Introduction to SQL

Alessandro Bozzon
TI1506: Web and Database Technology
ti1506-ewi@tudelft.nl

TU Delft

Announcements

- Midterm next week!
  - Enrol, check date and time
- **Optional** werkcollege on SQL
  - Google Chrome installed on the Laptop
  - Location: DW-TZ 2, time: 13.45

Course overview [DB]

1. Introduction to Database Systems
2. The Relational Model
3. **Introduction to SQL**
4. Introduction to NoSQL database systems
5. Advanced SQL Topics
6. Conceptual Data Modelling with ER Diagrams
7. Database Conceptual Design
8. Database Logical Design

At the end of this lecture, you should be able to …

- **Describe** and **design** SQL programs for the creation, altering, and manipulation of tables

- **Develop** logical database schema, with principled design that enforce data integrity

- **Prototype** and **deploy** database applications using open-source database systems (e.g., MySQL)

The SQL Language

Requirements of a database language

- **Functional requirements**
  - Create database and relation structures
  - Perform data management tasks
    - Insertion, Modification, Deletion
  - Perform simple and complex queries

- **Non-functional requirements**
  - Structure and syntax easy to learn
  - Portability across systems

The SQL Language

- The name is an acronym for **S**tructured **Q**uery **L**anguage
- Far richer than a query language: a **DML**, a **DDL/VDL**

- SQL is a **set-based** language
  - operators works on relations (tables)
  - results are always relations (tables)

- SQL is declarative
  - It describes **what** to do with data, not **how** to do it

Data Manipulation Language

- Instructions to retrieve information of interests
  - `SELECT`

- Instructions to modify instances of the database
  - **`INSERT`**: add new row(s) to a table
  - **`UPDATE`**: modify existing row(s) in a table
  - **`DELETE`**: delete existing row(s) from a table

Data Definition Language

- Specification of the **database schema**: creation, modification, deletion of tables
  - `CREATE TABLE, ALTER TABLE, DROP TABLE`

- Specification of derived tables, or views
  - `CREATE VIEW, ALTER VIEW, DROP VIEW`

- Specification of indexes
  - `CREATE INDEX, DROP INDEX`

- Management of user access control
  - `GRANT, REVOKE`

Example DB1 : Employees

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Department		
DeptName	**Address**	**City**
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

Example DB2 : Products

Products				
CodeP	**NameP**	**Color**	**Size**	**Storehouse**
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

Supplier			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply		
CodeS	**CodeP**	**Amount**
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Querying

SQL as a Query Language

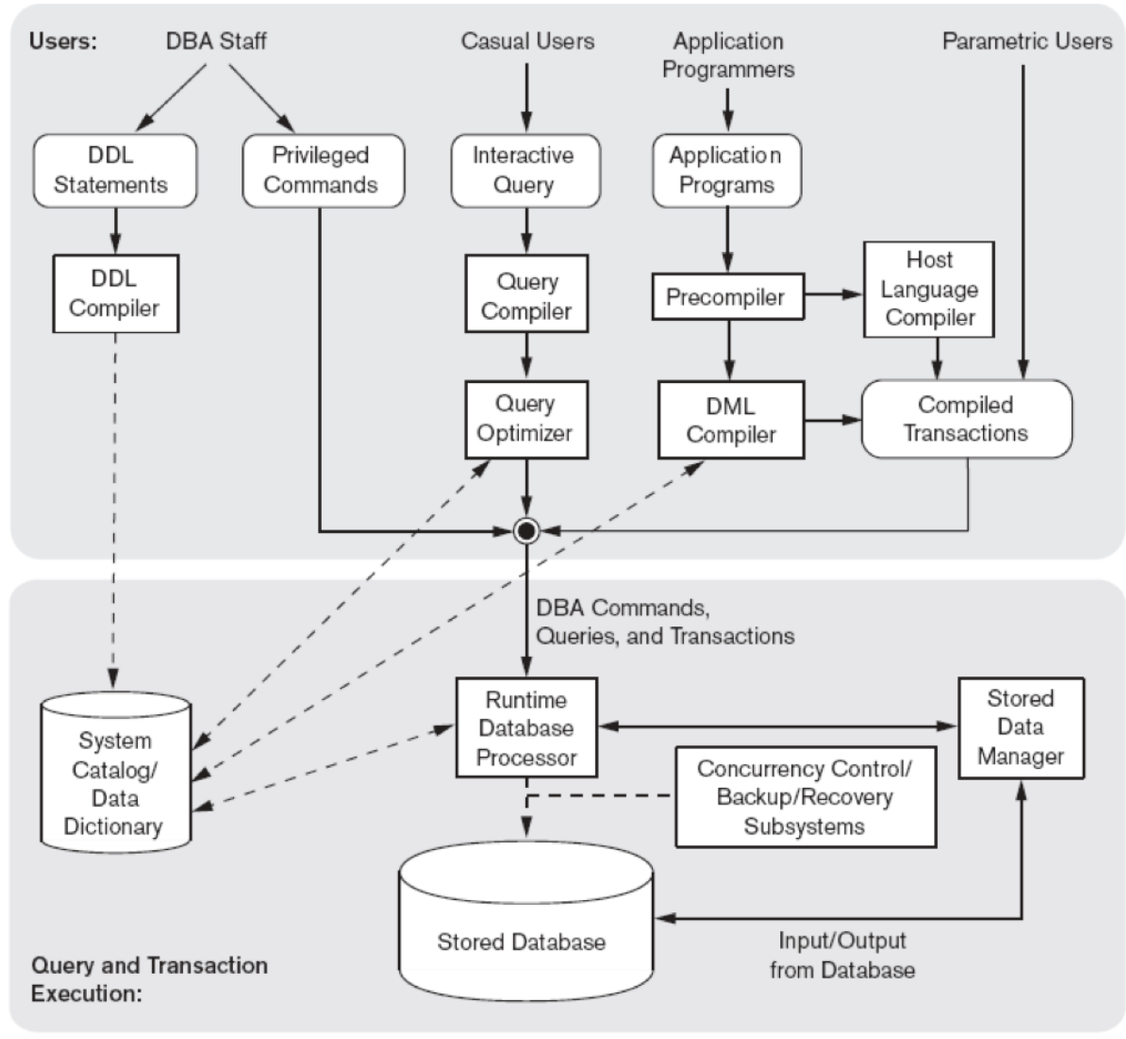


Figure 2.3
Component modules of a DBMS and their interactions.

