Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

INFO90002 Database Systems & Information Modelling

Week 04

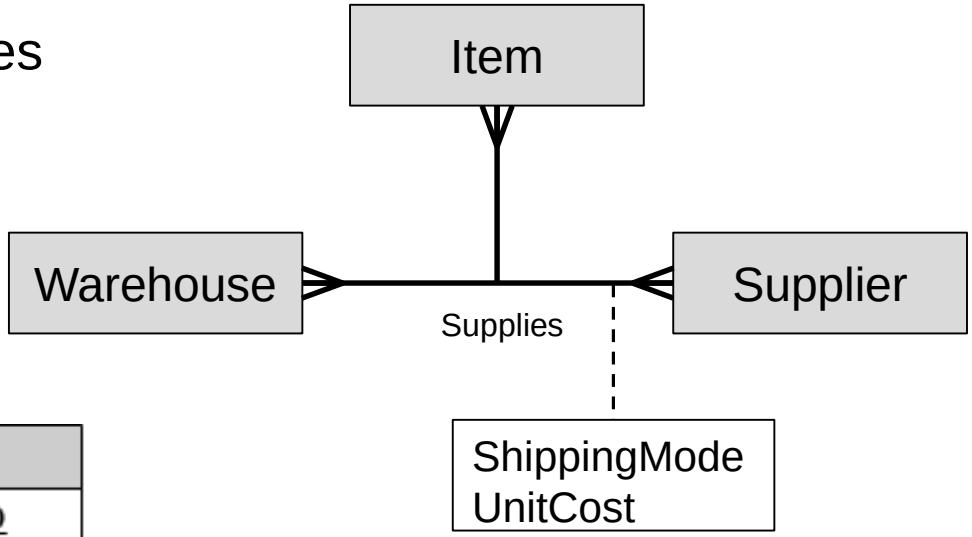
Data Modelling and SQL (3)

