SQL / DML / SELECT / Union(∪)

FirstName	LastName
Stanley	Kubrick
Clint	Eastwood
John	Travolta
Samuel	Jackson
Uma	Thurman
Clint	Eastwood

```
(SELECT FirstName, LastName FROM Director WHERE PlaceOfBirth='USA')
UNION ALL
(SELECT FirstName, LastName FROM Actor WHERE PlaceOfBirth='USA')
```

SQL / DML / SELECT / Select(σ)

Director

Id	FirstName	LastName	DateOfBirth	PlaceOfBirth	BestMovieId	MovieCount
1	Stanley	Kubrick	Jul. 26, 1928	USA	1	13
2	Alfred	Hitchcock	Aug. 13, 1899	England	203	47
3	Clint	Eastwood	May 31, 1930	USA	803	35

Which director made between 10 and 40 movies?

$\sigma_{\text{MovieCount >= 10 AND MovieCount <= 40}}(\text{Director})$

```
SELECT * FROM Director
WHERE MovieCount BETWEEN 10 AND 40
```

42

HAVING clause is used to specify filter condition for a group of rows.

Often used with the GROUP BY clause to filter groups based on specific conditions.

GROUP BY applies a filter condition to each group while WHERE clause applies the filter condition on each row.

If GROUP BY clause is omitted, it behaves like a WHERE clause.

Advanced SQL / OEDERED BY

Movie

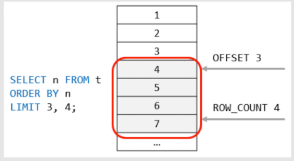
Id	Title	Language	ReleaseDate	RunningTime
2	Rosemary's Baby	English	1968	NULL
4	Planet of the Apes	English	1968	112
1	2001: A Space Odyssey	English	1968	142
3	The Birds	English	1963	119

List movies sorted by release date & running time?

```
SELECT *
FROM Movie
ORDER BY ReleaseDate DESC,
         RunningTime ASC ;
```

Advanced SQL / LIMIT

```
SELECT
   select_list
FROM
   table
LIMT [offset,] row_count;
```



List all staff with the string 'Glasgow' in their address.

```
SELECT sno, fname, lname, address
   FROM  staff
      WHERE   address LIKE '%Glasgow%';
```

Advanced SQL / GROUP BY / HAVING

orderdetails
* orderNumber
* productCode
quantityOrdered
priceEach
orderLineNumber

ordernumber	itemsCount	total
10100	151	10223.83
10101	142	10549.01
10102	80	5494.78
10103	541	50218.95
10104	443	40206.20
10105	545	53959.21
10106	675	52151.81
10107	229	22292.62

What are the total sales and number of items per order number ?

```
SELECT  orderNumber,  SUM(quantityOrdered * priceEach)
AS total ,  SUM (quantityOrdered) AS itemCounts
FROM ordersdetails
GROUP BY orderNumber;
```

Advanced SQL / GROUP BY /

Movie

Id	Title	Language	ReleaseDate	RunningTime
1	2001: A Space Odyssey	English	1968	142
2	Rosemary's Baby	English	1968	NULL
3	The Birds	English	1963	119
4	Planet of the Apes	English	1968	112

In which years more than 2 movies have been released?

```
SELECT ReleaseDate, COUNT(*)
FROM Movie
GROUP BY ReleaseDate
HAVING COUNT(*) > 2
```

ReleaseDate	COUNT(*)
1968	3

HAVING only accepts AGG or columns in GROUP

Advanced SQL

Movie

Id	Title	Language	ReleaseDate	RunningTime
1	2001: A Space Odyssey	English	1968	142
2	Rosemary's Baby	English	1968	NULL
4	Planet of the Apes	English	1968	112
3	The Birds	English	1963	119

Top-2 recent movies after skipping the most recent one?

```
SELECT *
FROM Movie
ORDER BY ReleaseDate DESC
LIMIT 1, 2
```

```
SELECT   select_list
FROM     table
WHERE    expr operator
                  (SELECT      select_list
                  FROM         table);
```

- The subquery (inner query) executes once before the main query.
- The result of the subquery is used by the main query (outer query).