SQL as a Query Language

- SQL expresses queries in **declarative** way
  - queries specify the properties of the result, not the way to obtain it

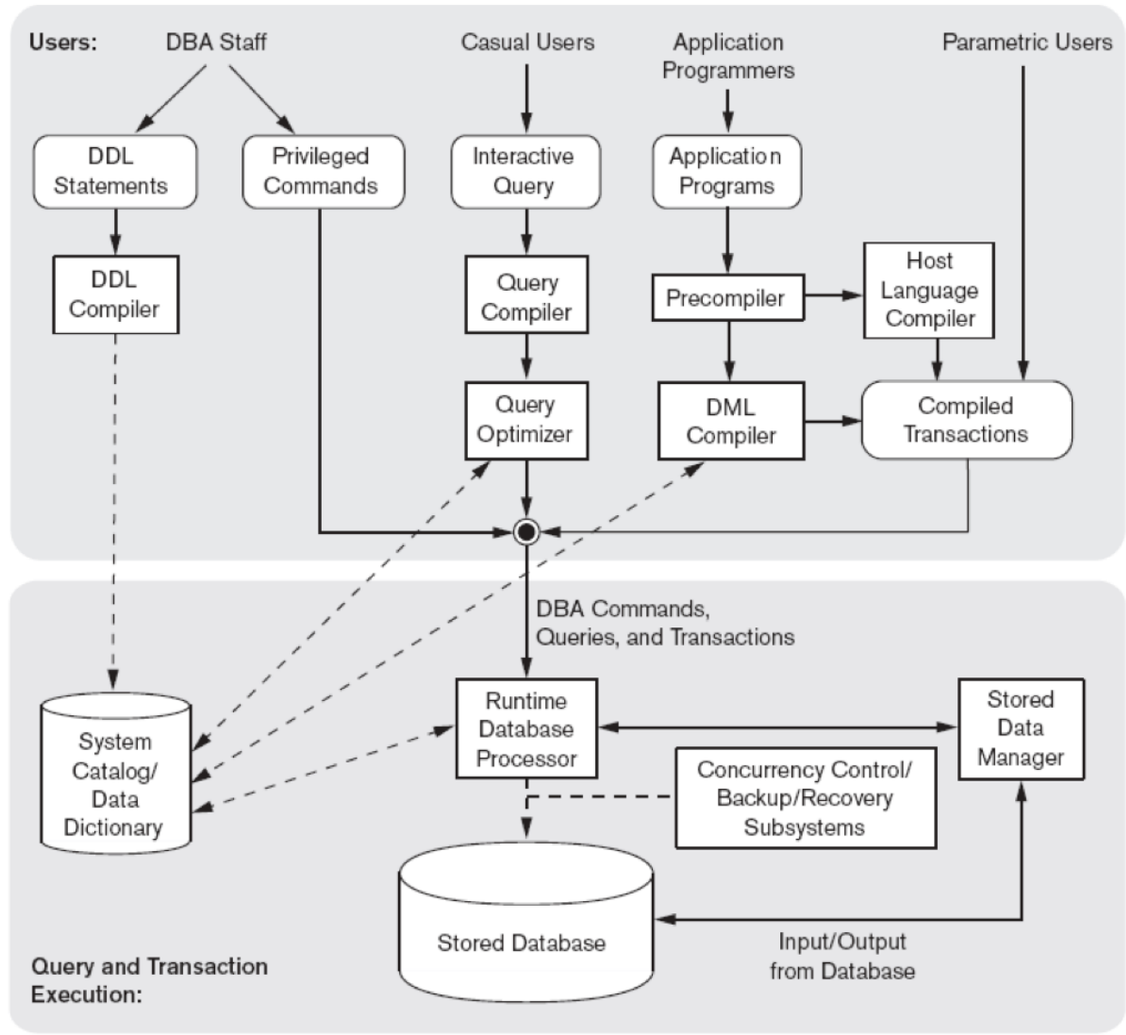


Figure 2.3
Component modules of a DBMS and their interactions.

SQL as a Query Language

- SQL expresses queries in **declarative** way
  - queries specify the properties of the result, not the way to obtain it

