Dr Simon D’Alfonso

THE UNIVERSITY OF MELBOURNE

INFO90002 Database Systems & Information Modelling

Week 06

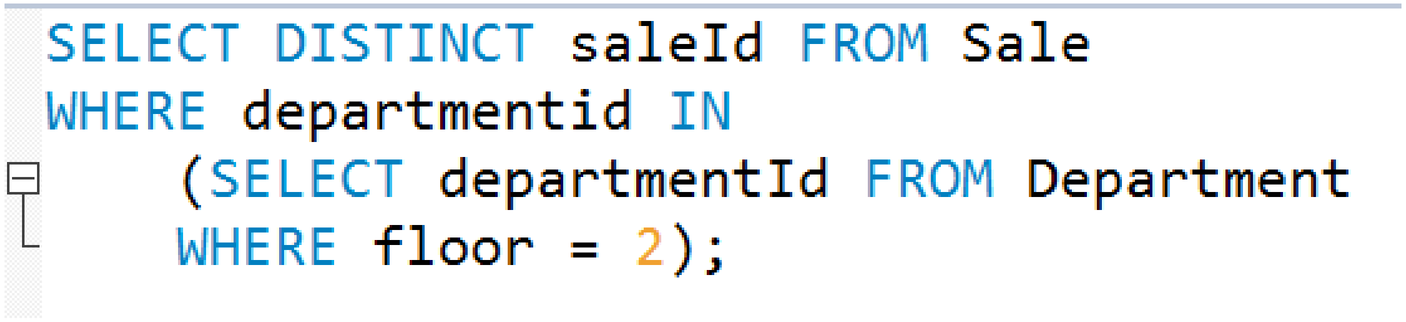
SQL 2

Today's Session…

- Nested/Sub queries
- DML
  - Comparison & Logic Operators, Set Operations, Multiple record INSERTs, INSERT from a table, UPDATE, DELETE, REPLACE
- DDL
  - ALTER and DROP, TRUNCATE, RENAME
- DCL
  - GRANT and REVOKE
- Views

SQL – Subqueries / nested queries

- Select allows you to nest *sub-queries* inside the main or “outer” query
- A nested query is simply another Select query you write to produce a table of data
  - remember that all select queries return a “table”
- A common use of sub-queries is to perform tests
  - set membership, set comparisons
- Often there is an equivalent Join query
- Put the subquery inside round brackets



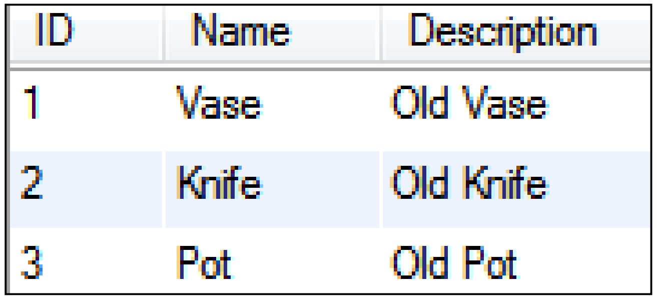
```
SELECT DISTINCT saleId FROM Sale
WHERE departmentid IN
    (SELECT departmentId FROM Department
    WHERE floor = 2);
```

Comparison Operators for SubQueries

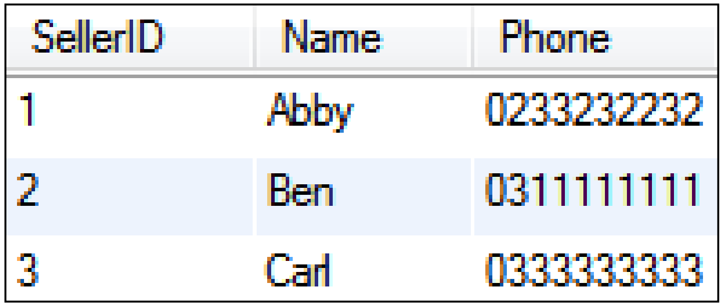
- IN / NOT IN
  - is the value a member of the set returned by the Subquery?
- ALL
  - true if all values returned meet the condition
- WHERE [NOT] EXISTS
  - true if the subquery yields any [/ no] results

Set Comparison examples

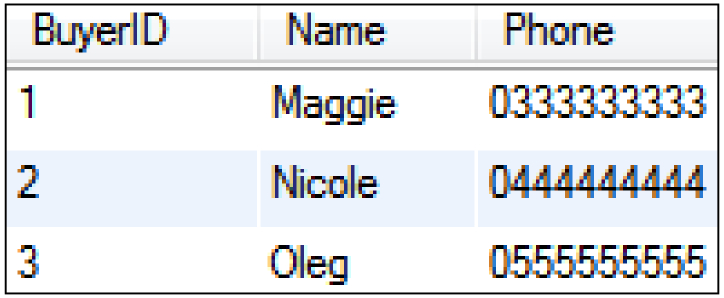
- auction example: Buyer, Seller, Artefact, Offer tables



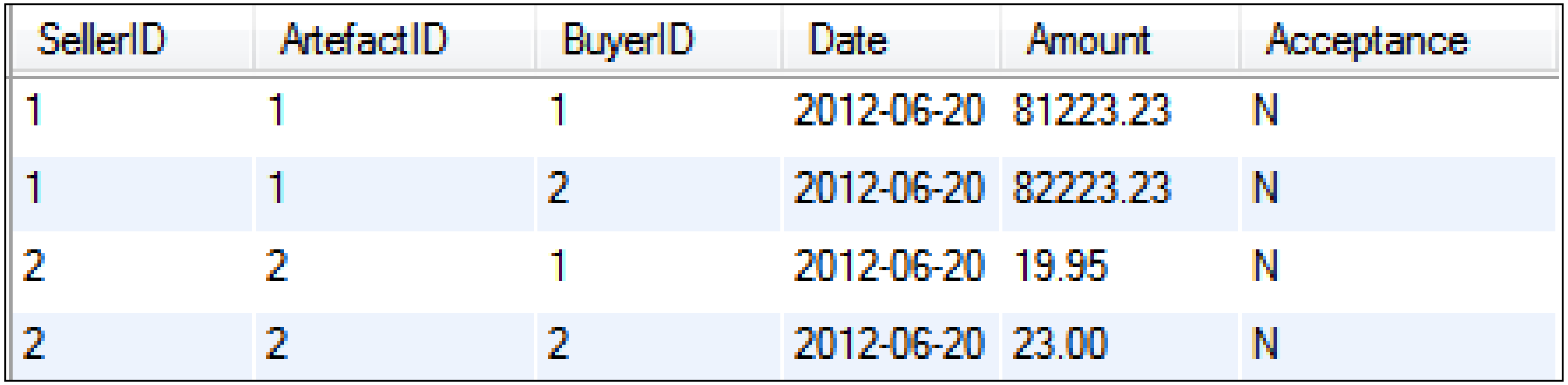
ID	Name	Description
1	Vase	Old Vase
2	Knife	Old Knife
3	Pot	Old Pot



SellerID	Name	Phone
1	Abby	0233232232
2	Ben	0311111111
3	Carl	0333333333



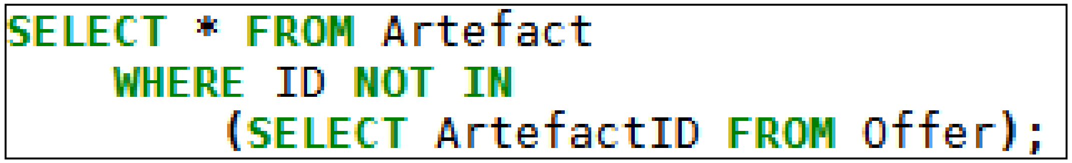
BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555



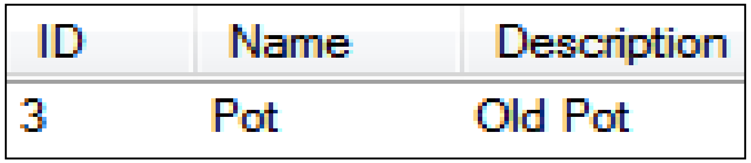
SellerID	ArtefactID	BuyerID	Date	Amount	Acceptance
1	1	1	2012-06-20	81223.23	N
1	1	2	2012-06-20	82223.23	N
2	2	1	2012-06-20	19.95	N
2	2	2	2012-06-20	23.00	N

Set queries

- which Artefacts *don't have* offers made on them?

