SQL Cont.

Divide Operation

- Find bars each of which sell all beers Justin likes.
- Relational Algebra: πbar,beerSells ÷ (πbeer(σdrinker='Justin' Likes))

Bar	Beer	Price
Australia Hotel	Burragorang Bock	3.5
Coogee Bay Hotel	New	2.25
Coogee Bay Hotel	Old	2.5
Coogee Bay Hotel	Sparkling Ale	2.8
Coogee Bay Hotel	Victoria Bitter	2.3
Lord Nelson	Three Sheets	3.75
Lord Nelson	Old Admiral	3.75
Marble Bar	New	2.8
Marble Bar	Old	2.8
Marble Bar	Victoria Bitter	2.8
Regent Hotel	New	2.2
Regent Hotel	Victoria Bitter	2.2
Royal Hotel	New	2.3
Royal Hotel	Old	2.3
Royal Hotel	Victoria Bitter	2.3

Drinker	Beer
Adam	Crown Lager
Adam	Fosters Lager
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

Divide Operation

- Find bars each of which sell all beers Justin likes.
  - πbar,beerSells ÷ (πbeer(σdrinker='Justin' Likes))
- SQL below

```
Select distinct a.bar
from sells a
where not exists
    ( (select b.beer from likes b
       where b.drinker = 'Justin')
       except
      (select c.beer from sells c
       where c.bar = a.bar )
      );
```

Find the brewers whose beers John likes.

```
SELECT Manf
FROM Likes, Beers
WHERE drinker = 'John' AND beer = name;
```

Likes:

Drinker	Beer
Adam	Crown Lager
Adam	Fosters Lager
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey's
Old	Toohey's
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey's
Sheaf Stout	Toohey's
Sparkling Ale	Cooper's
Stout	Cooper's
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

Recall: Querying Multi-relations

- Example: Find the brewers whose beers John likes.
  - Likes(drinker, beer)
  - Beers(name, manf)

```
SELECT Manf
FROM Likes, Beers
WHERE drinker = 'John' AND beer = name;
```

```
MANF
--------------------
Caledonian
Sierra Nevada
Sierra Nevada
Lord Nelson
```

- Note: could eliminate the duplicates by using ***DISTINCT***.
- Relational algebra: $\pi_{manf}(\sigma_{drinker='John'} Likes \bowtie_{(beer, name)} Beers)$.

Recall: Querying Multi-relations

- Syntax:

  SELECT *Attributes*

  FROM *R1, R2, ...*

  WHERE *Condition*

- FROM clause contains a list of relations.

Querying Multi-relations(cont.)

- For SQL *SELECT* statement on several relations:

  SELECT *Attributes*

  FROM *R1, R2, ...*

  WHERE *Condition*

- **Formal semantics (relational algebra):**
  - start with product $R1 \times R2 \times \ldots$ in FROM clause
  - apply σ using Condition in WHERE clause
  - apply π using Attributes in SELECT clause

Name Clashes - Attributes

- Two attributes from diff relations may have the same name.
  - Same names can cause confusion in the WHERE clause.
- Example: Which hotels have the same name as a beer?

```
SELECT Bars.name
FROM Bars, Beers
WHERE name = name;
```

Beers (name, manf)
Bars (name, addr, license)

- Solution: Disambiguate attributes with their relation name.

```
SELECT Bars.name
FROM Bars, Beers
WHERE Bars. name = Beers.name;
```

Name Clashes - Attributes

- You can qualified names even if there is no ambiguity:

```
SELECT Sells.beer
FROM Sells
WHERE Sells.price > 3.00;
```

- Advice:
  - qualify attribute names only when absolutely necessary.
  - SQL's **AS** operator cannot be used to resolve name clashes.