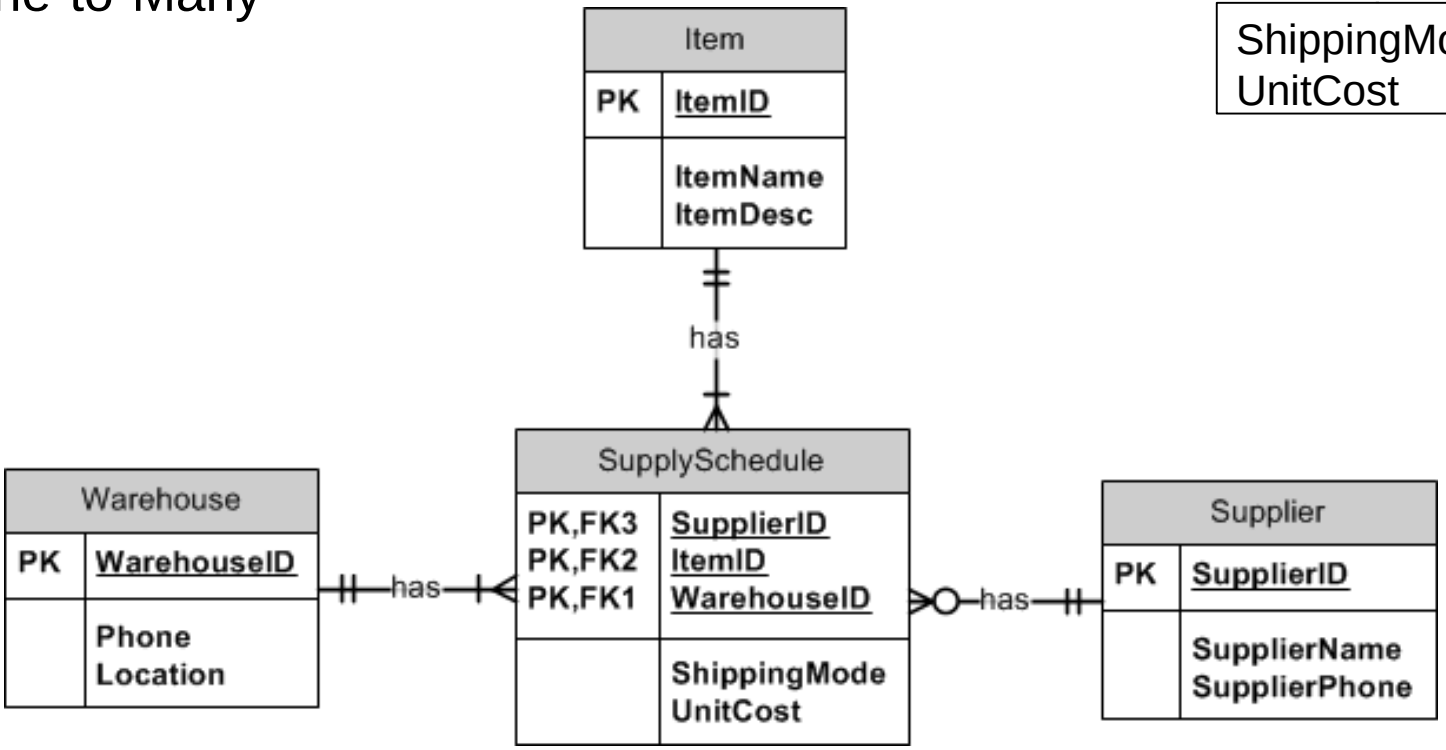
Today's Session…

- Data Modelling
  - Ternary relationships
    - 3 tables are involved
- SQL wrapup
  - 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

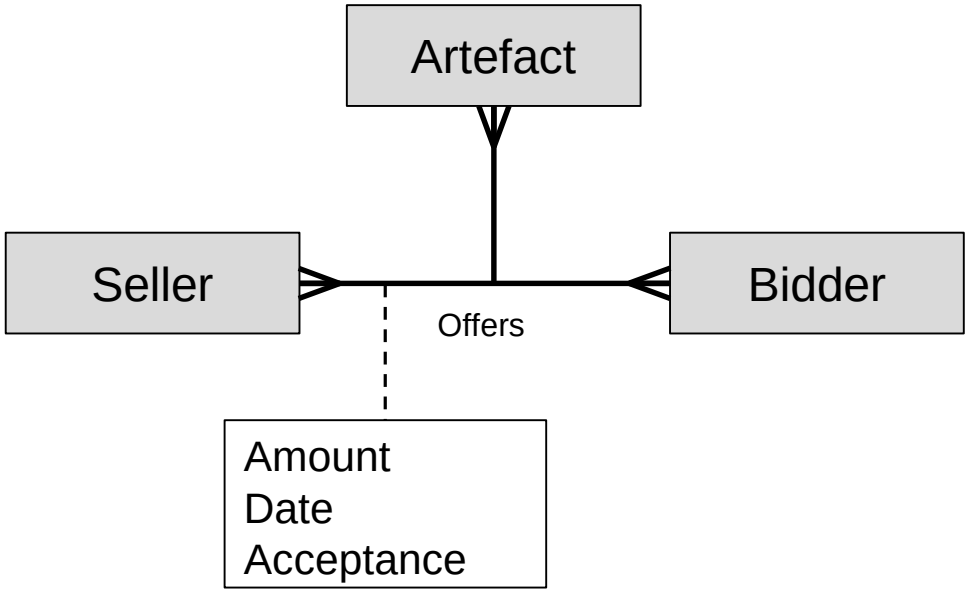
Ternary relationships

- Relationships between three entities
- Generate an Associative Entity
- Set up three One-to-Many relationships
- These are like any other One-to-Many