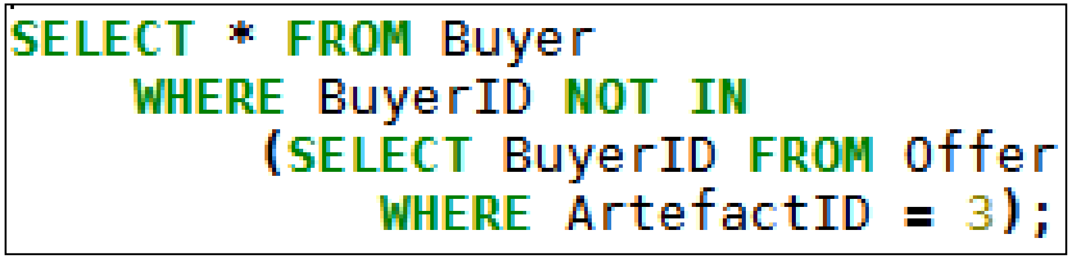


```
SELECT * FROM Artefact
    WHERE ID NOT IN
        (SELECT ArtefactID FROM Offer);
```

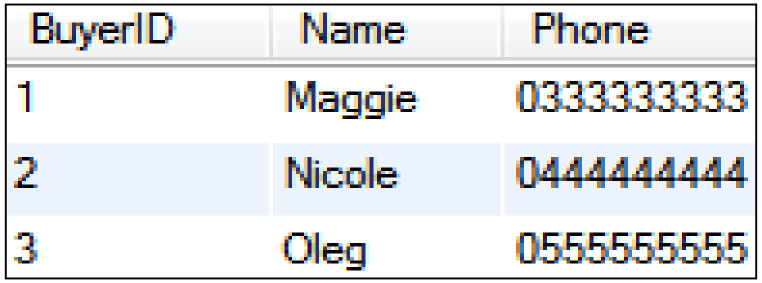


ID	Name	Description
3	Pot	Old Pot

- which Buyers *haven't* made a bid for Artefact 3?

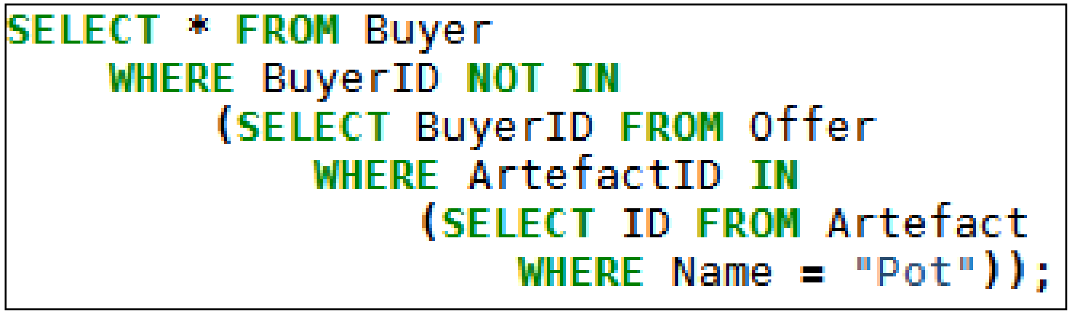


```
SELECT * FROM Buyer
    WHERE BuyerID NOT IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID = 3);
```

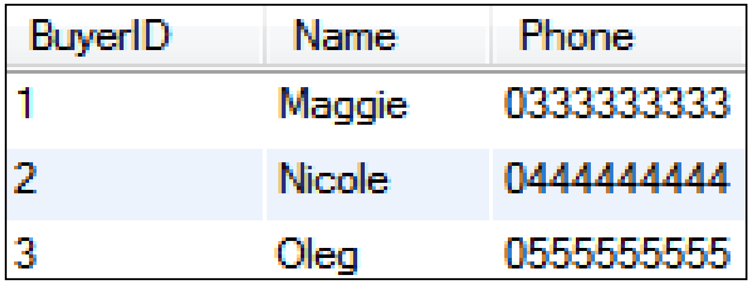


BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

- which Buyers *haven't* made a bid for the "Pot" Artefact?



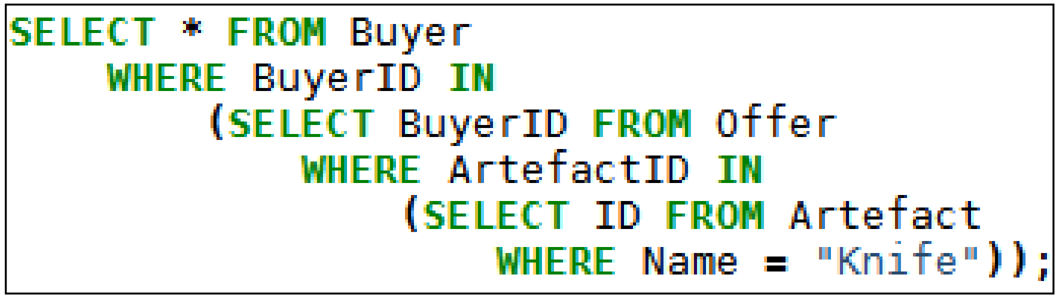
```
SELECT * FROM Buyer
    WHERE BuyerID NOT IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID IN
                (SELECT ID FROM Artefact
                    WHERE Name = "Pot"));
```



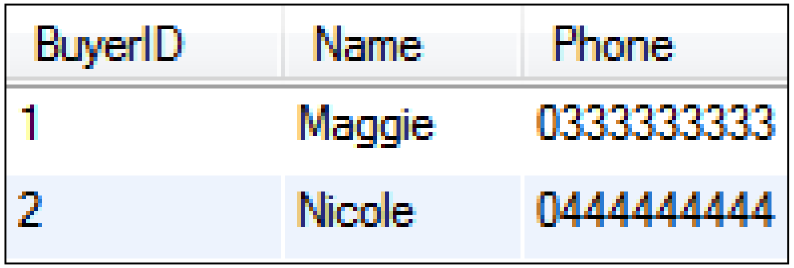
BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

Multiple subquery

- which Buyers have made a bid for the “Knife” Artefact?



```
SELECT * FROM Buyer
    WHERE BuyerID IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID IN
                (SELECT ID FROM Artefact
                    WHERE Name = "Knife"));
```



BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444

There is often an equivalent Join that will achieve the same result. The above is equivalent to:

```
SELECT Buyer.*
FROM Buyer NATURAL JOIN Offer NATURAL JOIN Artefact
WHERE Artefact.name = 'Knife' ;
```

THE UNIVERSITY OF MELBOURNE

Aggregate functions

These functions operate on a set of values (e.g. in a column of a table) and return a single value

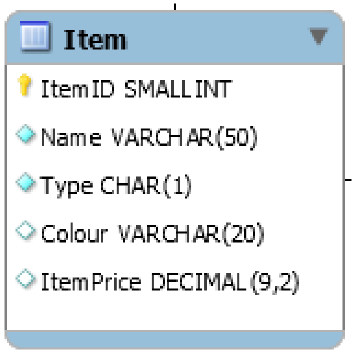
- AVG()
  - Average value
- MIN()
  - Minimum value
- MAX()
  - Maximum value
- COUNT()
  - Number of values
- SUM()
  - Sum of values

- and there are others …
  - http://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html
- These ignore null values, and return null if all values are null.
- But COUNT(*) counts the rows not the values,
  and thus even if the value is NULL it is still counted.

THE UNIVERSITY OF MELBOURNE

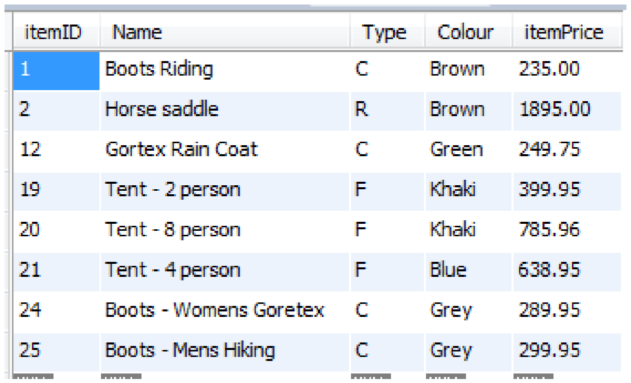
Finding above average

- Consider the Item table in our labs database
- Which items have a price that is higher than the average?



```
SELECT * FROM Item
WHERE itemPrice >
    (SELECT AVG(itemPrice) FROM Item);
```

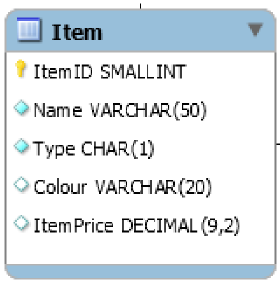
234.766400



itemID	Name	Type	Colour	itemPrice
1	Boots Riding	C	Brown	235.00
2	Horse saddle	R	Brown	1895.00
12	Gortex Rain Coat	C	Green	249.75
19	Tent - 2 person	F	Khaki	399.95
20	Tent - 8 person	F	Khaki	785.96
21	Tent - 4 person	F	Blue	638.95
24	Boots - Womens Goretex	C	Grey	289.95
25	Boots - Mens Hiking	C	Grey	299.95

Finding a maximum

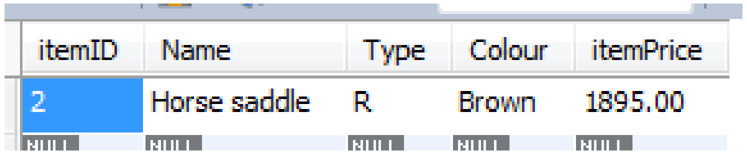
- Which item has the highest cost?



