databases:
  http://wp.sigmod.org/?p=688

- IDC White Paper (sponsored by Seagate):
  "Data Age 2025: The Evolution of Data to Life-Critical"
  https://www.seagate.com/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf

  Estimates that in 2025, the world will create and replicate 163 ZB of data, representing a tenfold increase from the amount of data created in 2016.

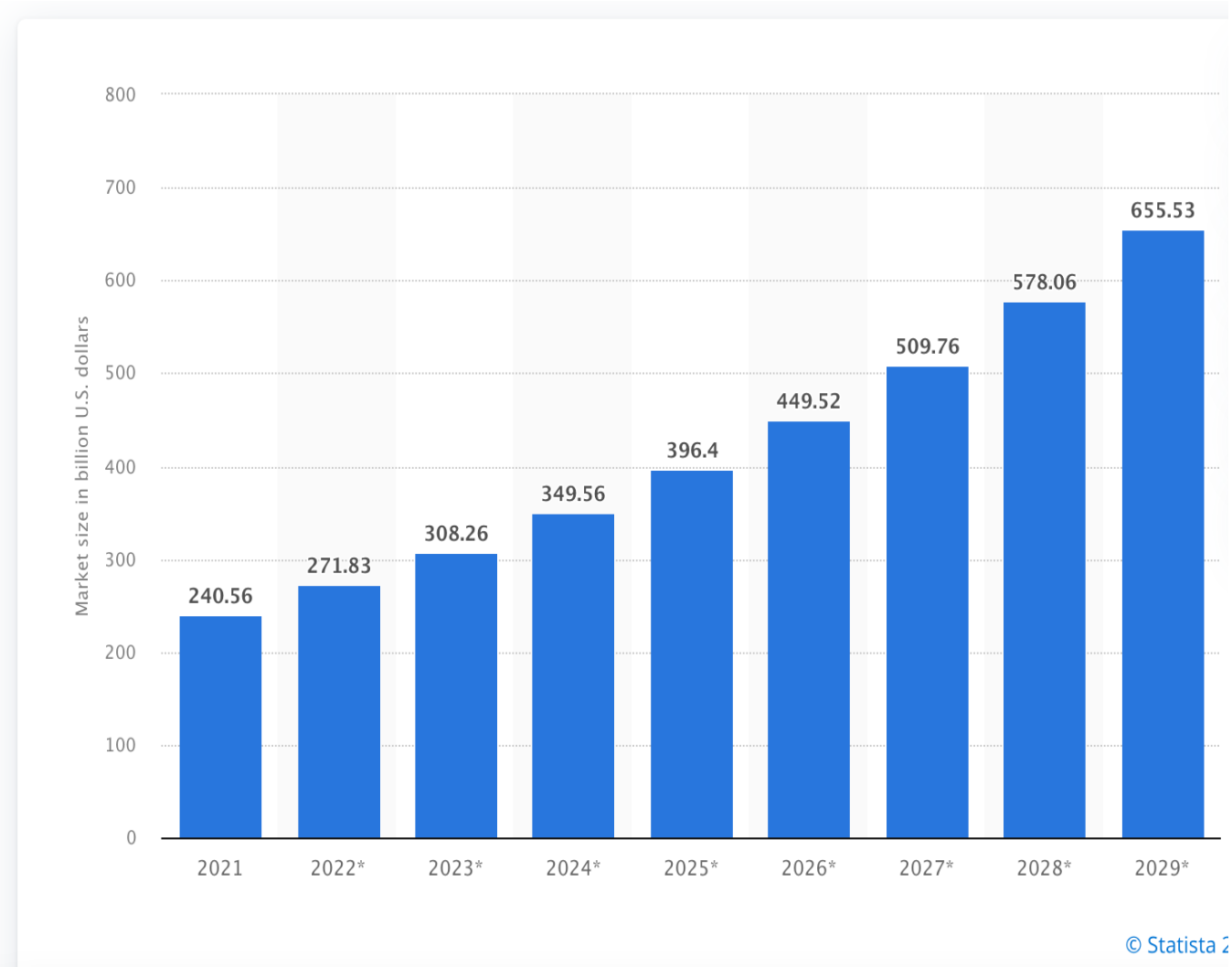
Big Time Big Data Statistics

Source: 25+ Impressive Big Data Statistics for 2023

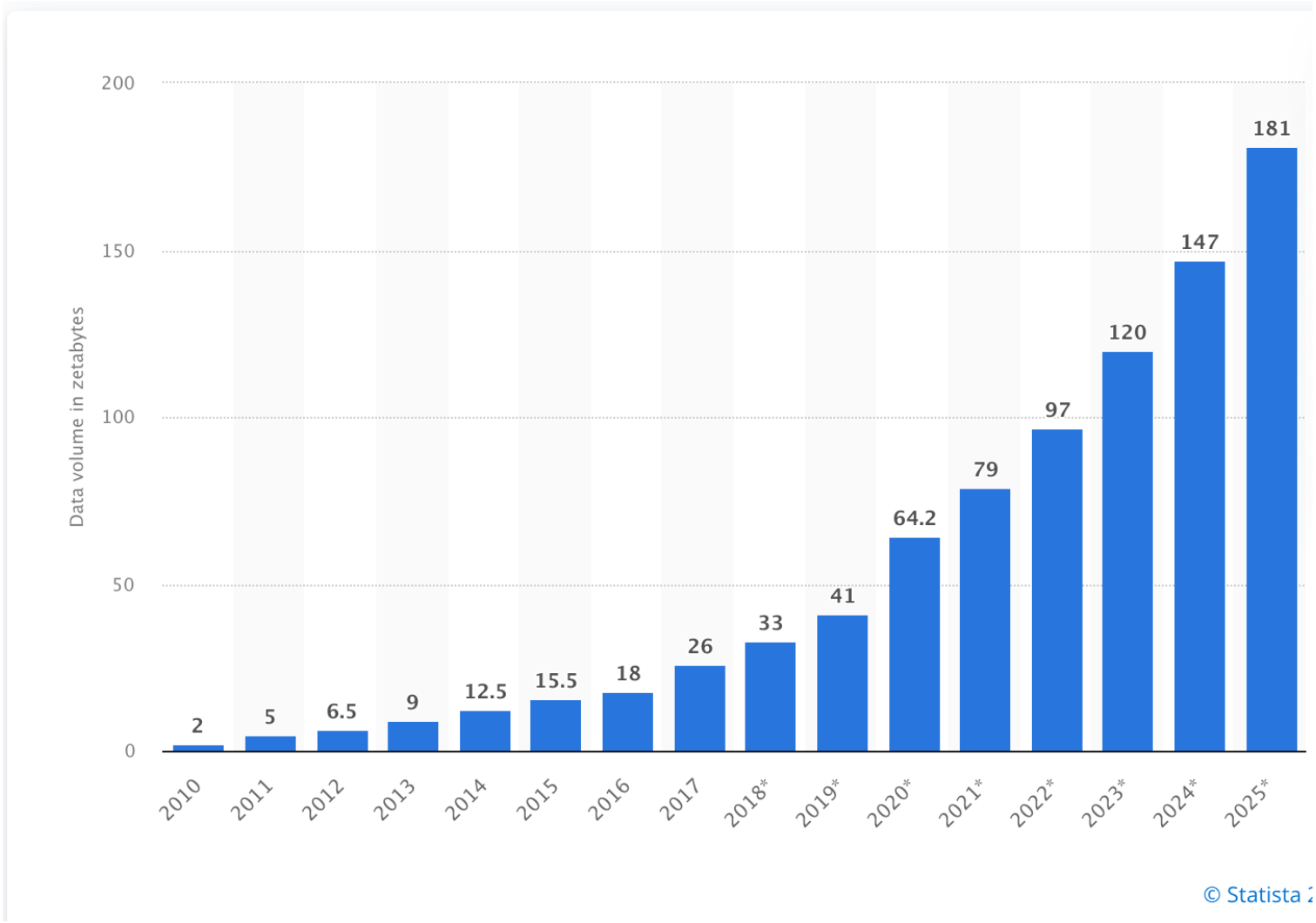
- The big data analytics market is set to reach $103 billion by 2023.