Name Clashes - Table

- The relation-dot-attribute convention doesn't help if we use the same relation twice in SELECT.
- To handle this, we need to define new names for each "instance" of the relation in the FROM clause.
- Example: Find pairs of beers by the same manufacturer.
- Note: we should avoid:
  - pairing a beer with itself e.g. (New,New)
  - same pairs with different order e.g. (New,Old) (Old,New)

```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND
b1.name < b2.name;
```

NAME	NAME
Crown Lager	Fosters Lager
Crown Lager	Invalid Stout
Fosters Lager	Invalid Stout
Fosters Lager	Melbourne Bitter
….	

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey's
Old	Toohey's
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey's
Sheaf Stout	Toohey's
Sparkling Ale	Cooper's
Stout	Cooper's
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

Recall: Renaming via as

- The renaming operator ***ρ*** was defined in relational algebra to avoid ***name clashes***.
  - Example: $\rho_{Beers(Brand,Brewer)}(Beers)$

- Changes the name of the relation and or the attribute names. The tuples are not changed.

Renaming via AS (cont.)

- SQL provides *AS* to achieve this; used in the SELECT part.
- Example: Changing name of relation *Beers(name, manf)*
  SELECT name AS Brand, manf AS Brewer FROM Beers;

BRAND	BREWER
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
…	

Expressions as Values in Columns

- ***AS*** can introduce an attribute of computed values
- Example: *Sells(bar, beer, price)*

  SELECT bar, beer, price*120 AS PriceInYen FROM Sells;

BAR	BEER	PRICEINYEN
Australia Hotel	Burragorang Bock	420
Coogee Bay Hotel	New	270
Coogee Bay Hotel	Old	300
Coogee Bay Hotel	Sparkling Ale	336
Coogee Bay Hotel	Victoria Bitter	276
…		

Inserting Text in Result Table

As can be used to intro. an attribute that is a **constant expression**.

Example: Likes(drinker, beer)

```
SELECT drinker, 'likes Cooper''s' AS WhoLikes
FROM Likes
WHERE beer = 'Sparkling Ale';
```

DRINKER	WHOLIKES
Gernot	likes Cooper's
Justin	likes Cooper's

Drinker	Beer
Adam	Crown Lager
Adam	Fosters Lager
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

Grouping

- SELECT-FROM-WHERE can be followed by GROUP BY to:
  - partition result relation into groups (according to values of specified attribute)
  - treat each group separately in computing aggregations
- Example: How many beers does each brewer make?

```
SELECT manf, COUNT(beer)
FROM Beers
GROUP BY manf;
```

MANF	COUNT(beer)
Caledonian	1
Carlton	5
Cascade	1
Cooper's	2
George IV Inn	1
Lord Nelson	2
Sierra Nevada	2
Toohey's	4

Grouping(cont.)

- GROUP BY is used as follows:

  SELECT attributes/aggregations

  FROM relations

  WHERE condition

  GROUP BY attribute

- Semantics:
  - partitions result into groups based on distinct values of attribute
  - apply any aggregation separately to each group

Grouping(cont.)

- Grouping is typically used in queries involving the phrase “for each”.
- Example: For each drinker, find the average price of New at the bars they frequently go to.

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'New' AND Frequents.bar = Sells.bar
GROUP BY drinker;
```

DRINKER	AVG(PRICE)
Adam	2.25
John	2.25
Justin	2.5

Grouping(cont.)

- When using grouping, every attribute in the SELECT list must:
  - have an aggregation operator applied to it OR
  - appear in a GROUP-BY clause
- Incorrect Example: Find the cheapest beer price in each bar.

```
SELECT bar, MIN(price)
FROM Sells;
ERROR:  column "sells.bar" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: select bar, min(price) from sells;
```

Grouping(cont.)

- How to answer the above query?