Item
ItemID SMALLINT
Name VARCHAR(50)
Type CHAR(1)
Colour VARCHAR(20)
ItemPrice DECIMAL(9,2)

```
SELECT * FROM Item
WHERE itemPrice =
      (SELECT MAX(itemPrice) FROM Item);
```

1,895



itemID	Name	Type	Colour	itemPrice
2	Horse saddle	R	Brown	1895.00
NULL	NULL	NULL	NULL	NULL

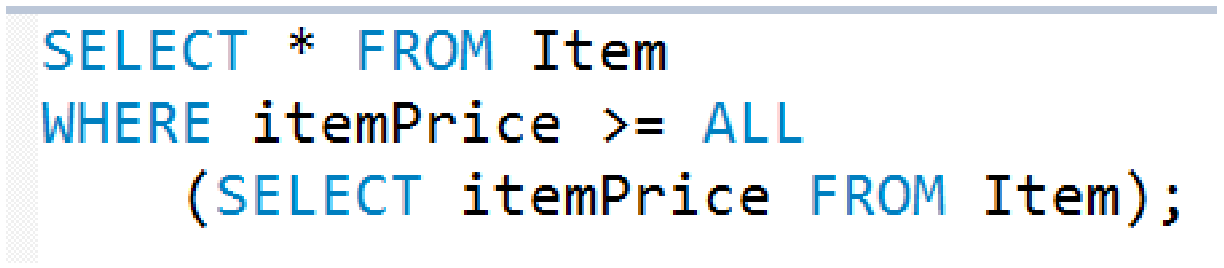
```
SELECT * FROM Item
ORDER BY itemprice DESC
LIMIT 1;
```

- Will these two methods always give the same answer?

THE UNIVERSITY OF MELBOURNE

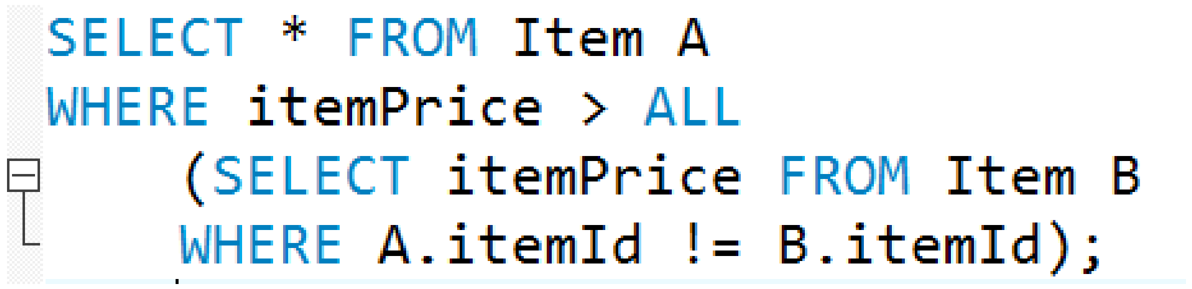
Finding a maximum

- another method



```
SELECT * FROM Item
WHERE itemPrice >= ALL
    (SELECT itemPrice FROM Item);
```

- and another: a “correlated subquery”



```
SELECT * FROM Item A
WHERE itemPrice > ALL
    (SELECT itemPrice FROM Item B
    WHERE A.itemId != B.itemId);
```

Case and calculation

- SQL keywords are *not* case-sensitive.
  - the traditional convention is to CAPITALISE them for clarity
- Table names *are* case sensitive in Unix, but not Windows
  (and possibly *not* case-sensitive if you use the InnoDb storage engine)
  - Account <> account <> ACCOUNT (in Unix)
- Column names are *not* case-sensitive
  - ACCOUNTID == AccountID == AcCoUnTID
- Case-sensitivity of DATA (‘strings in quotes’) depends on character set used.
  (The default 'latin1' set is *not* case-sensitive.)
- SQL handles expressions including maths:
  - SELECT 1*2+3/4-5;
  - SELECT now();

Comparison and Logic Operators

- Comparison

Operator	Description
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<> OR !=	Not equal to

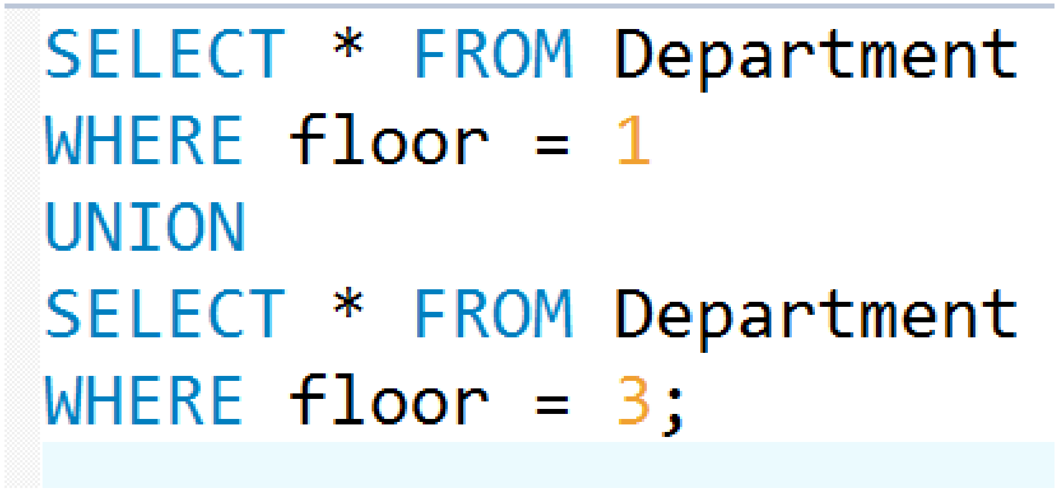
- Logic
  - SQL supports AND, NOT, OR logical operators
    - SELECT * FROM Furniture
      WHERE ((Type= ‘Chair’ AND Colour = ‘Black’)
      OR NOT (Type = ‘Lamp’ AND Colour = ‘White’));

Set Operations

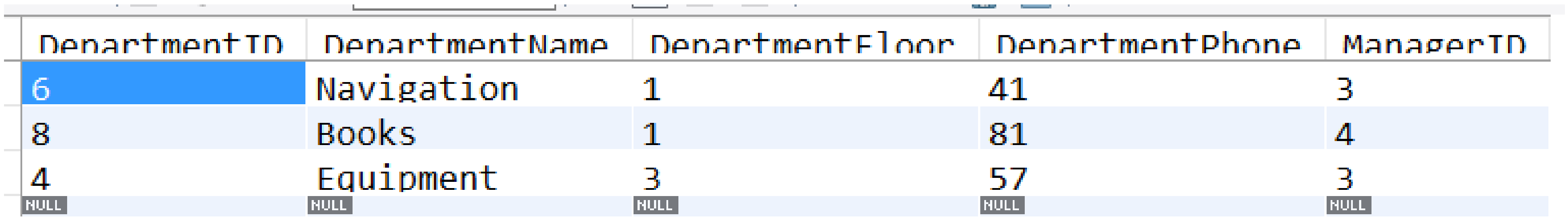
- We can combine results from two or more queries that return *the same number of columns* - although it usually only makes sense if they are the *same columns*.
- UNION
  - Show all rows returned from the queries, without duplicates
- INTERSECT
  - Show only rows that are common in the queries
- EXCEPT
  - Show only rows that are different in the queries
- [UNION/INTERSECT/EXCEPT] ALL
  - If you want duplicate rows shown in the results you need to use the ALL keyword, e.g. UNION ALL.

- In MySQL only UNION and UNION ALL are supported

UNION example



```
SELECT * FROM Department
WHERE floor = 1
UNION
SELECT * FROM Department
WHERE floor = 3;
```



DepartmentID	DepartmentName	DepartmentFloor	DepartmentPhone	ManagerID
6	Navigation	1	41	3
8	Books	1	81	4
4	Equipment	3	57	3
NULL	NULL	NULL	NULL	NULL

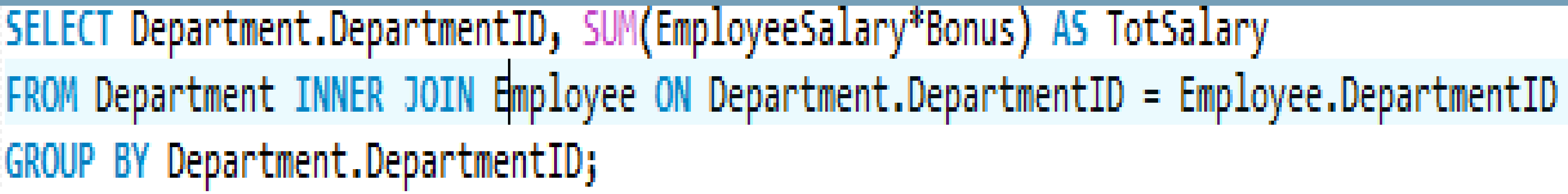
(what if the subsets overlap?)