- Poor data quality costs the US economy up to $3.1 trillion yearly.
- In 2020, every person generated 1.7 megabytes in just a second.
- Internet users generate about 2.5 quintillion bytes of data each day.
- 95% of businesses cite the need to manage unstructured data as a problem for their business.
- 97.2% of organizations are investing in big data and AI.
- Using big data, Netflix saves $1 billion per year on customer retention.
- Predictions estimate the world will generate 181 zettabytes of data by 2025.

Size of the big data analytics market worldwide from 2021 to 2029

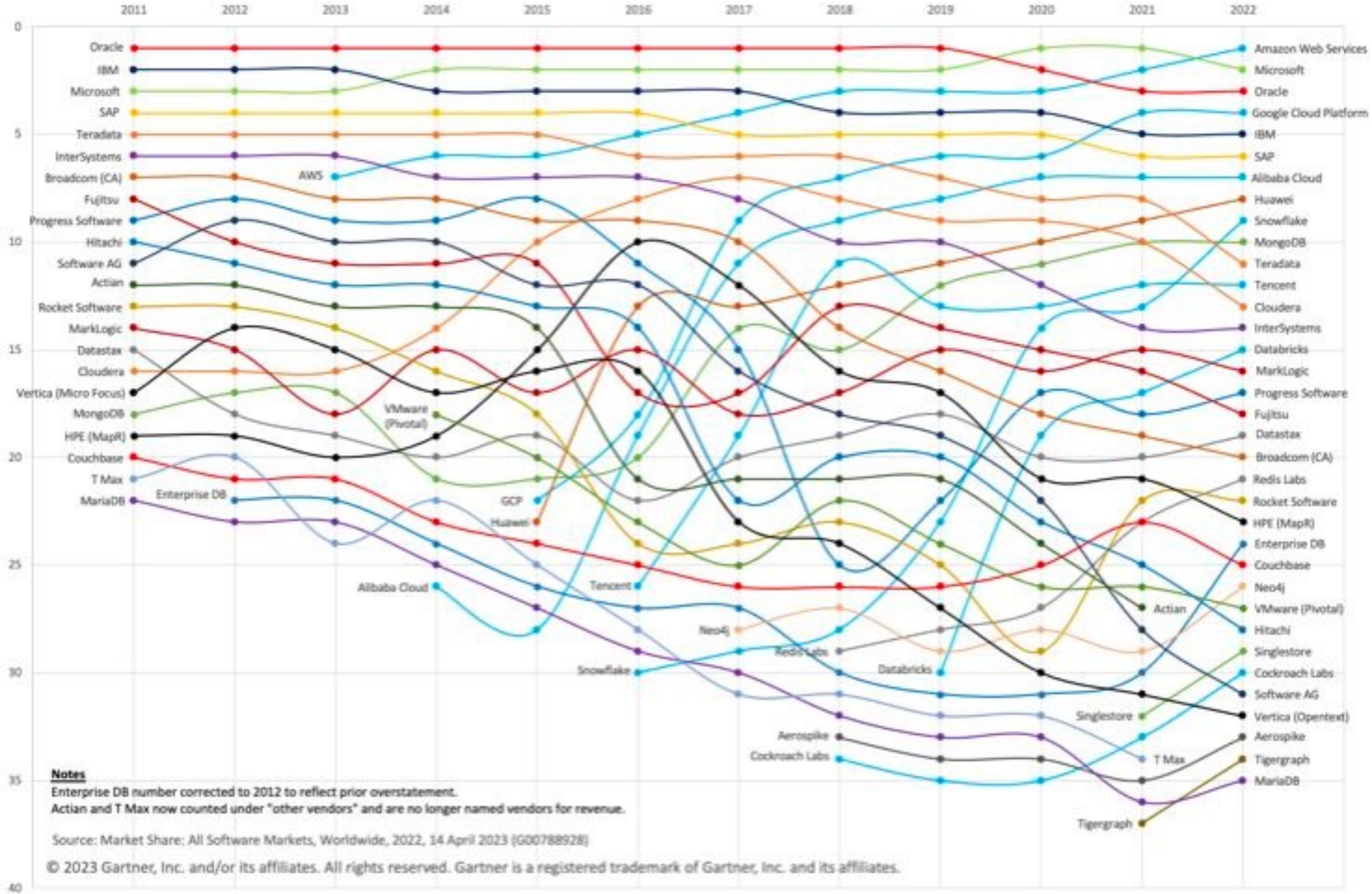
(in billion U.S. dollars)



Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025 *(in zettabytes)*



Gartner DBMS Market Share Ranks: 2011-2022



Notes

Enterprise DB number corrected to 2012 to reflect prior overstatement.

Actian and T Max now counted under "other vendors" and are no longer named vendors for revenue.

Source: Market Share: All Software Markets, Worldwide, 2022, 14 April 2023 [G00788928]

© 2023 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. and its affiliates.

DB-Engines Popularity Rating for RDBMS

Rank			DBMS	Database Model	Score		
Oct 2023	Sep 2023	Oct 2022			Oct 2023	Sep 2023	Oct 2022
1.	1.	1.	Oracle	Relational, Multi-model	1261.42	+20.54	+25.05
2.	2.	2.	MySQL	Relational, Multi-model	1133.32	+21.83	-72.06
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	896.88	-5.34	-27.80
4.	4.	4.	PostgreSQL	Relational, Multi-model	638.82	+18.06	+16.10
5.	5.	5.	IBM Db2	Relational, Multi-model	134.87	-1.85	-14.79
6.	6.	↑ 7.	SQLite	Relational	125.14	-4.06	-12.66
7.	7.	↓ 6.	Microsoft Access	Relational	124.31	-4.25	-13.85
8.	8.	↑ 9.	Snowflake	Relational	123.24	+2.35	+16.51
9.	9.	↓ 8.	MariaDB	Relational, Multi-model	99.66	-0.79	-9.65
10.	10.	10.	Microsoft Azure SQL Database	Relational, Multi-model	80.93	-1.80	-4.03
11.	11.	↑ 13.	Databricks	Multi-model	75.82	+0.64	+18.21
12.	12.	↓ 11.	Hive	Relational	69.18	-2.65	-11.42
13.	13.	↓ 12.	Teradata	Relational, Multi-model	58.56	-1.77	-7.51
14.	14.	14.	Google BigQuery	Relational	56.57	+0.11	+4.12
15.	15.	15.	FileMaker	Relational	53.32	-0.29	+0.91
16.	16.	16.	SAP HANA	Relational, Multi-model	49.44	-1.17	-2.63
17.	17.	17.	SAP Adaptive Server	Relational, Multi-model	41.92	-1.40	-1.05
18.	18.	↑ 20.	Microsoft Azure Synapse Analytics	Relational	27.13	-0.60	+4.03
19.	19.	19.	Firebird	Relational	25.68	+0.05	+0.68
20.	20.	↑ 21.	Informix	Relational, Multi-model	21.42	-0.92	-1.45
21.	21.	↓ 18.	Amazon Redshift	Relational	21.04	+1.00	-5.98
22.	↑ 23.	22.	Spark SQL	Relational	18.74	-0.47	-3.85
23.	↓ 22.	↑ 25.	Impala	Relational, Multi-model	18.51	-1.09	+0.83
24.	24.	↑ 28.	ClickHouse	Relational, Multi-model	15.13	-0.06	+1.26
25.	25.	↑ 27.	Presto	Relational	14.22	-0.63	-0.32
26.	↑ 29.		Apache Flink	Relational	13.90	-0.02	
27.	27.	↓ 23.	Vertica	Relational, Multi-model	13.88	-0.47	-5.97
28.	↓ 26.	↓ 26.	dBASE	Relational	13.84	-0.79	-1.46
29.	↓ 28.	↓ 24.	Netezza	Relational	13.78	-0.49	-4.06
30.	30.	30.	Greenplum	Relational, Multi-model	10.45	-0.16	-2.06
31.	31.	↓ 29.	Amazon Aurora	Relational, Multi-model	9.15	+0.03	-3.82
32.	32.	↓ 31.	H2	Relational, Multi-model	8.61	-0.17	-0.95
33.	33.	↓ 32.	Oracle Essbase	Relational	8.49	+0.17	-0.28
34.	↑ 35.	↑ 41.	Derby	Relational	7.06	+0.22	+1.40
35.	↓ 34.	↓ 33.	CockroachDB	Relational	6.60	-0.30	-1.36
36.	36.	↓ 34.	Microsoft Azure Data Explorer	Relational, Multi-model	6.55	-0.10	-1.12
37.	↑ 38.	37.	Interbase	Relational	6.00	-0.06	-0.11
38.	↓ 37.	↑ 49.	Trino	Relational, Multi-model	5.98	-0.09	+2.13
39.	39.	↓ 35.	SingleStore	Relational, Multi-model	5.78	+0.11	-1.76
40.	40.	↓ 38.	SAP SQL Anywhere	Relational	5.43	-0.17	-0.40