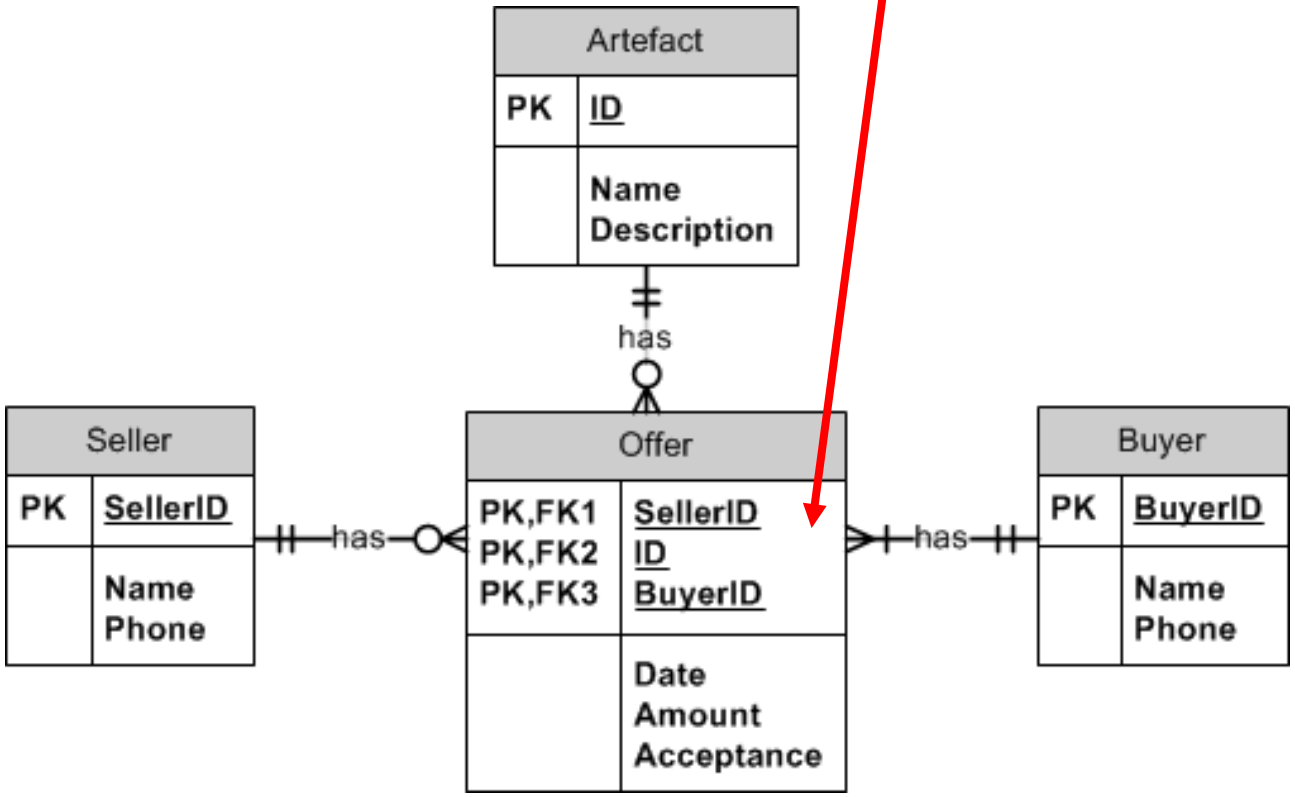


Ternary example – auction

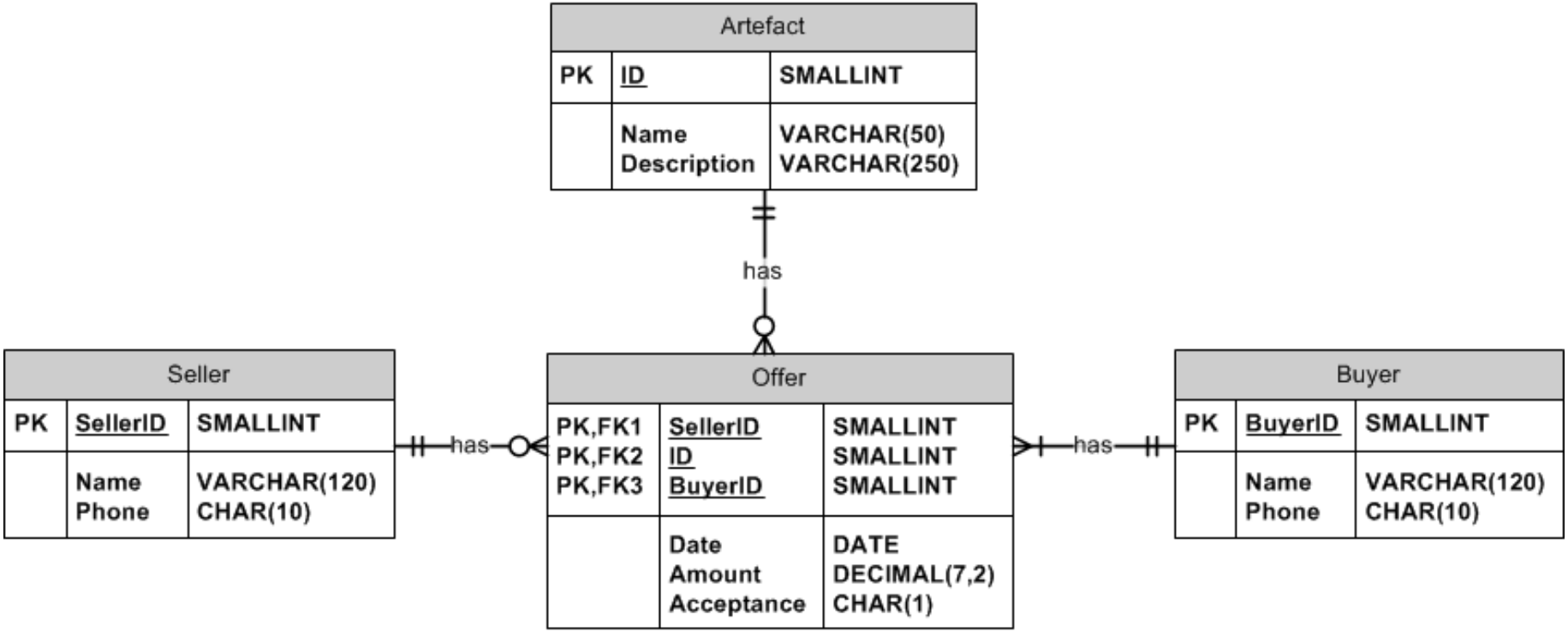


BUSINESS RULES:

- an Artefact can be sold several times by several different Sellers (over time)
- each Buyer can only make 1 offer to buy per Seller/Artefact combination



Auction Bids - Physical



Auction Bids – Table Creation

```
CREATE TABLE Seller (
  SellerID          smallint,
  Name              varchar(120) NOT NULL,
  Phone             char(10)     NOT NULL,
  PRIMARY KEY  (SellerID)
) ENGINE=InnoDB;
```

```
CREATE TABLE Buyer (
  BuyerID          smallint,
  Name              varchar(120) NOT NULL,
  Phone             char(10)     NOT NULL,
  PRIMARY KEY  (BuyerID)
) ENGINE=InnoDB;
```

```
CREATE TABLE Artefact (
  ID            smallint,
  Name          varchar(50)     NOT NULL,
  Description   varchar(250)    NOT NULL,
  PRIMARY KEY  (ID)
) ENGINE=InnoDB;
```

Auction Bids – Table Creation

```
CREATE TABLE Offer (
  SellerID          smallint     NOT NULL,
  ArtefactID        smallint     NOT NULL,
  BuyerID           smallint     NOT NULL,
  Date              DATE         NOT NULL,
  Amount            DECIMAL(12,2) NOT NULL,