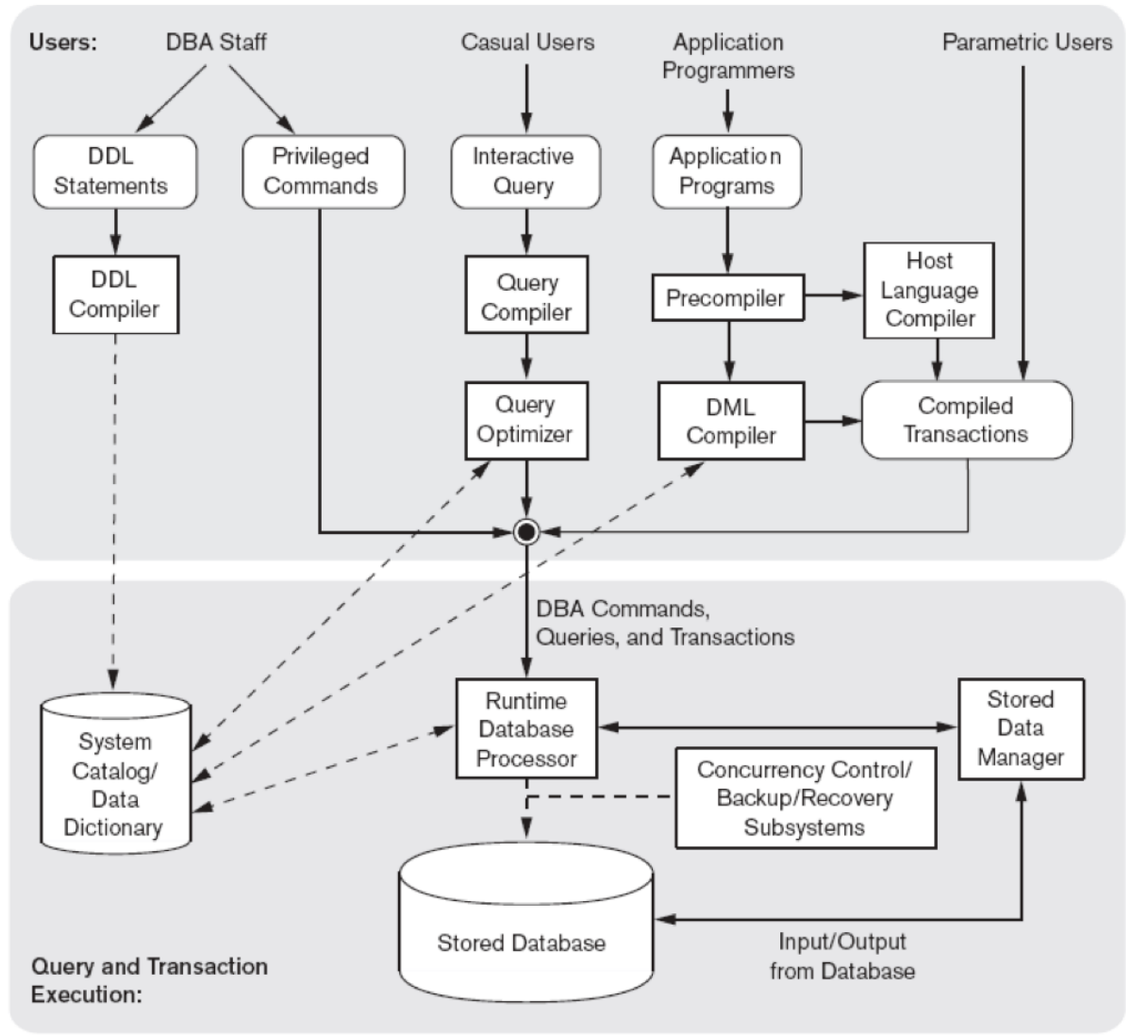


Figure 2.3
Component modules of a DBMS and their interactions.

SQL as a Query Language

- SQL expresses queries in **declarative** way
  - queries specify the properties of the result, not the way to obtain it

- Queries are translated by the query optimiser into the procedural language internal to the DBMS