DB-Engines Popularity Rating for DB-Engines

Rank Oct 2023	Rank Sep 2023	Rank Oct 2022	DBMS	Database Model	Score Oct 2023	Score Sep 2023	Score Oct 2022
1.	1.	1.	Oracle	Relational, Multi-model	1261.42	+20.54	+25.05
2.	2.	2.	MySQL	Relational, Multi-model	1133.32	+21.83	-72.06
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	896.88	-5.34	-27.80
4.	4.	4.	PostgreSQL	Relational, Multi-model	638.82	+18.06	+16.10
5.	5.	5.	MongoDB	Document, Multi-model	431.42	-8.00	-54.81
6.	6.	6.	Redis	Key-value, Multi-model	162.96	-0.72	-20.41
7.	7.	7.	Elasticsearch	Search engine, Multi-model	137.15	-1.84	-13.92
8.	8.	8.	IBM Db2	Relational, Multi-model	134.87	-1.85	-14.79
9.	9.	↑ 10.	SQLite	Relational	125.14	-4.06	-12.66
10.	10.	↓ 9.	Microsoft Access	Relational	124.31	-4.25	-13.85
11.	11.	↑ 13.	Snowflake	Relational	123.24	+2.35	+16.51
12.	12.	↓ 11.	Cassandra	Wide column, Multi-model	108.82	-1.24	-9.12
13.	13.	↓ 12.	MariaDB	Relational, Multi-model	99.66	-0.79	-9.65
14.	14.	14.	Splunk	Search engine	92.37	+0.98	-2.28
15.	15.	↑ 16.	Microsoft Azure SQL Database	Relational, Multi-model	80.93	-1.80	-4.03
16.	16.	↓ 15.	Amazon DynamoDB	Multi-model	80.91	+0.00	-7.44
17.	17.	↑ 20.	Databricks	Multi-model	75.82	+0.64	+18.21
18.	18.	↓ 17.	Hive	Relational	69.18	-2.65	-11.42
19.	19.	↓ 18.	Teradata	Relational, Multi-model	58.56	-1.77	-7.51
20.	20.	↑ 22.	Google BigQuery	Relational	56.57	+0.11	+4.12
21.	21.	↑ 23.	FileMaker	Relational	53.32	-0.29	+0.91
22.	22.	↑ 24.	SAP HANA	Relational, Multi-model	49.44	-1.17	-2.63
23.	23.	↓ 19.	Neo4j	Graph	48.44	-1.95	-10.25
24.	24.	↓ 21.	Solr	Search engine, Multi-model	45.36	-1.73	-8.14
25.	25.	25.	SAP Adaptive Server	Relational, Multi-model	41.92	-1.40	-1.05
26.	26.	26.	HBase	Wide column	34.69	-1.28	-6.97
27.	27.	27.	Microsoft Azure Cosmos DB	Multi-model	34.29	-1.16	-6.12
28.	28.	↑ 29.	InfluxDB	Time Series, Multi-model	29.74	-1.53	+0.16
29.	29.	↓ 28.	PostGIS	Spatial DBMS, Multi-model	27.30	-1.10	-3.55
30.	30.	↑ 34.	Microsoft Azure Synapse Analytics	Relational	27.13	-0.60	+4.03
31.	31.	↑ 33.	Firebird	Relational	25.68	+0.05	+0.68
32.	32.	↓ 30.	Couchbase	Document, Multi-model	22.41	-1.36	-5.54
33.	33.	↑ 35.	Informix	Relational, Multi-model	21.42	-0.92	-1.45
34.	↑ 35.	↓ 31.	Amazon Redshift	Relational	21.04	+1.00	-5.98
35.	↓ 34.	↓ 32.	Memcached	Key-value	20.84	-0.47	-4.55
36.	↑ 37.	36.	Spark SQL	Relational	18.74	-0.47	-3.85
37.	↓ 36.	↑ 40.	Impala	Relational, Multi-model	18.51	-1.09	+0.83
38.	38.	↓ 37.	Firebase Realtime Database	Document	17.48	-0.10	-2.64
39.	39.	↑ 44.	ClickHouse	Relational, Multi-model	15.13	-0.06	+1.26
40.	40.	↑ 43.	Presto	Relational	14.22	-0.63	-0.32

Meaning of an SQL Query with Multiple Relations in the FROM Clause

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[ORDER BY < list of attributes [ASC | DESC] >]

Suppose we now have more than 1 relation in the FROM clause.