  Acceptance        CHAR(1) NOT NULL DEFAULT "N",
  PRIMARY KEY  (SellerID, ArtefactID, BuyerID),
  FOREIGN KEY (ArtefactID) REFERENCES Artefact(ID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
  FOREIGN KEY (SellerID) REFERENCES Seller(SellerID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
  FOREIGN KEY (BuyerID) REFERENCES Buyer(BuyerID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Auction Bids – Data Creation

```
INSERT INTO Seller VALUES (1, "Abby", "0233232232");
INSERT INTO Seller VALUES (2, "Ben", "0311111111");

INSERT INTO Buyer VALUES (1, "Maggie", "0333333333");
INSERT INTO Buyer VALUES (2, "Nicole", "0444444444");

INSERT INTO Artefact VALUES (1, "Vase", "Old Vase");
INSERT INTO Artefact VALUES (2, "Knife", "Old Knife");

INSERT INTO Offer VALUES (1, 1, 1, "2012-06-20", 81223.23, DEFAULT);
INSERT INTO Offer VALUES (1, 1, 2, "2012-06-20", 82223.23, DEFAULT);
INSERT INTO Offer VALUES (2, 2, 1, "2012-06-20", 19.95, DEFAULT);
INSERT INTO Offer VALUES (2, 2, 2, "2012-06-20", 23.00, DEFAULT);
```

- list all Offers. Show Artefact, Seller, Buyer and Offer details
- this is a FOUR table join