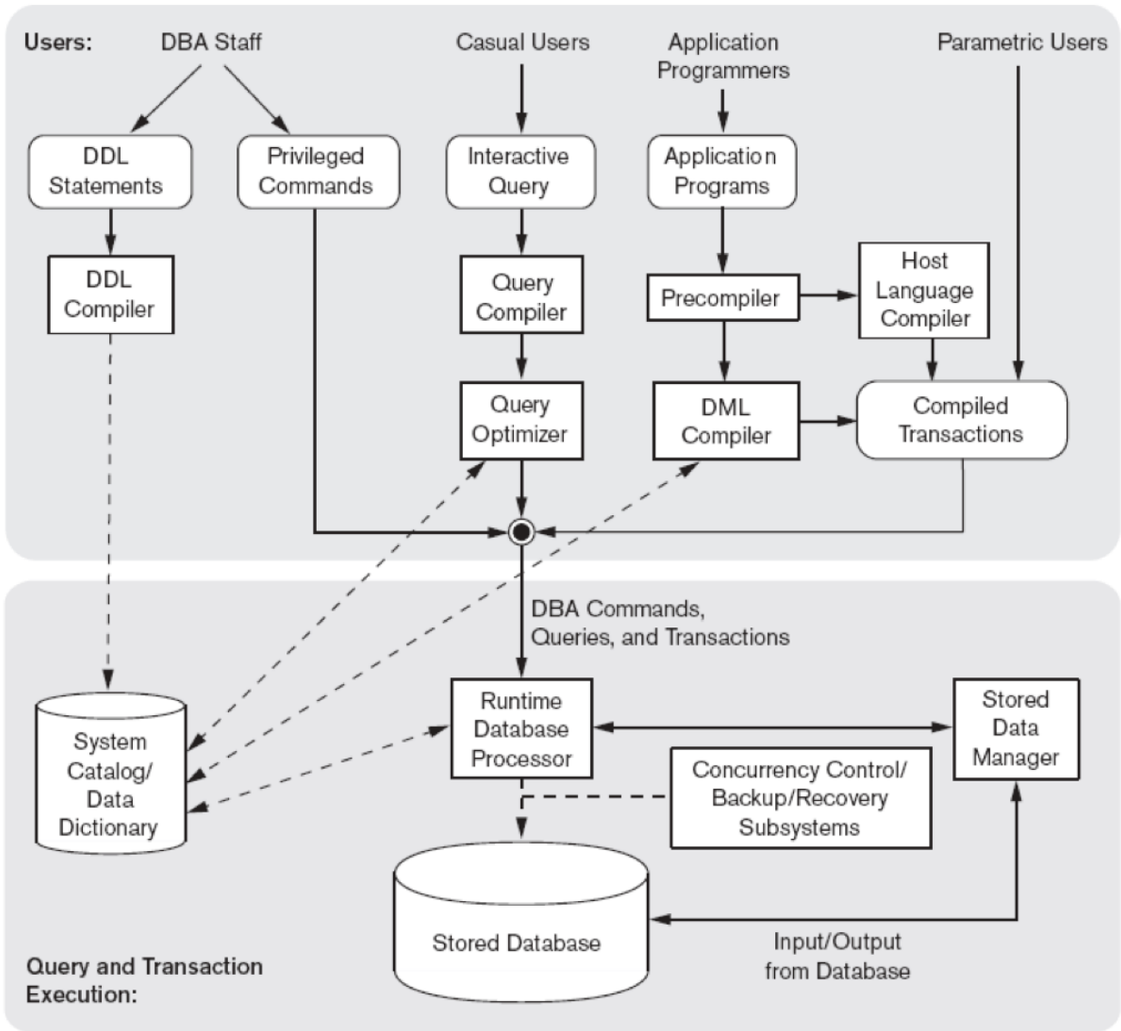


Figure 2.3
Component modules of a DBMS and their interactions.

SQL as a Query Language

- SQL expresses queries in **declarative** way
  - queries specify the properties of the result, not the way to obtain it

- Queries are translated by the query optimiser into the procedural language internal to the DBMS

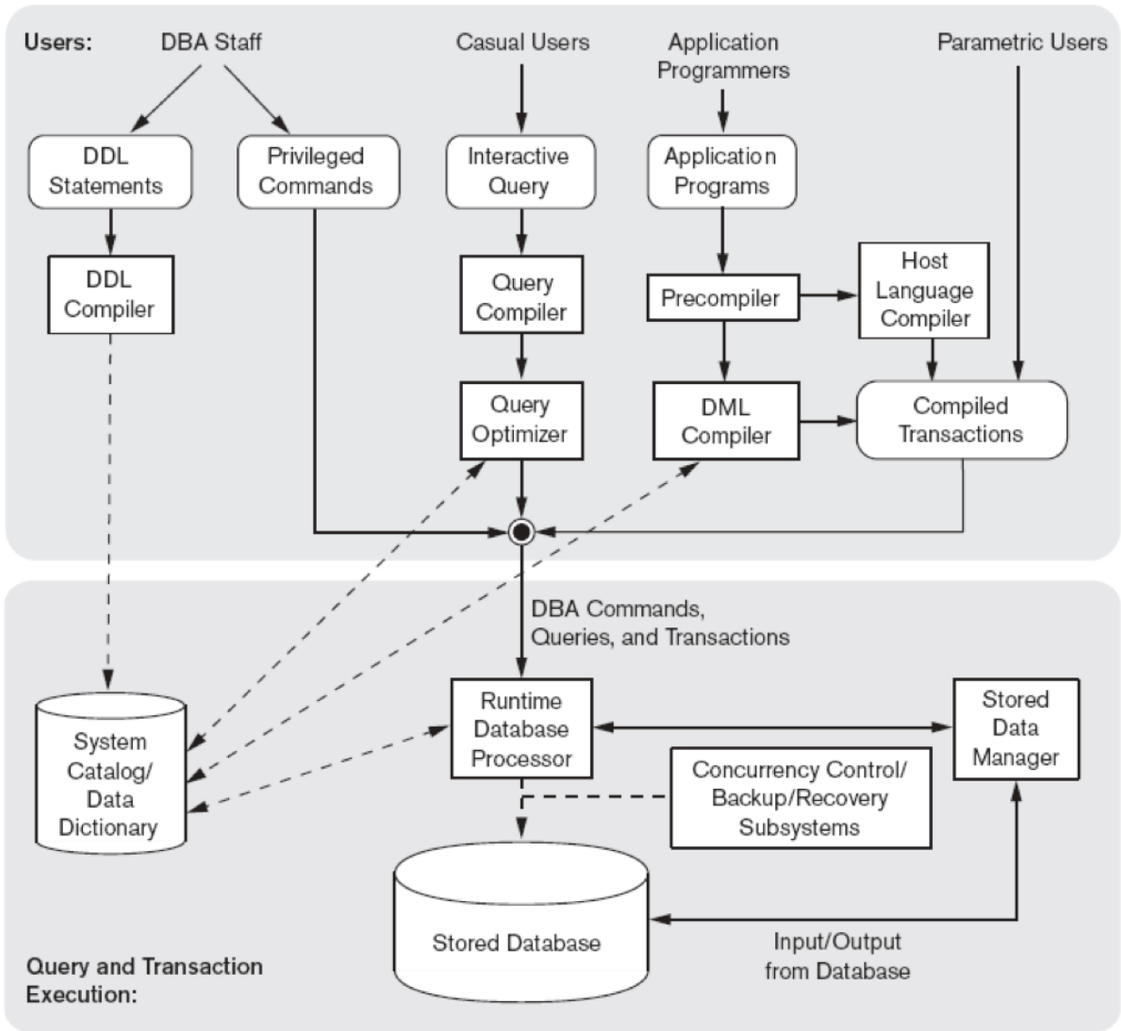


Figure 2.3
Component modules of a DBMS and their interactions.