- Let Result begin as an empty multiset of tuples.
- For every tuple t_1 from R_1, t_2 from R_2, …, t_n from R_n
  - if t_1, …, t_n satisfy *condition (*i.e., condition evaluates to true), then add the resulting tuple that consists of c_1, c_2, …, c_m components of *t* into Result.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- If ORDER BY <list of attributes> exists, order the tuples in Result according to ORDER BY clause.
- Return Result.

Database Schema for Some Examples

- Let us assume we have the following database schema with five relation schemas, with Primary Keys underlined.

  Movies(movieTitle, movieYear, length, genre, studioName, producerC#)

  StarsIn(movieTitle, movieYear, starName)

  MovieStar(starName, address, gender, birthdate)

  MovieExec(execName, address, cert#, netWorth)

  Studio(studioName, address, presC#)

We'll temporarily assume that the attributes of Movies are title and year, rather than movieTitle and movieYear.

Cartesian Product in SQL

```
SELECT *
FROM Movies, StarsIn;
```

Movies

title	year	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster's Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

StarsIn

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

Join in SQL

```
SELECT *
FROM Movies, StarsIn
WHERE title = movieTitle AND year = movieYear;
```

Movies

title	year	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster's Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

StarsIn

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

Disambiguating Attributes

```
SELECT *
FROM Movies, StarsIn
WHERE title = 'Pretty Woman' AND movieYear <= 1995;
```

Movies

title	year	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster's Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

StarsIn

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

Disambiguating Attributes

```
SELECT *
FROM Movies, StarsIn
WHERE movieTitle = 'Pretty Woman' AND movieYear <= 1995;
```

Movies

movieTitle	year	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster's Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

StarsIn

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

But what if the first attribute of Movies was also called “movieTitle”?

Disambiguating Attributes (cont'd)

```
SELECT *
FROM Movies, StarsIn
WHERE Movies.movieTitle = 'Pretty Woman' AND movieYear <= 1995;
```

Movies

movieTitle	year	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster's Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

StarsIn

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

Tuple Variables

What if the title attribute of Movies was also called movieTitle?

```
SELECT *
FROM Movies, StarsIn
WHERE movieTitle = title;
```

```
SELECT *
FROM Movies m, StarsIn s
WHERE m.movieTitle = s.movieTitle;
```

m and s are tuple variables.

- m binds to a tuple (row) in the Movies relation.
- s binds to a tuple (row) in StarsIn relation.
- *Could also write Movies.movieTitle = StarsIn.movieTitle, and not bother having the tuple variables m and s.*
- But we'll frequently use tuple variables for clarity and brevity.

Self Joins

```
SELECT *
FROM StarsIn s1, StarsIn s2
WHERE s1.movieYear = s2.movieYear;
```

StarsIn

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

Database Schema for Some Examples

- Let us assume we have the following database schema with five relation schemas, with primary keys underlined.
  - The Primary Key of Movies is once again (movieTitle, movieYear).

  Movies(movieTitle, movieYear, length, genre, studioName, producerC#)
  StarsIn(movieTitle, movieYear, starName)
  MovieStar(starName, address, gender, birthdate)
  MovieExec(execName, address, cert#, netWorth)
  Studio(studioName, address, presC#)

- The English query descriptions that are on the next slide don't clearly specify which attributes should appear in query results, nor do they say whether duplicates should be eliminated. (Yes, they should be.)
  - But questions on Assignments and Tests will always specify that.

Some Example Join Queries (We'll Write Answers on the Whiteboard)

1. Who were the male stars in the 2022 movie Avatar?
2. Which stars appeared in movies produced by MGM in 2019?
3. Who is the president of MGM?
4. Which movies are longer than the 2022 movie Avatar?

Solution for "Avatar" queries will be posted on Piazza under Resources→General Resources after we finished writing them in class.

For some queries, we provide more than one correct solution.

You may want to use ChatGPT to test your answers (and our answers) for these queries!

SQL Join Expressions

Reference:
A First Course in Database Systems, 3rd edition, Chapter 6.3.6 – 6.3.8

- JOIN
  - ON
  - CROSS JOIN
- NATURAL JOIN
- Outer Joins
  - FULL OUTER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
- All of these can appear in the FROM clause
  - OUTER JOIN will be explained later in the course.
  - NATURAL JOIN will reappear when we discuss Design Theory.

JOIN … ON …

```
SELECT *
FROM R JOIN S
    ON R.A = S.D AND R.B = S.E
WHERE R.C = 100 AND S.F < 50;
```

R(A,B,C) and S(D,E,F)

- FROM R **JOIN** S **ON** R.A=S.D AND R.B=S.E …
  - Selects only tuples from R and S which satisfy R.A=S.C and R.B=S.D
  - After ON, specify conditions joining tables R and S.
  - WHERE clause can have other conditions in it which aren't JOIN conditions.
  - Schema of the resulting relation: (R.A, R.B, R.C, S.C, S.D, S.E);
  - Equivalent to:

```
SELECT *
FROM R, S
WHERE R.A = S.E AND R.B = S.F
   AND R.C = 100 AND S.F < 50;
```

Advantage of JOIN … ON … Is that it makes it obvious how tables are being joined, usually on Primary Key to Foreign Key.

We will sometimes use JOIN … ON … in class, but you can always use it (if correct) on homeworks and exams.

SELECT clause specifies which attributes are projected in the result.

Avatar Queries Written Using JOIN ON were also posted on Piazza

1. Who were the male stars in the 2022 movie Avatar?
2. Which stars appeared in movies produced by MGM in 2019?
3. Who is the president of MGM?
4. Which movies are longer than the 2022 movie Avatar?