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison operator.
- Do not add an ORDER BY clause to a subquery.
- Use single-row operators with single-row subqueries.
- Use multiple-row operators with multiple-row subqueries.

```
SQL> SELECT    deptno, MIN(sal)
  2  FROM      emp
  3  GROUP BY  deptno
  4  HAVING    MIN(sal) >
  5                     (SELECT   MIN(sal)
  6                     FROM      emp
  7                     WHERE     deptno = 20);
```

800

Using ALL Operator in Multiple-Row Subqueries

Display the employees whose salary is greater than the average salaries of all the departments.

```
SQL> SELECT  empno, ename, job
  2  FROM    emp
  3  WHERE   sal > ALL
  4                 (SELECT      avg(sal)
  5                 FROM         emp
  6                 GROUP BY     deptno);

    EMPNO ENAME      JOB
--------- ---------- ---------
     7839 KING       PRESIDENT
     7566 JONES      MANAGER
     7902 FORD       ANALYST
     7788 SCOTT      ANALYST
```

Subquery × FROM

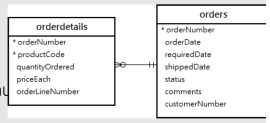
SELECT Columns
FROM Tables, (SELECT …) AS S1, (SELECT …) AS S2, ..
WHERE (S1.c1 > values)

```
SELECT column_list
FROM (
   SELECT column_list
   FROM table_1
) derived_table_name
WHERE derived_table_name.c1 > 0;
```

Derived table

Must have an alias

Subquery × FROM



Finds the top 5 products by sales revenu
in 2003.

```
SELECT  productCode, SUM(quantityOrdered * priceEach)  AS sales
FROM orderdetails
INNER JOIN  orders USING (orderNumber)
 WHERE
    YEAR(shippedDate) = 2003
   GROUP BY productCode
   ORDER BY sales DESC
    LIMIT 5;
```

productCode	sales
S18_3232	103480
S10_1949	67985
S12_1108	59852
S12_3891	57403
S12_1099	56462

Single-Row Subqueries Example

- Display the employees whose job title is the same as that of employee 7369

```
SQL> SELECT    ename, job
  2  FROM      emp
  3  WHERE     job =
  4                  (SELECT  job
  5                        FROM      emp
  6                        WHERE     empno = 7369);
```

```
ENAME      JOB
---------- ---------
JAMES      CLERK
SMITH      CLERK
ADAMS      CLERK
MILLER     CLERK
```

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

In Class activity

Movie

Id	Title	Language	ReleaseDate	RunningTime
1	2001: A Space Odyssey	English	1968	142
2	Rosemary's Baby	English	1968	NULL
3	The Birds	English	1963	119
4	Planet of the Apes	EN	1968	112

What are the longest movies in 1963?

```
SELECT *
FROM Movie
WHERE ReleaseDate = 1963  AND
        RunningTime = (SELECT MAX(RunningTime)
                       FROM Movie
                       WHERE ReleaseDate = 1963)
```

Decomposition Of Relations

The previous table can be decomposed into the following two tables

- A **functional dependency** occurs when the value of one (set of) attribute(s) determines the value of a second (set of) attribute(s):
  StudentID → StudentName
  StudentID → (DormName, DormRoom, Fee)
- The attribute on the left side of the functional dependency is called the **determinant**.
- Functional dependencies may be *based* on equations:
  ExtendedPrice = Quantity X UnitPrice
  (Quantity, UnitPrice) → ExtendedPrice
- But, function dependencies are definitely ***not*** equations!

Composite determinant: a determinant of a functional dependency that consists of more than one attribute.

(StudentName, ClassName) → (Grade)

- A **functional dependency** occurs when the value of one (or set of) attribute(s) determines the value of a second (or set of) attribute(s):
  StudentID → StudentName
  StudentID → (DormName, DormRoom, Fee)
- The attribute on the left side of the functional dependency is called the **determinant**, the attribute on the right side is called the **dependent**.
- Functional dependencies may be based on equations:
  ExtendedPrice = Quantity X UnitPrice
  (Quantity, UnitPrice) → ExtendedPrice
- Function dependencies are not equations

A superkey is an attribute or a set of attributes that identify an entity UNIQUELY.
- **In a relation (table), a SUPERKEY is any column or set of columns whose values can be used to distinguish one row from another.**

- **A superkey is a candidate key if it is minimal, i.e., if X is a superkey, then X minus {any attribute of X} is NOT a superkey.**

- **A primary key is a candidate key which we choose to be THE "key."**

Generalization and rule for trivial FD's:

An FD is trivial if it has the form:
X ----> Y, where Y is a subset of X.