SQL as a Query Language

- SQL expresses queries in **declarative** way
  - queries specify the properties of the result, not the way to obtain it

- Queries are translated by the query optimiser into the procedural language internal to the DBMS

- The programmer **should focus on readability, not on efficiency**

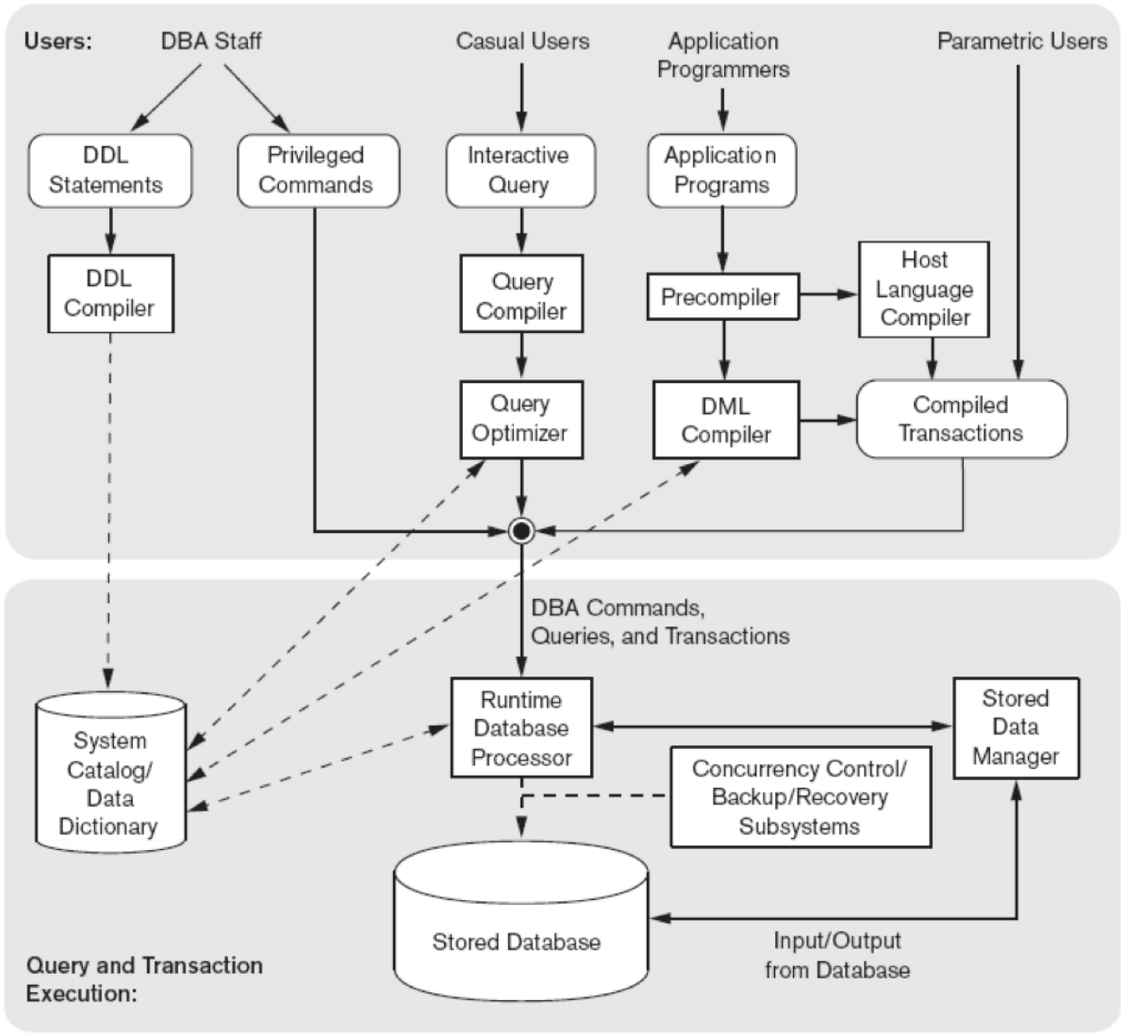


Figure 2.3
Component modules of a DBMS and their interactions.

SQL as a Query Language

- SQL expresses queries in **declarative** way
  - queries specify the properties of the result, not the way to obtain it

- Queries are translated by the query optimiser into the procedural language internal to the DBMS