SQL for "Avatar" queries using JOIN ON will be (or was) also posted on Piazza as a file under Resources→General Resources.

CROSS JOIN

R(A,B,C) and S(C,D,E)

- R **CROSS JOIN** S;
  - Product of the two relations R and S.
  - Schema of resulting relation: (R.A, R.B, R.C, S.C, S.D, S.E).
  - Equivalent to:

```
SELECT *
FROM R, S;
```

Nobody uses CROSS JOIN.

NATURAL JOIN

R(A,B,C) and S(C,D,E)

- R **NATURAL JOIN** S;
  - Schema of the resulting relation: (A, B, C, D, E)
  - Equivalent to:

```
SELECT R.A, R.B, R.C, S.D, S.E
FROM R, S
WHERE R.C = S.C;
```

  - Note that attribute C appears only once!

Why is this called *Natural Join*?

Do you think that it's *Natural*?

Why can *Natural Join* be dangerous?

We'll say more about *Natural Join* later in the quarter, when we discuss Design Theory.

Set and Bag Operations in SQL

- Set Union, Set Intersection, Set Difference
- Bag Union, Bag Intersection, Bag Difference

- Other set/bag operations
  - IN, NOT IN, op ANY, op ALL, EXISTS, NOT EXISTS
  - More on these operations later in this lecture

Reference:
A First Course in Database Systems,
3rd edition, Chapter 6.2.5, 6.4.1, 6.4.2

Set Union

R(A,B,C), S(A,B,C)

- Input to union must be *union-compatible*.
  - R and S must have attributes of the same types in the same order.
- Output of Union has the same schema as R or S.
- Meaning: Output consists of the **set** of all tuples from R and from S.
  - UNION could (should??) have been called UNION DISTINCT

```
( SELECT *
  FROM R )
UNION
( SELECT *
  FROM S );
```

```
( SELECT *
  FROM R
  WHERE A > 10 )
UNION
( SELECT *
  FROM S
  WHERE B < 300 );
```

Bag Union

R(A,B,C), S(A,B,C)

- Input to union must be *union-compatible*.
  - R and S must have attributes of the same types in the same order.
- Output of union has the same schema as R or S.
- Meaning: Output consists of the collection of all tuples from R and from S, including duplicate tuples.
  - *Subtlety: Attributes/column names may be different; R's are used.*

```
( SELECT *
  FROM R )
UNION ALL
( SELECT *
  FROM S );
```

```
( SELECT *
  FROM R
  WHERE A > 10)
UNION ALL
( SELECT *
  FROM S
  WHERE B < 300 );
```

Are These Two Queries Always Equivalent?

That is, Do They Always Have Same Result?

Assume that R(A,B,C) and S(A,B,C) are Union-Compatible

```
( SELECT DISTINCT *
  FROM R
  WHERE A > 10)

UNION ALL

( SELECT DISTINCT *
  FROM S
  WHERE B < 300);
```

```
( SELECT *
  FROM R
  WHERE A > 10 )

UNION

( SELECT *
  FROM S
  WHERE B < 300 );
```

Intersection; Difference (Except)

- Like Union, Intersection and Difference are binary operators.
  - Input to Intersection/Difference operator consists of two relations R and S, and they must be union-compatible.
  - Output has the same type as R or S.

- Set Intersection, Bag Intersection
  - <Query1> **INTERSECT** <Query2>, <Query1> **INTERSECT ALL** <Query2>
  - Find all tuples that are in the results of both Query1 and Query2.

- Set Difference, Bag Difference
  - <Query1> **EXCEPT** <Query2>, <Query1> **EXCEPT ALL** <Query2>
  - Find all tuples that are in the result of Query1, but not in the result of Query2.

UNION, INTERSECT, EXCEPT

IF R has **m** copies of a particular tuple, and S has **n** copies of that exact same tuple, then how many copies of that tuple are there in the result of each of the Set Operations on R and S?

	DISTINCT	ALL
R UNION S	MIN(m+n, 1)	m+n
R INTERSECT S	MIN(MIN(m,n,), 1)	MIN(m,n)
R EXCEPT S	IF m>0 AND n=0 THEN 1 ELSE 0	MAX(m-n, 0)

Reminder: UNION is the same as UNION DISTINCT, and that's also true for INTERSECT and EXCEPT.

Operator Precedence

- Query1 EXCEPT Query2 EXCEPT Query3 means
  (Query1 EXCEPT Query2) EXCEPT Query3

Order of operations originally was: UNION, INTERSECT and EXCEPT have the same priority, so operations were executed left-to-right.

But this changed in the SQL standard, and has changed in most implementations!
Now, INTERSECT has a higher priority than UNION and EXCEPT
(just as * has a higher priority than + and -), so:

Query1 UNION Query2 INTERSECT Query3
would be executed as:
Query1 UNION (Query2 INTERSECT Query3)
not as:
(Query1 UNION Query2) INTERSECT Query3

Subqueries

Reference:
A First Course in Database Systems, 3rd edition, Chapter 6.3.

- A subquery is a query that is embedded in another query.
  - Note that queries with UNION, INTERSECT, and EXCEPT have two subqueries.
- A subquery can return a constant (scalar value) which can be compared against another constant in the WHERE clause, or can be used in a boolean expression.
- A subquery can also return a relation.
- A subquery returning a relation can appear in the FROM clause, followed by a tuple variable that refers to the tuples in the result of the subquery
  - … just as a tuple variable can be used to refer to a table.