So, ABCD ----> ABC is a trivial FD.

A trivial FD does not make a significant statement about real world constraints - we are thus only interested in non-trivial FD's.

1NF

1st Normal Form requires that the domain of each attribute contains only atomic (indivisible) values.

No composite attribute
No multivalued attribute

Manager	Employees
Pat	Minash, Toby
Alex	Mary, Joe
Jane	Seb, Suba

- This data has some problems:
  - The Employees column is not atomic.
    - A column must be atomic, meaning that it can only hold a single item of data. This column holds more than one employee name.
- **Congratulations!**
- The fact that all our data and columns is atomic and we have a primary key means that we are in 1NF!

Manager	Employees
Pat	Minash
Pat	Toby
Alex	Mary
Alex	Joe
Jane	Seb
Jane	Suba

- A determinant has unique values (i.e., all values are different) in a relation if, and only if, it functionally determines every other attribute in the relation
  - *So, in SKU_Data, SKU has all different (unique) values, and it functionally determines every attribute in the table.*
  - *On the other hand, Buyer, though a determinant, does not have unique values, and does not functionally determine all the other attributes in the relation.*
- So, you cannot find the determinants of all functional dependencies simply by looking for unique values in one column

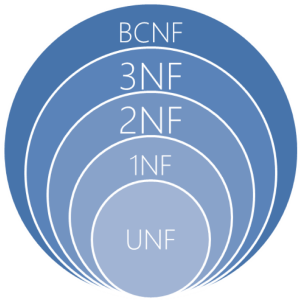
Consider the following scheme from an airline database system:
(P(pilot) , F(flight#), D(date), T (scheduled time to depart))

We have the following FD's :
- F ----> T PDT ----> F FD ----> P

Provide some superkeys:
- PDT is a superkey, and FD is a superkey.
- Is PDT a candidate key?
  - PD is not a superkey, nor is DT, nor is PT.
  - So, PDT is a candidate key.
- FD is also a candidate key, since neither F or D are superkeys.

- For a database to be in a normal form, it must meet all requirements of the previous forms:
  - e.g., For a database to be in 2NF, it must already be in 1NF.
  - For a database to be in 3NF, it must already be in 1NF and 2NF.

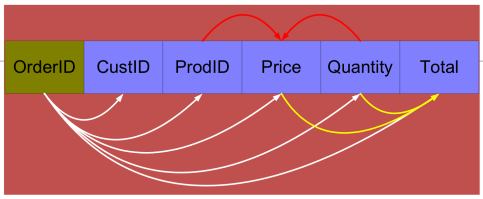
Normalization × Normal Forms



- A database in 2NF must also be in 1NF:
  - Data must be atomic
  - Every row (or tuple) must have a unique primary key
- Plus:
  - Subsets of data that apply to multiple rows (**repeating data**) are moved to separate tables
- To be in 3NF, a database must be:
  - In 2NF
  - All columns must be fully functionally dependent on the primary key (There are no transitive dependencies)

4	1003	MB-153	82	25	2,050
5	1004	ZA-245	42	10	420
6	1002	ZA-245	40	50	2,000
7	1001	AB-111	75	100	7,500

But there are some fields that are not dependent on OrderID:
- Total is the simple product of Price*Quantity. As such, has a transitive dependency to Price and Quantity.
- Because it is a calculated value, doesn't need to be included at all.



Let's diagram the dependencies.