```
SELECT bar, MIN(price)
FROM Sells
GROUP BY BAR
```

bar	MIN(PRICE)
Australia Hotel	3.5
Coogee Bay Hotel	2.25
Lord Nelson	3.75
Marble Bar	2.8
Regent Hotel	2.2
Royal Hotel	2.3

Eliminating Groups

- In some queries, you can use the WHERE condition to eliminate groups.

- Example: Average beer price by suburb excluding hotels in The Rocks.

```
SELECT Bars.addr, AVG(Sells.price)
FROM Sells, Bars
WHERE Bars.addr != 'The Rocks' AND Sells.bar = Bars.name
GROUP BY Bars.addr;
```

ADDR	AVG(SELLS.PRICE)
Coogee	2.4625
Kingsford	2.2
Randwick	2.3
Sydney	2.8

Eliminating Groups

- You can impose complex conditions on groups, use the HAVING clause. HAVING is used to qualify a GROUP-BY clause:

```
SELECT attributes/aggregations
FROM relations
WHERE condition
GROUP BY attribute
HAVING condition;
```

- Semantics of HAVING:
  - generate the groups as for GROUP-BY
  - eliminate any group not satisfying HAVING condition
  - apply an aggregation to remaining groups

Eliminating Groups(cont.)

- Example: Find the average price of popular beers (i.e. those that are served in more than one hotel).

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) > 1;
```

BEER	AVG(PRICE)
New	2.3875
Old	2.53333333
Victoria Bitter	2.4

Subqueries

- The result of a SELECT-FROM-WHERE query can be used in the WHERE clause of another query.

- **Simplest Case**: Subquery returns one tuple.
  - Can treat the result as a constant value and use =.

Example: Find bars that sell New at the price same as the Coogee Bay Hotel charges for VB.

Sells:

Bar	Beer	Price
Australia Hotel	Burragorang Bock	3.5
Coogee Bay Hotel	New	2.25
Coogee Bay Hotel	Old	2.5
Coogee Bay Hotel	Sparkling Ale	2.8
Coogee Bay Hotel	Victoria Bitter	2.3
Lord Nelson	Three Sheets	3.75
Lord Nelson	Old Admiral	3.75
Marble Bar	New	2.8
Marble Bar	Old	2.8
Marble Bar	Victoria Bitter	2.8
Regent Hotel	New	2.2
Regent Hotel	Victoria Bitter	2.2
Royal Hotel	New	2.3
Royal Hotel	Old	2.3
Royal Hotel	Victoria Bitter	2.3

Subqueries (cont.)

- Example: Find bars that sell New at the price same as the Coogee Bay Hotel charges for VB.

```
SELECT bar
FROM Sells
WHERE beer = 'New'
        AND price =
                (SELECT price
                 FROM Sells
                 WHERE bar = 'Coogee Bay Hotel'
                 AND beer = 'Victoria Bitter' );
```

```
BAR
----------
Royal Hotel
```

Without using subqueries

- Example: Find bars that sell New at the price same as the Coogee Bay Hotel charges for VB.

```
SELECT b2.bar
FROM Sells b1, Sells b2
WHERE b1.beer = 'Victoria Bitter' and b1.bar = 'Coogee Bay
Hotel' and b1.price = b2.price and b2.beer = 'New';
```

```
BAR
----------
Royal Hotel
```

Subqueries(cont.)

- Subquery returns multiple tuples/a relation.
  - Treat it as a list of values, and use the various operators on lists/sets (e.g. IN).

- IN Operator
  - Tests whether one specified tuple is contained in a relation.
  - Tuple IN relation: is true iff the tuple is contained in the relation.
  - Tuple NOT IN relation : is true iff the tuple is not contained in the relation.

Example: Find the name and brewers of beers that John likes.

Likes:

Drinker	Beer
Adam	Crown Lager
Adam	Fosters Lager
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey's
Old	Toohey's
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey's
Sheaf Stout	Toohey's
Sparkling Ale	Cooper's
Stout	Cooper's
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

Subqueries (cont.)