Subqueries that Return Scalar Values

Movies(movieTitle, movieYear, length, genre, studioName, producerC#)
MovieExec(execName, address, cert#, netWorth), **with execName UNIQUE**

- Find names of executives who produced the movie 'Star Wars'.
  - Careful: Don't write query the second way--possible runtime error!

```
SELECT e.execName
FROM Movies m, MovieExec e
WHERE m.movieTitle='Star Wars' AND m.producerC# = e.cert#;
```

```
SELECT e.execName
FROM MovieExec e
WHERE e.cert# = (SELECT m.producerC#
                 FROM Movies m
                 WHERE m.movieTitle = 'Star Wars');
```

Outer query

Inner query

Could write this correctly using either "= ANY" or "IN", as we'll soon learn.

Subqueries that Return Relations

```
SELECT e.execName
FROM MovieExec e
WHERE e.cert#  IN (SELECT m.producerC#
                   FROM Movies m
                   WHERE m.movieTitle = 'Star Wars');
```

- **IN, NOT IN**

Is this query **equivalent** to the one above?

```
SELECT e.execName
FROM MovieExec e, Movies m
WHERE m.movieTitle = 'Star Wars'
    AND m.producerC# = e.cert#;
```

Putting a Subquery that Returns a Relation into the FROM Clause

```
SELECT e.execName
FROM MovieExec e,
                    (SELECT m.producerC#
                     FROM Movies m
                     WHERE m.movieTitle = 'Star Wars') p
WHERE e.cert# = p.producerC# ;
```

Is this query equivalent to the one above?

```
SELECT e.execName
FROM MovieExec e, Movies m
WHERE e.cert# = m.producerC#
    AND m.movieTitle = 'Star Wars';
```

Subqueries with Subqueries

What does this query do? **(Assume that MovieExec.execName is UNIQUE.)**

```
SELECT e.execName
FROM MovieExec e
WHERE e.cert#  IN (SELECT m.producerC#
                   FROM Movies m
                   WHERE (m.movieTitle, m.movieYear) IN (SELECT s.movieTitle, s.movieYear
                                                         FROM StarsIn s
                                                         WHERE s.starName = 'Harrison Ford')
                  );
```

Is this query equivalent? (Tuple variables help make queries clearer.)

```
SELECT e.execName
FROM MovieExec e, Movies m, StarsIn s
WHERE e.cert# = m.producerC# AND m.movieTitle = s.movieTitle
    AND m.movieYear = s.movieYear AND s.starName = 'Harrison Ford';
```

Rewriting Bottom Query Using JOIN … ON

Is this query equivalent?

```
SELECT e.execName
FROM MovieExec e
   JOIN Movies m ON e.cert# = m.producerC#
   JOIN StarsIn s ON (m.movieTitle, m.movieYear) = (s.movieTitle, s.movieYear)
WHERE s.starName = 'Harrison Ford';
```

Correlated Subqueries

- In all the examples so far, the inner query has been independent of the outer query.
- An inner query can also depend on attributes in the outer query; that's called **correlation**.
- Find the movie titles that have been used for two or more movies.

```
SELECT DISTINCT m.movieTitle
FROM Movies m
WHERE m.movieYear < ANY (SELECT m2.movieYear
                         FROM Movies m2
                         WHERE m2.movieTitle = m.movieTitle);
```

DISTINCT needed, because movieTitle may have been used for three or more movies!

Correlation via tuple variable m

Checks that year is less than at least one of the answers returned by the subquery.
< ANY, <= ANY, > ANY, >= ANY, <> ANY, = ANY
< ALL, <= ALL, > ALL, >= ALL, <> ALL, = ALL

Correlated Subqueries (cont'd)

```
SELECT s.starName
FROM StarsIn s
WHERE EXISTS ( SELECT *
                 FROM StarsIn c
                 WHERE s.movieTitle <> c.movieTitle
                     AND s.movieYear = c.movieYear
                     AND s.starName = c.starName
             );
```

Checks that the subquery returns a non-empty result.
Can write EXISTS or NOT EXISTS.

Set Comparison Operators

- x IN Q
  - Returns true if x occurs in the collection Q.
- x NOT IN Q
  - Returns true if x does not occur in the collection Q.
- EXISTS Q
  - Returns true if Q is a non-empty collection.
- NOT EXISTS Q
  - Returns true if Q is an empty collection.

Set Comparison Operators (cont'd)

- *x op* ANY Q, x *op* ALL Q in WHERE clause
  - x is a scalar expression
  - Q is a SQL query
  - *op* is one of { <, <=, >, >=, <>, = }.

- x *op* ANY Q
  - What does this mean?
  - SOME can be used instead of ANY

- x op ALL Q
  - What does this mean?

Reminder: Subqueries Can Appear in the FROM Clause

- Find the names of all movie executives who produced movies in which Harrison Ford starred.

```
SELECT e.execName
FROM MovieExec e, ( SELECT m.producerC#
                    FROM Movies m, StarsIn s
                    WHERE (m.movieTitle, m.movieYear)
                          = (s.movieTitle, s.movieYear)
                      AND s.starName = 'Harrison Ford') p
WHERE e.cert# = p.producerC#;
```

Yes, this could be written using JOIN … ON …

Movies(movieTitle, movieYear, length, genre, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieExec(execName, address, cert#, netWorth)

Ski Activity Examples

Customers

cid	cname	level	type	age
36	Cho	Beginner	snowboard	18
34	Luke	Inter	snowboard	25
87	Ice	Advanced	ski	20
39	Paul	Beginner	ski	33

Activities

cid	slopeid	day
36	s3	2021-01-05
36	s1	2021-01-06
36	s1	2021-01-07
87	s2	2021-01-07
87	s1	2021-01-07
34	s2	2021-01-05

Slopes

slopeid	name	color
s1	Mountain Run	blue
s2	Olympic Lady	black
s3	Magic Carpet	green
s4	KT-22	black

Example 1

- Find the id and name of each customer who did some activity on 01/07/21.