© University of Melbourne 2019

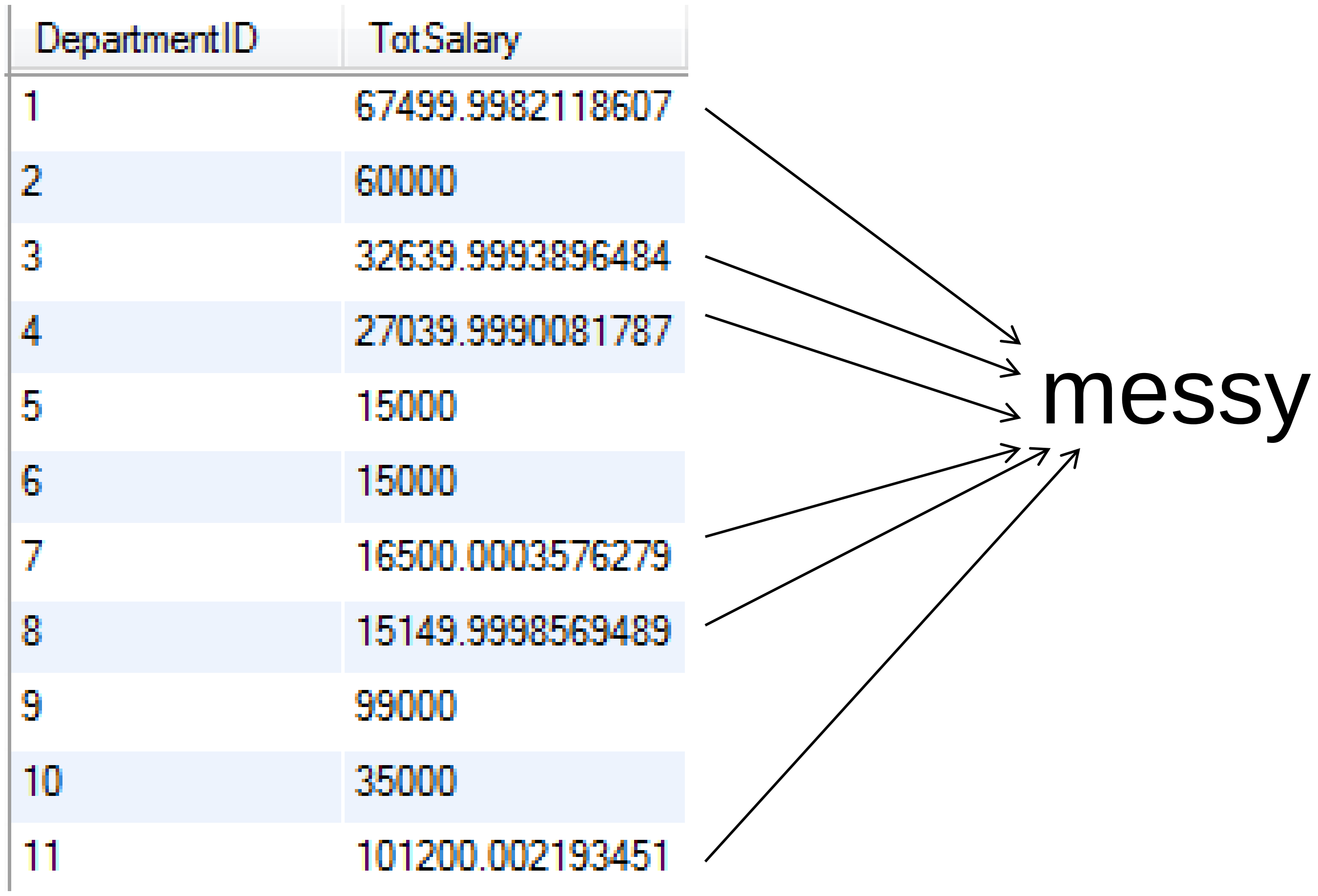
Formatting the result

- FORMAT()
  - changes format of output of Select
  - e.g. FORMAT (N, D)
    - N: A number which may be an integer, a decimal or a float.
    - D: How many decimals the output contains
    - FORMAT(123456.1234, 2) gives ‘123,456.12’
- CAST()
  - changes data type of output
  - e.g. CAST (Expression AS Type)
    - CAST(“1234.55” AS UNSIGNED) Gives 1235
    - CAST("1234.55" AS DECIMAL(7,1)) Gives 1234.6
    - Valid types include
      - BINARY[(N)], CHAR[(N)], DATE, DATETIME, DECIMAL[(M[,D])], SIGNED, TIME, UNSIGNED

Output without Format



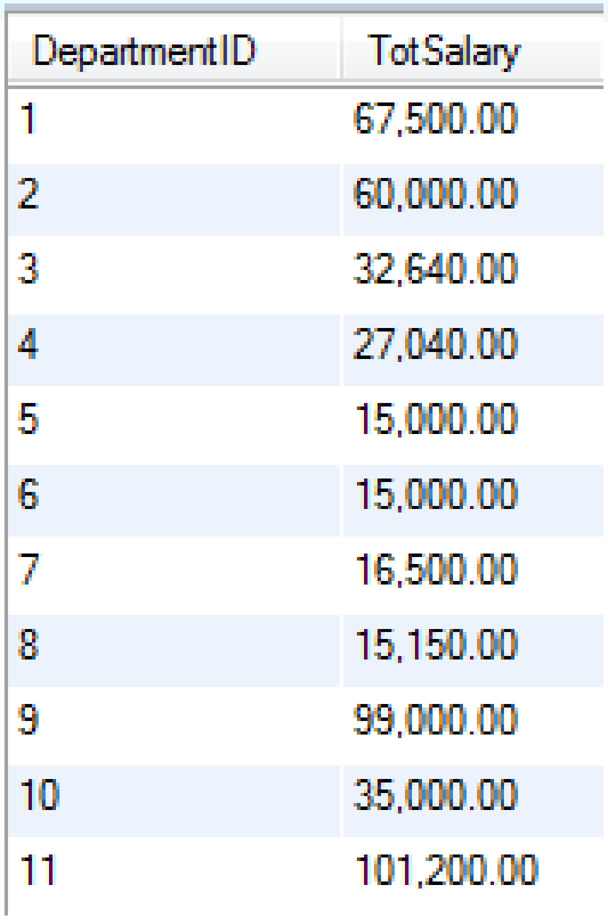
```
SELECT Department.DepartmentID, SUM(EmployeeSalary*Bonus) AS TotSalary
FROM Department INNER JOIN Employee ON Department.DepartmentID = Employee.DepartmentID
GROUP BY Department.DepartmentID;
```



DepartmentID	TotSalary
1	67499.9982118607
2	60000
3	32639.9993896484
4	27039.9990081787
5	15000
6	15000
7	16500.0003576279
8	15149.9998569489
9	99000
10	35000
11	101200.002193451

Formatting output (Format)

```sql
SELECT Department.DepartmentID, FORMAT(SUM(EmployeeSalary*Bonus),2) AS TotSalary
FROM Department INNER JOIN Employee ON Department.DepartmentID = Employee.DepartmentID
GROUP BY Department.DepartmentID;
```