- Example: Find the name and brewers of beers that John likes.

```
SELECT *
FROM Beers
WHERE name IN
        (SELECT beer
         FROM Likes
         WHERE drinker = 'John'
         );
```

NAME	MANF
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Pale Ale	Sierra Nevada
Three Sheets	Lord Nelson

- The subquery answers the question "What are the names of the beers that John likes?"
- Note that this query can be answered equally well without using IN.
- The subquery version is potentially (but not always) less efficient.

Subqueries (cont.)

- Example: Find the name and brewers of beers that John likes.

```
SELECT *
FROM Beers
WHERE name IN
            (SELECT beer
             FROM Likes
             WHERE drinker = 'John'
             );
```

```
SELECT Beers.*
FROM Beers, Likes
Where Beers.name = Likes.beer
and Likes.drinker = 'John';
```

NAME	MANF
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Pale Ale	Sierra Nevada
Three Sheets	Lord Nelson

Example: Find the beers uniquely made by their manufacturer.

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey's
Old	Toohey's
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey's
Sheaf Stout	Toohey's
Sparkling Ale	Cooper's
Stout	Cooper's
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

EXISTS Function

- Usage: EXISTS (relation), returns true iff relation is non-empty.
- Example: Find the beers uniquely made by their manufacturer.

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS
                (SELECT *
                 FROM Beers
                 WHERE manf = b1.manf
                 AND name != b1.name
                 );
```

```
NAME
--------------------
80/-
Burragorang Bock
Premium Lager
```

Quantifiers

- ANY and ALL behave as existential and universal quantifiers respectively.
- **Example:** Find the beers sold for the highest price.

```
SELECT beer
FROM Sells
WHERE price >=
        ALL(
            SELECT price
            FROM sells
             );
```

```
BEER
--------------------
Three Sheets
Old Admiral
```

Aggregation

- Selection clauses can contain aggregation operations.
- Example: What is the average price of New?

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'New';
```

AVG(PRICE) ← Same as AVG (DISTINCT price) ?

2.3875

- All prices for 'New' will be included, even if two hotels sell it at the same price.
- If set semantics used, the result would be wrong.

Aggregation(cont.)

- If we want set semantics, we can force using DISTINCT.
- Example: How many different bars sell beer?

```
SELECT COUNT(DISTINCT bar) FROM Sells;

COUNT(DISTINCT BAR)
-------------------
6
```

- Without DISTINCT, the result is 15 ... the number of entries in the Sells table.

Aggregation(cont.)

- The following operators apply to a list of numeric values in one column of a relation:
  - SUM AVG MIN MAX COUNT

- The notation COUNT(*) gives the number of tuples in a relation. Example: *How many different beers are there?*

```
SELECT COUNT(*) FROM Beers;

COUNT(*)
---------
18
```

Appendix: More on Changing Tables

- Accomplished via the ALTER TABLE operation:

  ALTER TABLE Relation Modifications
- Some possible modifications are:
  - add a new column (attribute),
  - change the properties of an existing attribute,
  - remove an attribute
- What happens to the data?

Appendix: More on Changing Tables

- Example: Add phone numbers for hotels.

```
ALTER TABLE Bars
ADD phone char(10) DEFAULT 'Unlisted';
```

- This appends a new column to the table and sets value for this attribute to 'Unlisted' in every tuple.
- Specific phone numbers can subsequently be added via:

```
UPDATE Bars
SET phone = '9665-0000'
WHERE name = 'Coogee Bay Hotel';
```

- If no default values is given, new column is set to all NULL.

Appendix: More on Changing Tables

- Can make multiple changes to one relation with a single ALTER.
- Example: Add opening and closing times to Bars

```
ALTER TABLE Bars
Add opens NUMERIC(4,2) DEFAULT 10.00 ,
Add closes NUMERIC(4,2) DEFAULT 23.00 ,
Add manager VARCHAR(20)
;
```

- Note that manager will be initially NULL for all hotels.