- The programmer **should focus on readability, not on efficiency**

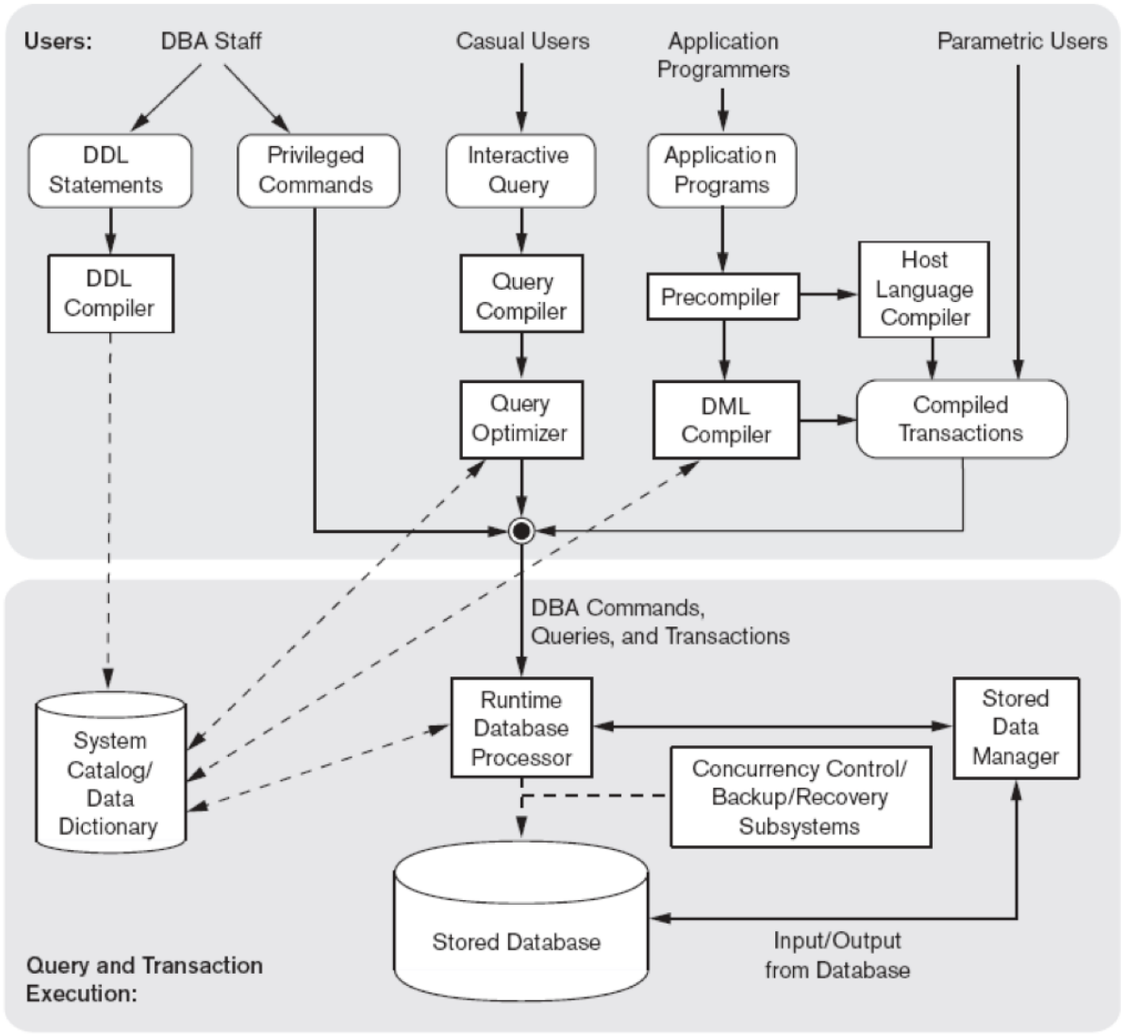


Figure 2.3
Component modules of a DBMS and their interactions.

SQL Queries

- Expressed by the SELECT statement

```
SELECT TargetList
FROM TableList
[ WHERE Condition ]
[ GROUP BY GroupingAttributeList ] [ HAVING AggregateCondition ]
[ ORDER BY OrderingAttributeList ]
```

- The three parts of the query are usually called:
  - target list or **`SELECT`** clause
  - **`FROM`** clause
  - **`WHERE`** clause

- The query:
  - considers the cartesian product of the tables in the **`FROM`** clause
  - considers only the rows that satisfy (evaluate to **`TRUE`**) the condition in the **`WHERE`** clause
  - for each row evaluates the attribute expressions in the TargetList, and returns them
  - More on **`ORDER BY`**, **`GROUP BY`** and **`HAVING`** later

SELECT Clause

SELECT query from / I

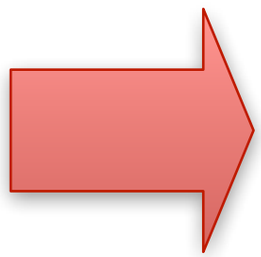
DB2

Find the *code* of **all** products in the DB

```
SELECT CodeP
FROM Products
```

Result

Products				
CodeP	**NameP**	**Color**	**Size**	**Storehouse**
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam



CodeP
P1
P2
P3
P4
P5
P6

SELECT query from /2

DB2

Find the *code* and *number of shareholders* of suppliers

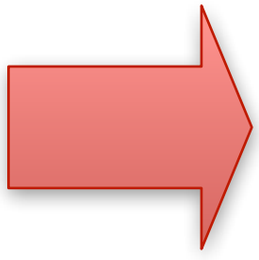
located in "Den Haag"

```
SELECT CodeS, Shareholders
FROM Supplier
WHERE Office = "Den Haag"
```

Result

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht



CodeS	Shareholders
S2	1
S3	3

SELECT query from /2

DB2

Find the *code* and *number of shareholders* of suppliers

located in "Den Haag"

```
SELECT CodeS, Shareholders
FROM Supplier
WHERE Office = "Den Haag"
```

Result

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

CodeS	Shareholders
S2	1
S3	3

Only combination of tuples evaluating the logical expression in the WHERE clause to TRUE are selected

* in the target list

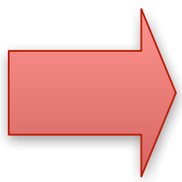
DB1

Find all the information relating to employees named “Brown”

```
SELECT *
FROM Employee
WHERE Surname = "Brown"
```

Result

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse



FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	Brown	Planning	14	80	London

Attribute Expressions with AS / 1

DB1

- The keyword **AS** allows the definition of an **alias**. Used in attribute expressions, it defines a new **temporary** column per the calculated expression

Find the monthly salary of the employees *named* "**White**"

```
SELECT Salary / 12 AS MonthlySalary
FROM Employee
WHERE Surname = "White"
```

Result

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

MonthlySalary
3.00

Attribute Expressions with AS / 2

DB1

Find the salaries of employees named "**Brown**", and *alias* it as "**Remuneration**"

```
SELECT Salary AS Remuneration
FROM Employee
WHERE Surname = "Brown"
```

Result

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Remuneration
45
80

Duplicates

DB1

- In relational algebra and calculus the results of queries **do not contain duplicates** (set semantics)
- In SQL, tables may have identical **rows** (bag semantics)
- Duplicates **rows** can be removed using the keyword `DISTINCT`
  - This applies also with rows having multiple columns

```
SELECT City
FROM Department
```

Department		
DeptName	**Address**	**City**
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

City
London
Toulouse
Brighton
London
San Joné

```
SELECT DISTINCT City
FROM Department
```

Department		
DeptName	**Address**	**City**
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

City
London
Toulouse
Brighton
San Joné

DISTINCT Keyword

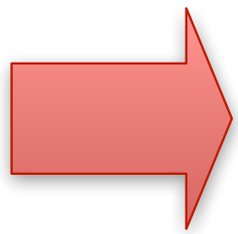
DB2

Find the *code* of the products ***supplied at least by one supplier***

```
SELECT DISTINCT CodeP
FROM Supply
```

Result

Supply		
CodeS	**CodeP**	**Amount**
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400



CodeP
P1
P2
P3
P4
P5
P6

WHERE Clause

`WHERE` Clause

- Selection conditions apply to **each single tuple** resulting from the evaluation of the `FROM` clause

- Defined as a **boolean expression of simple predicates**

- Simple predicates
  - comparison between attributes and/or constant values
  - set membership
  - textual matching
  - `NULL` values

Predicate Conjunction / I

DB1

Find the *first names* and *surnames* of the employees **who work in office number 20 of the Administration department**

```
SELECT FirstName, Surname
FROM Employee
WHERE Office = "20" AND Dept = "Administration"
```

Result

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName	Surname
Gus	Green

Predicate Conjunction /2

DB1

Find the *first names* and *surnames* of the employees who work in the *Administration* department **and** in the *Production* department

```
SELECT FirstName, Surname
FROM Employee
WHERE Dept = "Administration" AND Dept = "Production"
```

Result

No Results!