```
SELECT * FROM Artefact
       INNER JOIN Offer ON Artefact.ID = Offer.ArtefactID
       INNER JOIN Seller ON Seller.SellerID = Offer.SellerID
       INNER JOIN Buyer ON Buyer.BuyerID = Offer.BuyerID;
```

Ternary Query Output

ID	Name	Description	SellerID	ArtefactID	BuyerID	Date	Amount	Accep	SellerID	Name	Phone	BuyerID	Name	Phone
1	Vase	Old Vase	1	1	1	2012-06-20	81223.23	N	1	Abby	0233232232	1	Maggie	0333333333
1	Vase	Old Vase	1	1	2	2012-06-20	82223.23	N	1	Abby	0233232232	2	Nicole	0444444444
2	Knife	Old Knife	2	2	1	2012-06-20	19.95	N	2	Ben	0311111111	1	Maggie	0333333333
2	Knife	Old Knife	2	2	2	2012-06-20	23.00	N	2	Ben	0311111111	2	Nicole	0444444444

- Note the value of Accepted
  - "N" – the default value from our create statement
- Note that some columns have ambiguous names
  - SellerID
  - BuyerID
  - Name
  - Phone

Better output by using aliases

```
SELECT A.ID, A.Name AS Artefact, A.Description AS ArtDesc, Date AS OfferDate,
       Amount AS OfferAmount, Acceptance AS OfferAccepted, S.SellerID,
       S.Name AS Seller, S.Phone AS SellerPhone, B.BuyerID, B.Name AS Buyer,
       B.Phone AS BuyerPhone
       FROM Artefact A
            INNER JOIN Offer O ON A.ID = O.ArtefactID
            INNER JOIN Seller S ON S.SellerID = O.SellerID
            INNER JOIN Buyer B ON B.BuyerID = O.BuyerID;
```

ID	Artefact	ArtDesc	OfferDate	OfferAmount	OfferAccepted	SellerID	Seller	SellerPhone	BuyerID	Buyer	BuyerPhone
1	Vase	Old Vase	2012-06-20	81223.23	N	1	Abby	0233232232	1	Maggie	0333333333
1	Vase	Old Vase	2012-06-20	82223.23	N	1	Abby	0233232232	2	Nicole	0444444444
2	Knife	Old Knife	2012-06-20	19.95	N	2	Ben	0311111111	1	Maggie	0333333333
2	Knife	Old Knife	2012-06-20	23.00	N	2	Ben	0311111111	2	Nicole	0444444444

- aliases for table names: “A” “O” “S” “B”
- aliases for column names: Artefact, ArtDesc etc

SQL Wrapup

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?)

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

messy

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;
```

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

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	Sean	Sean's Address	2012-02-02	NULL	S	NULL	NULL	92000.00	N	NULL
2	Linda	Linda's Address	2011-06-12	NULL	S	NULL	NULL	92300.00	Y	NULL
3	Alice	Alice's Address	2012-12-02	NULL	H	NULL	NULL	NULL	NULL	23.43
4	Alan	Alan's Address	2010-01-22	NULL	H	NULL	NULL	NULL	NULL	29.43
5	Peter	Peter's Address	2010-09-07	NULL	C	19223	210.00	NULL	NULL	NULL
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

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%.”

```sql
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;
```

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

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

Result 11

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

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

```
6 • SELECT * FROM DepartmentSales;
```

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

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

```
6 • SELECT * FROM DepartmentSales
7   WHERE numSales > 5;
```

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

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

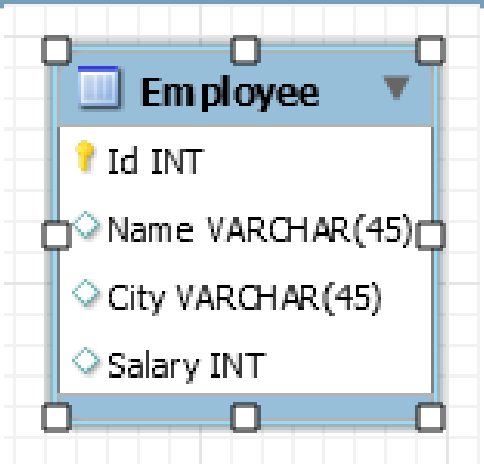
When can we Update or Insert a view?

- 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;
```

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

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

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