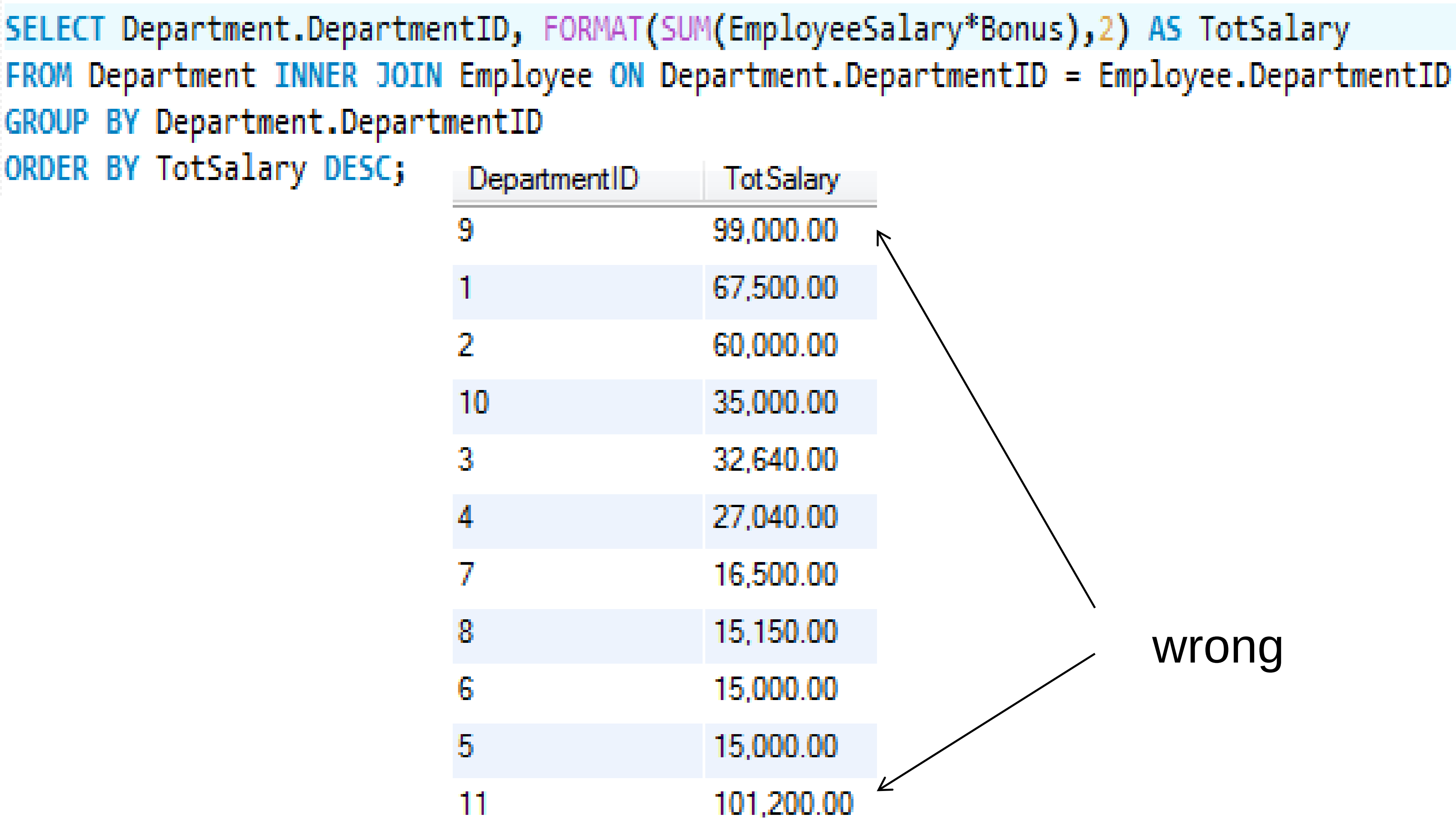


DepartmentID	TotSalary
1	67,500.00
2	60,000.00
3	32,640.00
4	27,040.00
5	15,000.00
6	15,000.00
7	16,500.00
8	15,150.00
9	99,000.00
10	35,000.00
11	101,200.00

but Format() converts numbers to strings …

what happens now
if we sort by TotSalary?

Formatting output (Format)



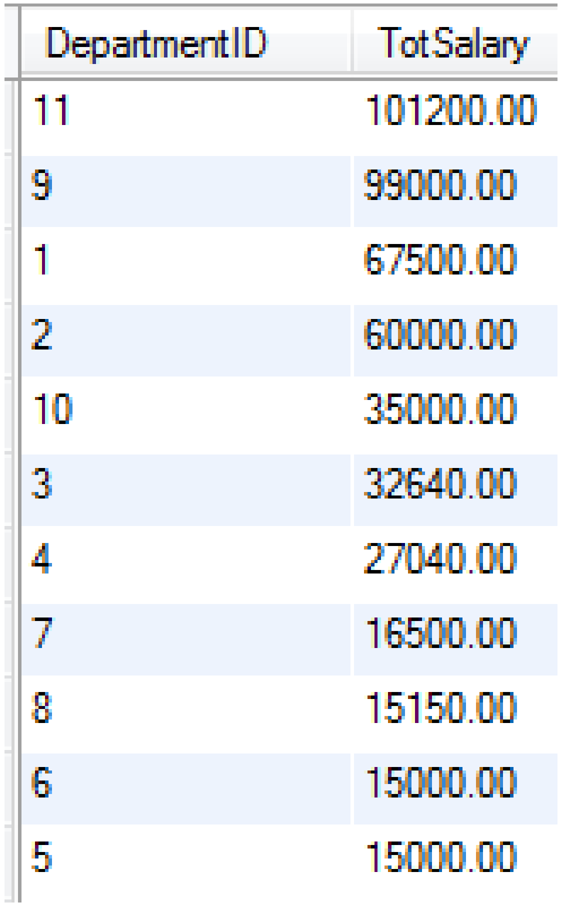
```
SELECT Department.DepartmentID, FORMAT(SUM(EmployeeSalary*Bonus),2) AS TotSalary
FROM Department INNER JOIN Employee ON Department.DepartmentID = Employee.DepartmentID
GROUP BY Department.DepartmentID
ORDER BY TotSalary DESC;
```

DepartmentID	TotSalary
9	99,000.00
1	67,500.00
2	60,000.00
10	35,000.00
3	32,640.00
4	27,040.00
7	16,500.00
8	15,150.00
6	15,000.00
5	15,000.00
11	101,200.00

wrong

Formatting output (Cast)

```
SELECT Department.DepartmentID, CAST(SUM(EmployeeSalary*Bonus) AS DECIMAL(9,2)) AS TotSalary
FROM Department INNER JOIN Employee ON Department.DepartmentID = Employee.DepartmentID
GROUP BY Department.DepartmentID
ORDER BY TotSalary DESC;
```



DepartmentID	TotSalary
11	101200.00
9	99000.00
1	67500.00
2	60000.00
10	35000.00
3	32640.00
4	27040.00
7	16500.00
8	15150.00
6	15000.00
5	15000.00

These are numbers, so ordering works again

THE UNIVERSITY OF MELBOURNE

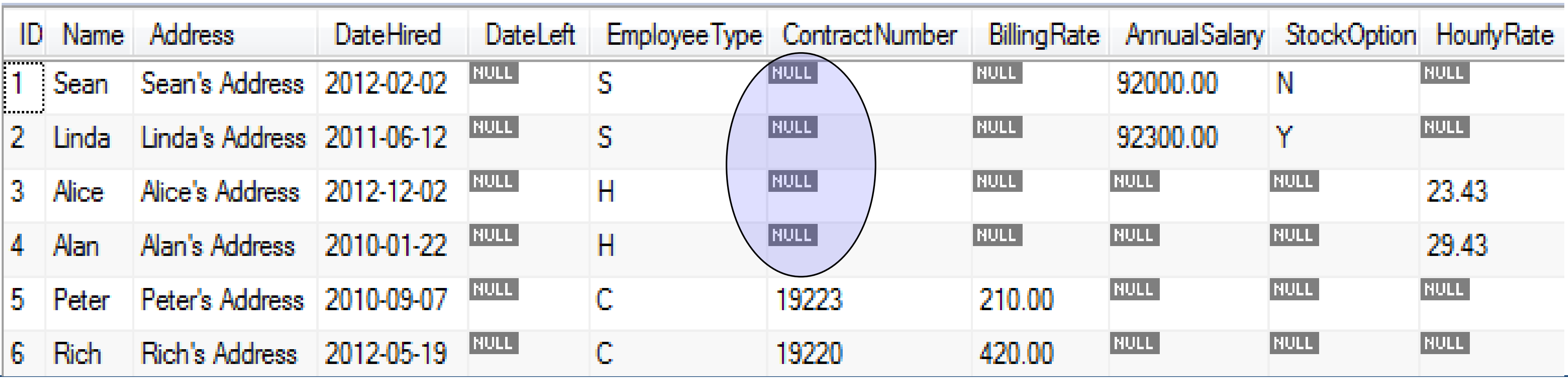
Other useful functions

- IFNULL()
  - Can convert a null to a zero (can be useful in calculations)
    - SELECT 1 + IFNULL(wagevalue, 0)
    - gives 1+0 for null fields, and 1+wagevalue for non null fields
    - failure to do this results in a NULL answer for values where wagevalue is NULL

(example on next two slides)

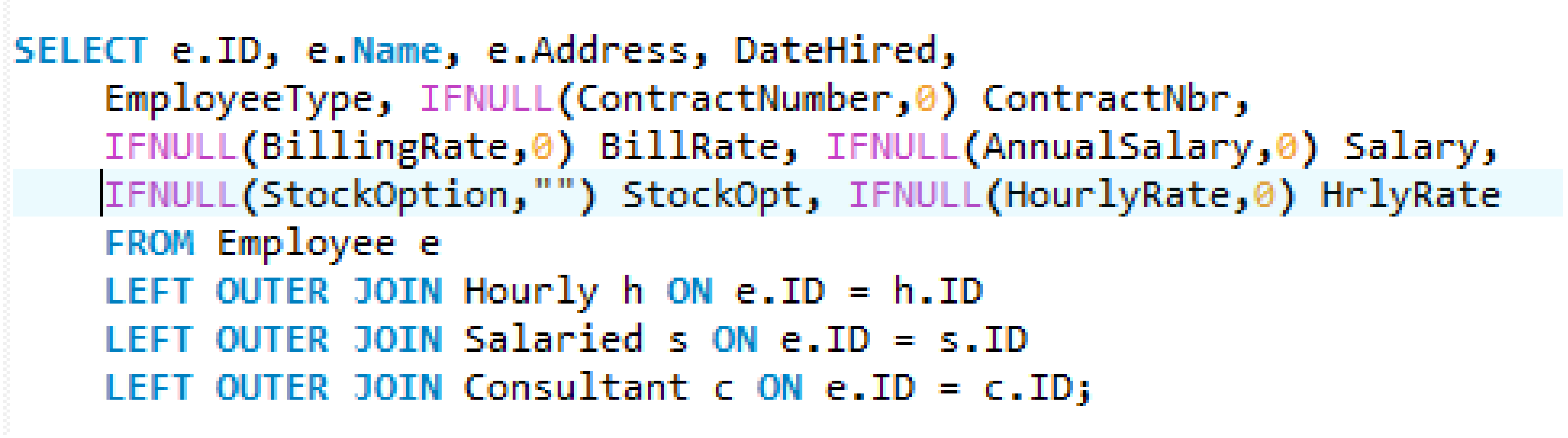
Don't want NULLs in output?

```
SELECT e.ID, e.Name, e.Address, DateHired, DateLeft,
       EmployeeType, ContractNumber, BillingRate,
       AnnualSalary, StockOption, HourlyRate
    FROM Employee e
    LEFT OUTER JOIN Hourly h ON e.ID = h.ID
    LEFT OUTER JOIN Salaried s ON e.ID = s.ID
    LEFT OUTER JOIN Consultant c ON e.ID = c.ID;
```