Predicate Disjunction / I

DB1

Find the *first names* and *surnames* of the employees **who work in either the Administration or the Production department**

```
SELECT FirstName, Surname
FROM Employee
WHERE Dept = "Administration" OR Dept = "Production"
```

Result

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName	Surname
Mary	Brown
Charles	White
Gus	Green
Pauline	Bradshaw
Alice	Jackson

Complex Logical Expressions

DB1

Find the **first names** of the employees **named "Brown" who work in the Administration department or the Production department**

```
SELECT FirstName
FROM Employee
WHERE Surname = "Brown" AND
                (Dept = "Administration" OR Dept = "Production")
```

Result

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName
Mary

MySQL Logical Operators: http://dev.mysql.com/doc/refman/5.0/en/logical-operators.html

Operator `LIKE`

- Matching string patterns

- The character “_” is a matching term for any single character, which must be found in the specified position

- The character “%” is a matching term for any sequence of N characters, possibly empty

Operator LIKE / I

DB2

Find the code and the name of the products having name **starting with the letter S**

```
SELECT CodeP, NameP
FROM Products
WHERE NameP LIKE "S%"
```

Result

Products				
CodeP	**NameP**	**Color**	**Size**	**Storehouse**
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P1	Sweater
P3	Shirt
P4	Shirt
P5	Skirt

Operator `LIKE` /2

DB1

Find the employees with surnames **that have "r" as the second letter** and **end in "n"**

```
SELECT *
FROM Employee
WHERE Surname LIKE "_r%n"
```

Result

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Gus	Green	Administration	20	40	Oxford
Charles	Brown	Planning	14	80	London

Operator `LIKE` /3

DB2

Find Suppliers having the Office address **containing** the string **“Den Haag”**

```
Address LIKE "%Den Haag%"
```

Find Suppliers where the **supplier code is 2**

Assume we don’t know the first character, and that the code is exactly 2 characters long

```
CodeS LIKE "_2"
```

Find Products that are in storehouses in locations having names that **do not contain an “e” as second character**

```
Storehouse NOT LIKE "_e%"
```

MySQL LIKE Operator Reference: http://dev.mysql.com/doc/refman/5.7/en/string-comparison-functions.html#operator_like

Other MySQL String Functions: http://dev.mysql.com/doc/refman/5.7/en/string-functions.html

Burst1_Q1

Burst1_Q2

Burst1_Q3

Dealing with `NULL` values

- `NULL` values may mean that:
  - a value is **unknown** (exists but it is not known)
  - a value is **not available** (exists but it is purposely withheld)
  - a value is **not applicable** (undefined for this tuple)

- Each individual **`NULL`** value is considered **to be different** from every other **`NULL`** value
- When a **`NULL`** is involved in a comparison operation, the results is considered to be **`UNKNOWN`**

- SQL uses a **three-valued logic**
  - **`TRUE`**, **`FALSE`**, and **`UNKNOWN`**
  - All logical operators evaluate to **`TRUE`**, **`FALSE`**, or **`NULL`** (UNKNOWN)
  - In MySQL, these are implemented as `1` (TRUE), `0` (FALSE), and **`NULL`**
  - Most of this is common to different SQL database servers, although some servers may return any nonzero

Comparisons involving NULL and Three-Valued Logic

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

NULL comparison

DB2

Find the code and the name of products **having size greater than 44**

```
SELECT CodeP, NameP
FROM Products
WHERE Size > 44
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	NULL	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P2	Jeans
P3	Shirt

NULL comparison

DB2

Find the code and the name of products **having size greater than 44**

```
SELECT CodeP, NameP
FROM Products
WHERE Size > 44
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	NULL	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P2	Jeans
P3	Shirt

Tuples with `Size = NULL` are not selected

NULL comparison

DB2

Find the code and the name of products **having size greater than 44**

```
SELECT CodeP, NameP
FROM Products
WHERE Size > 44
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	NULL	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P2	Jeans
P3	Shirt

Tuples with `Size = NULL` are not selected

Only combination of tuples evaluating the logical expression in the WHERE clause to TRUE are selected

The `IS NULL` Operator

DB2

- AttributeName `IS [NOT] NULL`

Find the code and the name of products having no specified `Size`

```
SELECT CodeP, NameP
FROM Products
WHERE Size IS NULL
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	NULL	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P5	Skirt

The `IS NULL` Operator

DB2

Find the code and the name of products **having size greater than 44**, or that **might have size greater than 44**

```
SELECT CodeP, NameP
FROM Products
WHERE Size > 44 OR Size IS NULL
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	NULL	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P2	Jeans
P3	Shirt
P5	Skirt

Burst2_Q1

Burst2_Q2

Burst2_Q3

Ordering of Results

Ordering

- The **`ORDER BY`** clause, at the end of the query, orders the rows of the results
  - Last operator applied by the database

- Syntax:

```
ORDER BY OrderingAttribute [ asc | desc ]
         {, OrderingAttribute [ asc | desc ] }
```

- The implicit ordering is **`ASC`**: ascending

ORDER BY /I

DB1

Find the code, name and the size of all the products, in *descending order of size*