Please try to write query yourselves.

Example 1

- Find the id and name of each customer who did some activity on 01/07/21.

```
SELECT c.cid, c.cname
FROM Customers c, Activities a
WHERE a.day = DATE '2021-01-07' AND a.cid = c.cid;
```

Wrong! Result has duplicates

```
SELECT c.cid, c.cname
FROM Customers c
WHERE c.cid IN ( SELECT a.cid
                 FROM Activities a
                 WHERE a.day = DATE '2021-01-07');
```

Example 1 (Equivalent, using EXISTS?)

- Find the id and name of each customer who did some activity on the day 01/07/21.

```
SELECT c.cid, c.cname
FROM Customers c
WHERE EXISTS ( SELECT *
                    FROM Activities a
                    WHERE a.cid = c.cid AND a.day = DATE '2021-01-07' );
```

Is this equivalent to the second query on previous slide?

Example 2

- Find the id and name of each customer who did not do any activity on 01/07/21.

```
SELECT c.cid, c.cname
FROM Customers c
WHERE c.cid NOT IN ( SELECT a.cid
                     FROM Activities a
                     WHERE a.day = DATE '2021-01-07' );

SELECT c.cid, c.cname
FROM Customers c
WHERE NOT EXISTS ( SELECT *
                   FROM Activities a
                   WHERE a.cid = c.cid AND a.day = DATE '2021-01-07' );
```

Example 3

- Find the id and name of each customer who went on the slope “Olympic Lady” on 01/07/21.

```
SELECT c.cid, c.cname
FROM   Customers c
WHERE c.cid IN ( SELECT a.cid
                 FROM Activities a
                 WHERE a.day = DATE '2021-01-07' AND a.slopeid IN ( SELECT s.slopeid
                                                                    FROM Slopes s
                                                                    WHERE s.name='Olympic Lady' )
               );
```

Could you also rewrite this as a join, with SELECT DISTINCT?

Example 3 (Equivalent?)

- Find the id and name of each customer who went on the slope “Olympic Lady” on 01/07/21.

```
SELECT DISTINCT c.cid, c.cname
FROM   Customers c, Activities a, Slopes s
WHERE c.cid = a.cid
   AND a.day = DATE '2021-01-07'
   AND s.name='Olympic Lady'
   AND a.slopeid = s.slopeid;
```

```
SELECT DISTINCT c.cid, c.cname
FROM   Customers c
   JOIN Activities a ON c.cid = a.cid
   JOIN Slopes s ON a.slopeid = s.slopeid
WHERE a.day = DATE '2021-01-07'
   AND s.name='Olympic Lady';
```

Is this equivalent to the query on previous slide?

Example 4

- Determine the colors of all slopes that Cho went on.

Please try to write query yourselves.

Example 4

- Determine the colors of all slopes that Cho went on.

```
SELECT DISTINCT s.color
FROM  Activities a, Slopes s
WHERE a.slopeid = s.slopeid AND
          a.cid = ( SELECT c.cid
                    FROM Customers c
                    WHERE c.cname='Cho' );
```

  - Would this be correct, if cname **is UNIQUE**?
  - Would this be correct, if cname **is not UNIQUE**?
  - How might you fix this, if cname is not UNIQUE?

Example 5

- Find the names of all customers who did some activity.

```
SELECT c.cname
FROM Customers c
WHERE c.cid = ANY ( SELECT a.cid
                    FROM Activities a );
```

Example 6

- Find the names of all customers who are skiers and whose age is greater than the age of every snowboarder.

```
SELECT c.cname
FROM Customers c
WHERE c.type='skier' AND c.age  > ALL ( SELECT c2.age
                                        FROM Customers c2
                                        WHERE c2.type = 'snowboard' );
```

What happens if this subquery returns the empty set?

- Find the names of the oldest customers.

  Please try to write this final query yourselves.

Practice Homework 3: Schema

The example database records information on bars, customers, beers, and the associations among them.

- Beers(name, manf): stores information about beers, including the manufacturer of each beer.
- Bars(name, city, addr, license, phone): stores information about bars including their city, street address, phone number and their operating license.
- Drinkers(name, city, addr, phone): stores information about drinkers, including their city, street address and phone number.
- Likes (drinker, beer): indicates which drinker likes which beers (Note that a drinker may like many beers and many drinkers may like the same beer.)
- Sells (bar, beer, price): indicates the price of each beer sold at each bar (Note that each bar can sell many beers and many bars can sell the same beer, at possibly different prices.)
- Frequents (drinker, bar): indicates which drinker frequents which bars (Note that each drinker may frequent many bars and many drinkers may frequent the same bar.)

Practice Homework 3: Easy Queries

1. Find the names of all beers, and their prices, served by the bar 'Blue Angel'.
2. Find the name and phone number of every drinker who likes the beer 'Budweiser'.
3. Find the names of all bars frequented by both 'Vince' and 'Herb'.
4. Find all bars in 'Chicago' (and display all attributes) for which we know either the address (i.e., addr in our schema) or the phone number but not both.