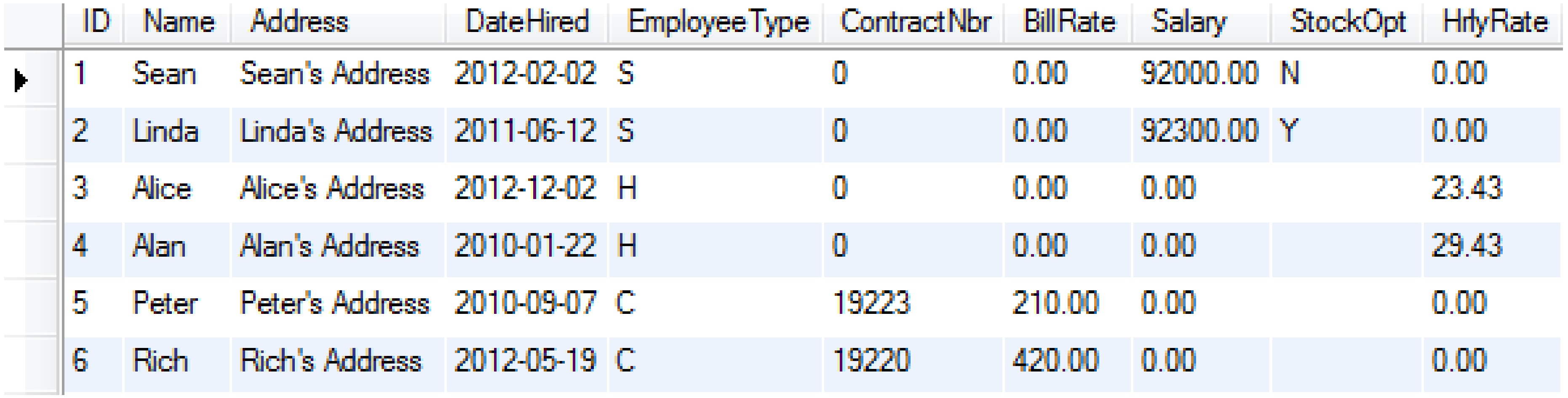


	ID	Name	Address	DateHired	DateLeft	EmployeeType	ContractNumber	BillingRate	AnnualSalary	StockOption	HourlyRate
1	1	Sean	Sean's Address	2012-02-02	NULL	S	NULL	NULL	92000.00	N	NULL
2	2	Linda	Linda's Address	2011-06-12	NULL	S	NULL	NULL	92300.00	Y	NULL
3	3	Alice	Alice's Address	2012-12-02	NULL	H	NULL	NULL	NULL	NULL	23.43
4	4	Alan	Alan's Address	2010-01-22	NULL	H	NULL	NULL	NULL	NULL	29.43
5	5	Peter	Peter's Address	2010-09-07	NULL	C	19223	210.00	NULL	NULL	NULL
6	6	Rich	Rich's Address	2012-05-19	NULL	C	19220	420.00	NULL	NULL	NULL

Nulls replaced with zeros



```
SELECT e.ID, e.Name, e.Address, DateHired,
    EmployeeType, IFNULL(ContractNumber,0) ContractNbr,
    IFNULL(BillingRate,0) BillRate, IFNULL(AnnualSalary,0) Salary,
    IFNULL(StockOption,"") StockOpt, IFNULL(HourlyRate,0) HrlyRate
    FROM Employee e
    LEFT OUTER JOIN Hourly h ON e.ID = h.ID
    LEFT OUTER JOIN Salaried s ON e.ID = s.ID
    LEFT OUTER JOIN Consultant c ON e.ID = c.ID;
```



ID	Name	Address	DateHired	EmployeeType	ContractNbr	BillRate	Salary	StockOpt	HrlyRate
1	Sean	Sean's Address	2012-02-02	S	0	0.00	92000.00	N	0.00
2	Linda	Linda's Address	2011-06-12	S	0	0.00	92300.00	Y	0.00
3	Alice	Alice's Address	2012-12-02	H	0	0.00	0.00		23.43
4	Alan	Alan's Address	2010-01-22	H	0	0.00	0.00		29.43
5	Peter	Peter's Address	2010-09-07	C	19223	210.00	0.00		0.00
6	Rich	Rich's Address	2012-05-19	C	19220	420.00	0.00		0.00

© University of Melbourne 2019

Other useful functions

- LOWER() / UPPER()
  - Change string to lower / upper case
    - e.g. SELECT LOWER(‘That’) gives ‘that’
    - SELECT UPPER(‘That’) gives ‘THAT’

- LEFT() / RIGHT()
  - Returns the leftmost / rightmost N characters from a string
    - e.g. SELECT LEFT(‘This is a test’, 6) gives “This i”
    - e.g. SELECT RIGHT(‘This is a test’, 6) gives “a test”

- Date and time functions
  - http://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html
    - including DATEDIFF(), TIMEDIFF(), NOW() or TIMESTAMP(), CURDATE(), CURTIME()

More on INSERT

- Inserting records from another table
  - Note: table must already exist

```
INSERT INTO NewEmployee
    SELECT * FROM Employee;
```

- Insert multiple rows

```
INSERT INTO Employee VALUES
    (DEFAULT, "A", "A's Addr", "2012-02-02", NULL, "S"),
    (DEFAULT, "B", "B's Addr", "2012-02-02", NULL, "S"),
    (DEFAULT, "C", "C's Addr", "2012-02-02", NULL, "S");
```

```
INSERT INTO Employee
    (Name, Address, DateHired, EmployeeType)
    VALUES
        ("D", "D's Addr", "2012-02-02", "C"),
        ("E", "E's Addr", "2012-02-02", "C"),
        ("F", "F's Addr", "2012-02-02", "C");
```

More on UPDATE

- Be careful to specify a WHERE clause
  - unless you want it to operate on EVERY row in the table

```
UPDATE Hourly
    SET HourlyRate = HourlyRate * 1.10;
```

- Increase salaries greater than $100k by 10% and all other salaries by 5%

```
UPDATE Salaried
    SET AnnualSalary = AnnualSalary * 1.05
    WHERE AnnualSalary <= 100000;
UPDATE Salaried
    SET AnnualSalary = AnnualSalary * 1.10
    WHERE AnnualSalary > 100000;
```

- Any problems with this?

Flow Control using CASE

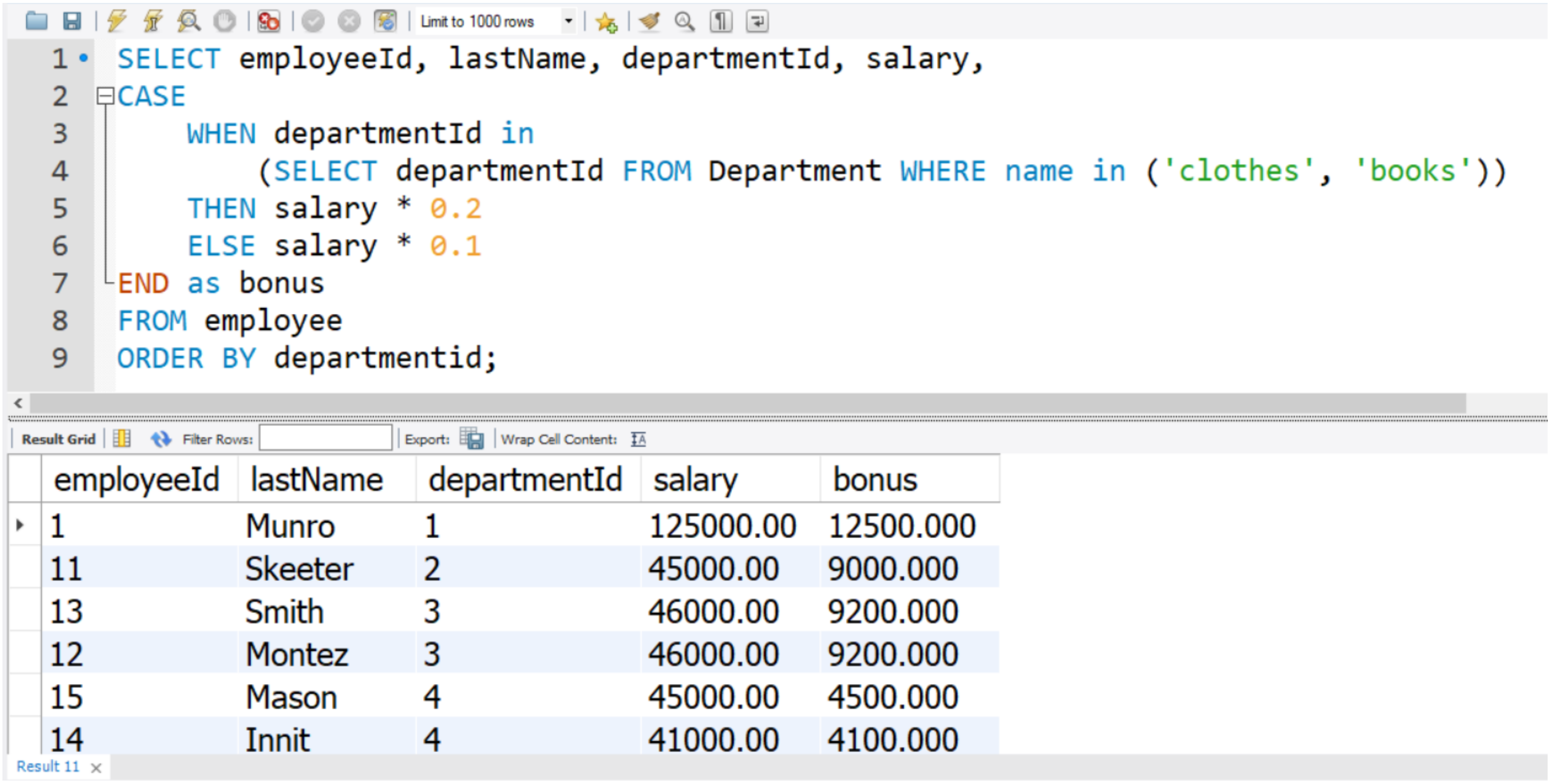
- A better solution is to use the CASE expression

```
UPDATE Salaried
    SET AnnualSalary =
        CASE
            WHEN AnnualSalary <= 100000
            THEN AnnualSalary * 1.05
            ELSE AnnualSalary * 1.10
        END;
```