We can see that all fields are dependent on OrderID, the Primary Key

- Price is also determined by both ProdID and Quantity rather than the primary key (red lines). This is called a **transitive dependency**. We must get rid of transitive dependencies to have 3NF.
- A database is in 3NF if:
  - It is in 2NF
  - It has no transitive dependencies
    - *A transitive dependency exists when one attribute (or field) is determined by another non-key attribute (or field)*
    - *We remove fields with a transitive dependency to a new table and link them by a foreign key.*

- Elements can have **attributes**

```
<course course_id= "CS-101">
    <title> Intro. to Computer Science</title>
    <dept name> Comp. Sci. </dept name>
    <credits> 4 </credits>
</course>
```

- Attributes are specified by *name=value* pairs inside the starting tag of an element
- An element may have several attributes, but each attribute name can only occur once

```
<course  course_id = "CS-101"  credits="4">
```

- Distinction between subelement and attribute
  - In the context of documents, attributes are part of markup, while subelement contents are part of the basic document contents
  - In the context of data representation, the difference is unclear and may be confusing
    - Same information can be represented in two ways

```
– <course course_id= "CS-101"> … </course>
– <course>
    <course_id>CS-101</course_id> …
  </course>
```

  - Suggestion: use attributes for identifiers of elements, and use subelements for contents
- The type of an XML document can be specified using a DTD
- DTD constraints structure of XML data
  - What elements can occur
  - What attributes can/must an element have
  - What subelements can/must occur inside each element, and how many times.
- DTD does not constrain data types
  - All values represented as strings in XML
- DTD syntax
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST element (attributes) >

Boyce-Codd Normal Form (BCNF)

A relation is in BCNF, if and only if, every determinant is a candidate key.

The difference between 3NF and BCNF is that for a functional dependency A → B, 3NF allows this dependency in a relation if B is a primary-key attribute and A is not a candidate key,

whereas BCNF insists that for this dependency to remain in a relation, A must be a candidate key.

Superkey is set of colums with no repitition

Court	Start Time	End Time	Rate Type
1	09:30	10:30	SAVER
1	11:00	12:00	SAVER
1	14:00	15:30	STANDARD
2	10:00	11:30	PREMIUM-B
2	11:30	13:30	PREMIUM-B
2	15:00	16:30	PREMIUM-A

BCNF Transformation

Rate Type	Court
SAVER	1
STANDARD	1
PREMIUM-A	2
PREMIUM-B	2

Rate Type	Start Time	End Time
SAVER	09:30	10:30
SAVER	11:00	12:00
STANDARD	14:00	15:30
PREMIUM-B	10:00	11:30
PREMIUM-B	11:30	13:30
PREMIUM-A	15:00	16:30

Now if we upgrade the court we can guarantee the rate type will reflect this change, and we cannot charge the wrong price for a court.

- Subelements can be specified as
  - names of elements, or
  - #PCDATA (parsed character data), i.e., character strings
- Example

```
<! ELEMENT department (dept_name, building, budget)>
<! ELEMENT dept_name (#PCDATA)>
<! ELEMENT budget (#PCDATA)>
```

- Subelement specification may have regular expressions

```
<!ELEMENT university ( ( department | course | instructor | teaches )+)>
```

  - Notation:
    - "|" - alternatives
    - "+" - 1 or more occurrences
    - "*" - 0 or more occurrences

```
<!DOCTYPE university [
  <!ELEMENT university ( (department|course|instructor|teaches)
  +)>
  <!ELEMENT department ( dept name, building, budget)>
  <!ELEMENT course ( course id, title, dept name, credits)>
  <!ELEMENT instructor (IID, name, dept name, salary)>
  <!ELEMENT teaches (IID, course id)>
  <!ELEMENT dept name( #PCDATA )>
  <!ELEMENT building( #PCDATA )>
  <!ELEMENT budget( #PCDATA )>
  <!ELEMENT course id ( #PCDATA )>
  <!ELEMENT title ( #PCDATA )>
  <!ELEMENT credits( #PCDATA )>
  <!ELEMENT IID( #PCDATA )>
  <!ELEMENT name( #PCDATA )>
  <!ELEMENT salary( #PCDATA )>
]>
```

How to use DTD with XML: Option 1

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE mytvseries[
        <!ELEMENT mytvseries (tvseries*)>
        <!ELEMENT tvseries (name, actor+)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT actor (name, birthdate)>
        <!ELEMENT birthdate (#PCDATA)>
]>
<mytvseries>
  <tvseries>
    <name>Breaking Bad</name>
        <actor>
      <name>Bryan Cranston</name>
      <birthdate>3/7/1956</birthdate>
    </actor>
    <actor>
      <name>Aaron Paul</name>
      <birthdate>8/27/1979</birthdate>
    </actor>
  </tvseries>…
</mytvseries>
```