```
SELECT CodeP, NameP, Size
FROM Products
ORDER BY Size DESC
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP	Size
P2	Jeans	48
P3	Shirt	48
P4	Shirt	44
P6	Coat	42
P1	Sweater	40
P5	Skirt	40

ORDER BY /2

DB1

Find all the information about products, in *ascending order* of *name* and *descending order of size*

```
SELECT *
FROM Products
ORDER BY NameP, Size DESC
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP	Color	Size	Storehouse
P6	Coat	Red	42	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P1	Sweater	Red	40	Amsterdam

ORDER BY /3

DB1

Find the code and the *american* size of all the products, in *ascending order of size*

```
SELECT CodeP, Size - 14 AS AmericanSize
FROM Products
ORDER BY AmericanSize
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	AmericanSize
P1	26
P5	26
P6	28
P4	30
P2	34
P3	34

Join

Querying Multiple Tables

DB1

- All possible tuple combinations
- What if we want to retrieve

the name of all the **suppliers of product "P2"**

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Cross Product /1

DB1

- All possible tuple combinations

Find the name of **all** the suppliers **of** product P2

```
SELECT NameS
FROM Supplier, Supply
```

Result

Supplier				Supply		
CodeS	**NameS**	**Shareholders**	**Office**	**CodeS**	**CodeP**	**Amount**
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S3	P2	200
...	...	...	...	...	...	...

Cross Product /2

DB1

Find the name of **all** the suppliers **of** product P2

Result

Supplier				Supply		
CodeS	**NameS**	**Shareholders**	**Office**	**CodeS**	**CodeP**	**Amount**
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300

What is the problem with this result set?

DB1

Simple JOIN / I

```
SELECT NameS
FROM Supplier, Supply
WHERE Supplier.CodeS = Supply.CodeS
```

Result

=

Supplier				Supply		
CodeS	**NameS**	**Shareholders**	**Office**	**CodeS**	**CodeP**	**Amount**
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S3	P2	200
...	...	...	...	...	...	...

Simple `JOIN` /2

- `Supplier.CodeS = Supply.CodeS` is a **`JOIN` CONDITION**

Result

Supplier				Supply		
CodeS	**NameS**	**Shareholders**	**Office**	**CodeS**	**CodeP**	**Amount**
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

Our Original Query

DB1

Find the name of **all** the suppliers **of** product P2

```
SELECT NameS
FROM Supplier, Supply
WHERE Supplier.CodeS = Supply.CodeS AND CodP = "P2"
```

Result

Supplier				Supply		
CodeS	**NameS**	**Shareholders**	**Office**	**CodeS**	**CodeP**	**Amount**
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

NameS
John
Victor
Anna

Another Query

DB1

Find the name of supplier of **at least one red product**

```
SELECT NameS
FROM Supplier, Supply, Products
WHERE Supplier.CodeS = Supply.CodeS AND Supply.CodeP = Product.CodeP
        AND Color = "Red"
```

Result

Products				
CodeP	**NameP**	**Color**	**Size**	**Storehouse**
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

Supplier			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply		
CodeS	**CodeP**	**Amount**
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

NameS
John
Victor

If there are *N* tables in the **`FROM`** clause, at least *N* – *1* **`JOIN`** conditions in the **`WHERE`** clause

Using AS keyword for tables

Find the **code pairs** of **suppliers** having their office **in the same city**

```
SELECT S1.CodeS, S2.CodeS
FROM Supplier AS S1, Supplier AS S2
WHERE S1.Office = S2.Office
```

- Note the use of alias in the **`FROM`** clause

Result?

Supplier as S1			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supplier as S2			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Result / 1

Find the **code pairs** of **suppliers** having their office **in the same city**

Supplier as S1			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supplier as S2			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Result?

S1.CodeS	S2.CodeS
S1	S1
S1	S4
S2	S2
S2	S3
S3	S2
S3	S3
S4	S1
S4	S4
S5	S5

- **Pairs** of identical values
- **Permutations** of the same pair of values

Result /2

Find the **code pairs** of **suppliers** having their office **in the same city**

```
SELECT S1.CodeS, S2.CodeS
FROM Supplier AS S1, Supplier AS S2
WHERE S1.Office = S2.Office AND S1.CodeS < > S2.CodeS
```

Result?

S1.CodeS	S2.CodeS
~~S1~~	~~S1~~
S1	S4
~~S2~~	~~S2~~
S2	S3
S3	S2
~~S3~~	~~S3~~
S4	S1
~~S4~~	~~S4~~
~~S5~~	~~S5~~

- Remove pairs with the same code value

Result /3

Find the **code pairs** of **suppliers** having their office **in the same city**

```
SELECT S1.CodeS, S2.CodeS
FROM Supplier AS S1, Supplier AS S2
WHERE S1.Office = S2.Office AND S1.CodeS < S2.CodeS
```

Result?

S1.CodeS	S2.CodeS
~~S1~~	~~S1~~
S1	S4
~~S2~~	~~S2~~
S2	S3
~~S3~~	~~S2~~
~~S3~~	~~S3~~
~~S4~~	~~S1~~
~~S4~~	~~S4~~
~~S5~~	~~S5~~

Let’s keep only the right ones

Burst3_Q1

Burst3_Q2

Burst3_Q3

Today we covered

- SQL as
  - a Retrieval Language
- Next Lecture
  - SQL as a Schema Creation and Modification Language
  - a Data Manipulation Language

Readings

- **Book**
  - *4.3: Basic Retrieval Queries in SQL*
  - *5.1.1: Comparison involving NULL and three-valued logic*
  - *5.1.5: Explicit Sets and Renaming of Attributes in SQL*
  - *5.1.6: Joined Tables in SQL and Outer Joins*

- **Suggested Readings on Blackboard**

End of Lecture