- now we process each row independently, one at a time

Flow Control using CASE

- CASE can also be used in SELECT statements
- e.g “Calculate our annual bonuses. Give each employee a 10% bonus, except those who work work in Clothes or Books, who get 20%.”

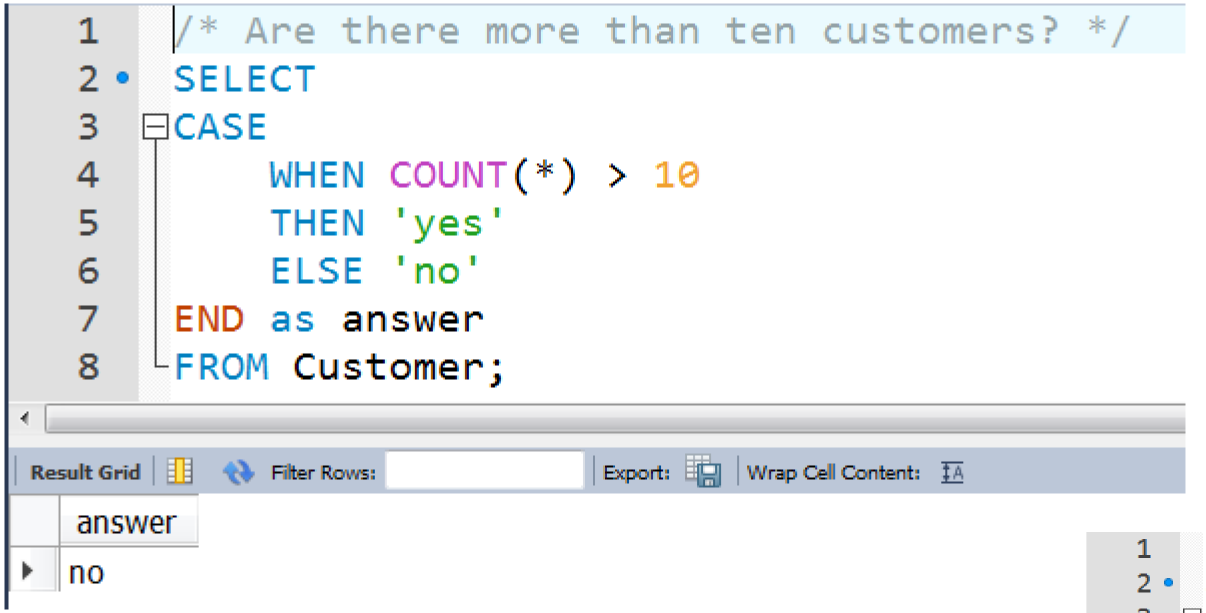


```
SELECT employeeId, lastName, departmentId, salary,
CASE
    WHEN departmentId in
        (SELECT departmentId FROM Department WHERE name in ('clothes', 'books'))
    THEN salary * 0.2
    ELSE salary * 0.1
END as bonus
FROM employee
ORDER BY departmentid;
```

employeeId	lastName	departmentId	salary	bonus
1	Munro	1	125000.00	12500.000
11	Skeeter	2	45000.00	9000.000
13	Smith	3	46000.00	9200.000
12	Montez	3	46000.00	9200.000
15	Mason	4	45000.00	4500.000
14	Innit	4	41000.00	4100.000

Yes/No Questions using CASE

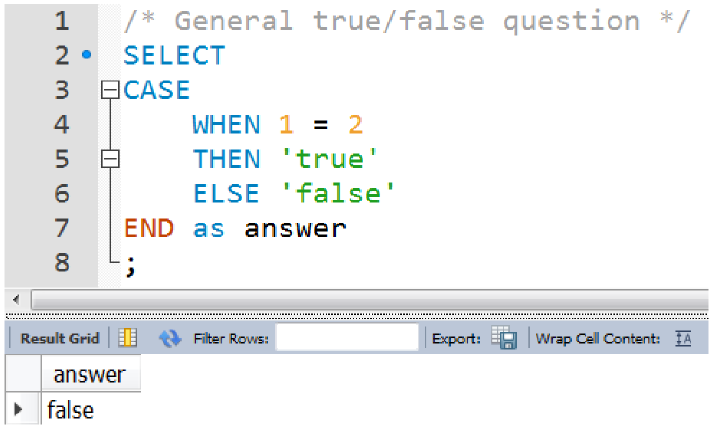
- You can use CASE to answer yes/no or true/false questions.
- e.g “Are there more than ten customers?”



```sql
/* Are there more than ten customers? */
SELECT
CASE
     WHEN COUNT(*) > 10
     THEN 'yes'
     ELSE 'no'
END as answer
FROM Customer;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

```
answer
no
```



```sql
/* General true/false question */
SELECT
CASE
     WHEN 1 = 2
     THEN 'true'
     ELSE 'false'
END as answer
;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

```
answer
false
```

DELETE, REPLACE

- REPLACE
  - REPLACE works the same as INSERT
    - EXCEPT that if an old row in a table has a key value the same as the new row, then it is overwritten…
- DELETE
  - be careful to use a WHERE clause … What does this do?

```
DELETE FROM Employee;
```

    - Usually you should do use a filter:

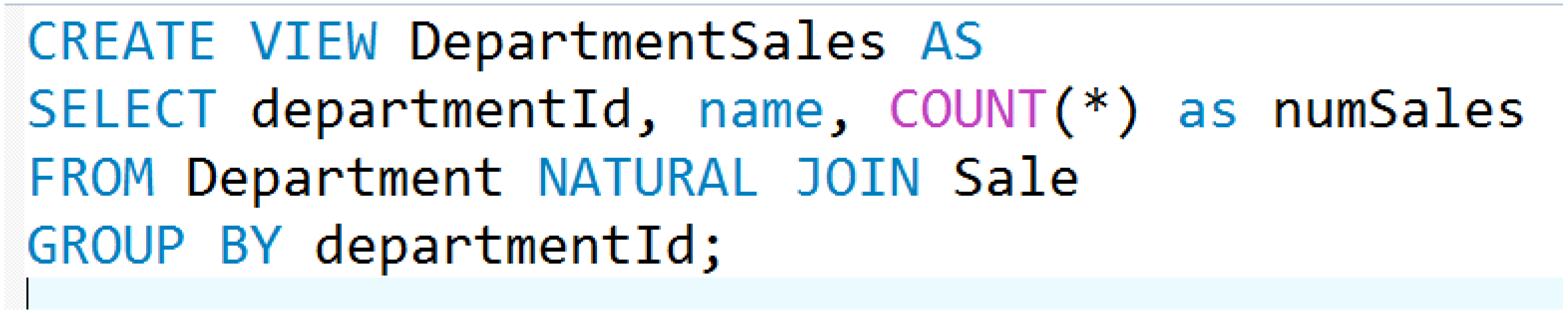
```
DELETE FROM Employee
    WHERE Name = "Grace";
```

  - If you delete a row that has rows in other tables dependent on it, either:
    - the dependent rows are deleted too, or
    - the dependent rows get ‘null’ or a default, or
    - your attempt to delete is blocked
    - you decide what action to take when you set up the tables
      - ON DELETE CASCADE or ON DELETE RESTRICT…

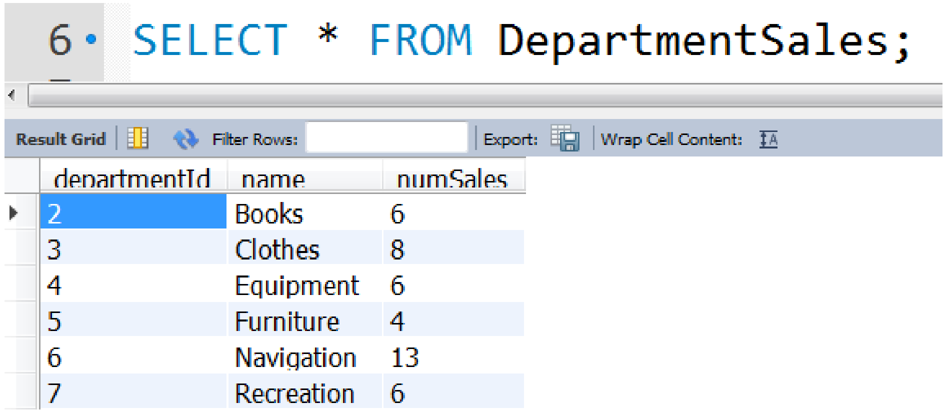
Views

- a View is a select statement that persists, and can be treated as though it were a table by other SQL statements
- Used to:
  - hide the complexity of queries from users
  - hide structure of data from users
  - hide data from users
    - different users use different views
      - e.g. allow someone to access employee table, but not salaries column
    - one way of improving database security
- To create a view…
  - **CREATE VIEW** nameofview **AS** validSelectStatement
  - its definition (but not its output) is stored in the database
  - can be used as though it is a table

CREATE VIEW example

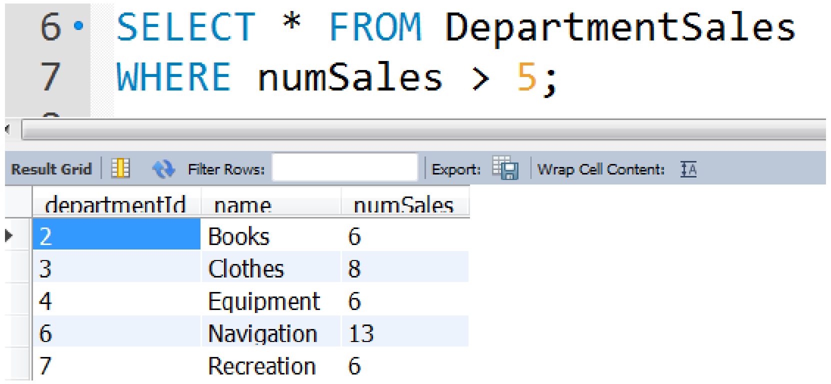


```sql
CREATE VIEW DepartmentSales AS
SELECT departmentId, name, COUNT(*) as numSales
FROM Department NATURAL JOIN Sale
GROUP BY departmentId;
```



```sql
SELECT * FROM DepartmentSales;
```

departmentId	name	numSales
2	Books	6
3	Clothes	8
4	Equipment	6
5	Furniture	4
6	Navigation	13
7	Recreation	6



```sql
SELECT * FROM DepartmentSales
WHERE numSales > 5;
```

departmentId	name	numSales
2	Books	6
3	Clothes	8
4	Equipment	6
6	Navigation	13
7	Recreation	6

© University of Melbourne 2019

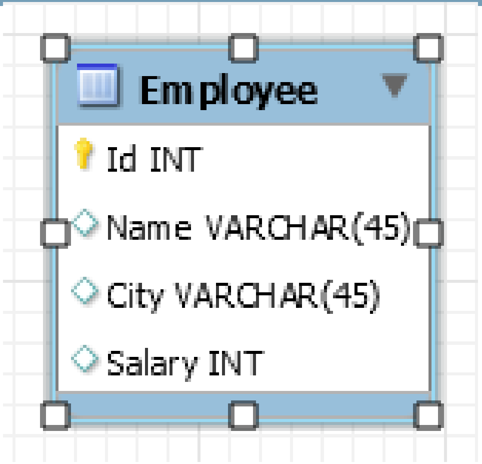
When can we Update or Insert a view?

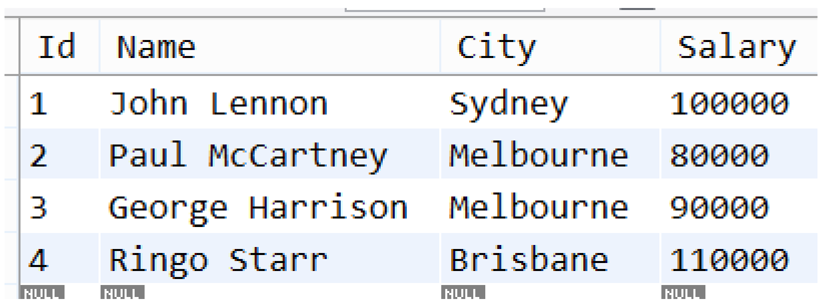
- Conditions that must be satisfied:
  - the select clause only contains attribute names
    - not expressions, aggregates or distinct
  - any attributes not listed in the select clause can be set to null
  - the query does not have a group by or having clause

- MySQL conditions for updatable views are quite stringent
  - see http://dev.mysql.com/doc/refman/5.0/en/view-updatability.html

Updating a View: example

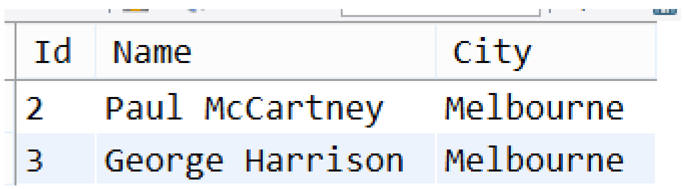
Underlying base table
->





Id	Name	City	Salary
1	John Lennon	Sydney	100000
2	Paul McCartney	Melbourne	80000
3	George Harrison	Melbourne	90000
4	Ringo Starr	Brisbane	110000
NULL	NULL	NULL	NULL

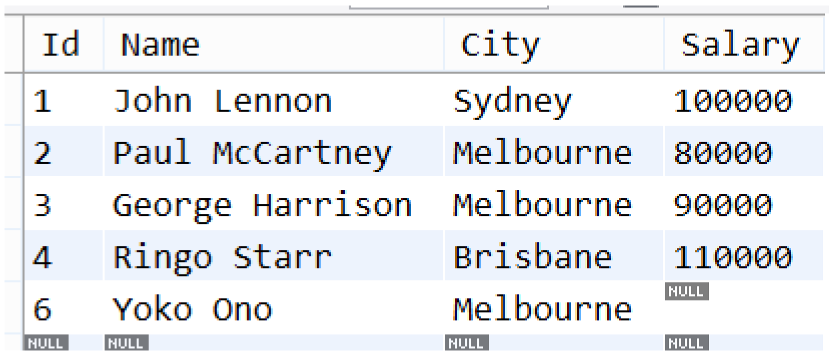
CREATE VIEW MelbRestricted AS
(SELECT id, name, city from Employee
WHERE city = 'Melbourne');



Id	Name	City
2	Paul McCartney	Melbourne
3	George Harrison	Melbourne

SELECT * FROM MelbRestricted;

INSERT INTO MelbRestricted VALUES
(null, 'Yoko Ono', 'Melbourne');



Id	Name	City	Salary
1	John Lennon	Sydney	100000
2	Paul McCartney	Melbourne	80000
3	George Harrison	Melbourne	90000
4	Ringo Starr	Brisbane	110000
6	Yoko Ono	Melbourne	NULL
NULL	NULL	NULL	NULL

More DDL commands

- (beyond CREATE)
- ALTER
  - Allows us to add or remove columns from a table
    - ALTER TABLE TableName ADD AttributeName AttributeType
    - ALTER TABLE TableName DROP AttributeName
      - not supported by all vendors (MySQL supports it)
- RENAME
  - Allows the renaming of tables
    - RENAME TABLE CurrentTableName TO NewTableName

More DDL commands

- TRUNCATE
  - like “DELETE FROM table” but it does more
  - differences are vendor-specific, see http://stackoverflow.com/questions/139630/whats-the-difference-between-truncate-and-delete-in-sql and https://dev.mysql.com/doc/refman/5.0/en/truncate-table.html
  - in MySQL, resets auto_increment PKs
  - cannot ROLL BACK a TRUNCATE command
    - have to get data back from backup…
- DROP
  - potentially DANGEROUS
    - Removes the table definition and the data in the table
      - There is NO UNDO COMMAND! (have to restore from backup)
    - DROP TABLE TableName

Data Control Language / Other Commands

- DCL
  - Users and permissions
    - **CREATE USER, DROP USER**
    - **GRANT, REVOKE**
    - **SET PASSWORD**
- Other commands offered
  - Database administration
    - **BACKUP TABLE, RESTORE TABLE**
    - **ANALYZE TABLE**
  - Miscellaneous
    - **DESCRIBE tablename**
    - **USE db_name**
  - MySql calls these
    ‘Database Administration Statements’

SQL Language in summary

- Data Definition Language (DDL)
  - To define and set up the database
  - CREATE, ALTER, DROP
    - Also TRUNCATE, RENAME
- Data Manipulation Language (DML)
  - To maintain and use the database
  - SELECT, INSERT, DELETE, UPDATE
    - MySQL also provides others…. eg REPLACE
- Data Control Language (DCL)
  - To control access to the database
    - GRANT, REVOKE
- Other Commands
  - Administer the database
  - Transaction Control