SQL Queries

- Dr Claire Ellul
- c.ellul@ucl.ac.uk

Assignment Progress ..

- By now you should have written your system specification and created the corresponding conceptual and logical ER Diagrams
- You should also have started drafting the DDL and DML for your entities (including constraints, but without the spatial components)

Spatial Data Management

- Overview
  - SQL - Query Language
    - Simple Queries
    - Aggregates, Group By, Order By and Distinct
    - Querying Multiple Tables
  - Moving from Logical to Physical

The Select Query

- General Select Query

```
SELECT <FIELD LIST>
FROM <TABLENAME>
WHERE <CONDITION>;
```

- -- List all the cleaners who work at UCL

```
select * from assets.cleaner;
```

In SQL, the * means 'everything' so select * will get you all the rows and columns in a table

The Select Query

- Selecting individual fields

```
SELECT <FIELD1, FIELD2, FIELD3>
FROM <TABLENAME>
WHERE <CONDITION>;
```

```
SELECT room_number, room_use from assets.rooms;
```

The Select Query

- The WHERE clause
  - Allows you to only select some of the records
  - Can use mathematical expressions such as =, <, >
  - Can also use string comparisons such as LIKE
  - Case sensitive
  - Kind of like a filter in Excel…

The Select Query

- The WHERE clause - string comparison
  - Find all the cleaners whose surname begins with B

```
select * from assets.cleaner where
cleaner_surname like 'B%';
```

The Select Query

- The WHERE clause
  - Test some variations of the above query

```
select * from assets.cleaner where cleaner_surname like 'b%';

select * from assets.cleaner where cleaner_surname like '%e%';
```

  - Which windows were installed after 1969?

```
select * from assets.windows where window_installation_date > '1-Jan-1970'
```

The Select Query

- The WHERE clause – combinations
  - -- Find classrooms that have been repainted in 2000 or later and where the room number starts with 1
  - Build the query up bit by bit

```
SELECT * FROM assets.rooms where room_number like '1%'

SELECT * FROM assets.rooms
where room_number like '1%'
And last_repainted > '1-Jan-2000'
and room_use = 'classroom';
```


Quotation Marks

- Note that you use quotation marks to surround any STRING or DATE values in the WHERE clause
- As before, these should be straight quotes
  - So don't type your SQL into Microsoft Word as it doesn't use the correct quotes

The Select Query

- Finding the number of records (rows)

```
SELECT COUNT(*)
FROM <TABLENAME>;

-- Find out the number of buildings at UCL

select count(*) from assets.buildings;
```

The Select Query

- You can also use an *alias* to rename a column
  - Find out the number of cleaners who work for UCL

```
select count(*) from assets.cleaner;

select count(*) as num_cleaners from
assets.cleaner;
```

UCL

Spatial Data Management

- Overview
  - **SQL - Query Language**
    - Simple Queries
    - **Aggregates, Group By, Order By and Distinct**
    - Querying Multiple Tables
  - Moving from Logical to Physical

UCL

Aggregates, Group By, Order By and Distinct

- Maximum, Minimum and Sum values
  - Queries can be performed on numerical fields

```
SELECT MAX(<FIELDNAME>)
FROM <TABLENAME>
WHERE <CONDITION>

SELECT MIN(<FIELDNAME>)
FROM <TABLENAME>
WHERE <CONDITION>

SELECT SUM(<FIELDNAME>)
FROM <TABLENAME>
WHERE <CONDITION>
```

UCL

Aggregates, Group By, Order By and Distinct

- Maximum, Minimum, Sum and Average values
  - Find the window(s) that were installed earliest

```
select min(window_installation_date) as oldest_window
from assets.windows;

-- Find the highest temperature value

select max(value_degrees_c)
from assets.temperature_values;
```

UCL

Aggregates, Group By, Order By and Distinct

- Maximum, Minimum, Sum and Average values
  - What is the average temperature recorded at UCL

```
select avg(value_degrees_c) from
assets.temperature_values;

-- Find the total area of rooms on campus - we use an
st_area query for this (see next week)

select sum(st_area(location))
from assets.rooms;
```


UCL

Aggregates, Group By, Order By and Distinct

- Group by queries
  - Allows you to divide the table into subsets before you do sum, max, min operations
  - Avoids repeating queries over and over for each different subset

```
SELECT MIN(<FIELDNAME>)
FROM <TABLENAME>
GROUP BY <FIELDNAME>
```

UCL

Aggregates, Group By, Order By and Distinct

-- Find the number of classrooms of each type

Without GROUP BY you need to run multiple queries and also know the different room uses

Aggregates, Group By, Order By and Distinct

```
select count(*) as num_rooms
from assets.rooms where room_use = 'classroom';
select count(*) as num_rooms
from assets.rooms where room_use = 'kitchen';
select count(*) as num_rooms
from assets.rooms where room_use = 'other';
select count(*) as num_rooms
from assets.rooms where room_use = 'engineering lab';
select count(*) as num_rooms
from assets.rooms where room_use = 'computer lab';
```

Aggregates, Group By, Order By and Distinct

- Find the number of classrooms of each type
  - Group By gives you a list for all room_use values

```
select count(*) as num_rooms, room_use
from assets.rooms
group by room_use;
```

Aggregates, Group By, Order By and Distinct

- Group by queries
  - Do any of the rooms in different buildings have the same room number:

```
select count(*) as num_rooms, room_number
from assets.rooms
group by room_number;
```

Aggregates, Group By, Order By and Distinct

- Group By
  - What is the average temperature for each sensor

```
select avg(temperature_degrees_c),
temperature_sensor_id from
assets.room_temperature_values
group by temperature_sensor_id;
```


Aggregates, Group By, Order By and Distinct

- Order By
  - Allows you to sort items in numerical or alphabetical order

```
SELECT <FIELDNAME1>, <FIELDNAME2>,<FIELDNAME3>
FROM <TABLENAME>
WHERE <CONDITION>
ORDER BY <FIELDNAME2>, <FIELDNAME1>
```

Aggregates, Group By, Order By and Distinct

- List the cleaners in alphabetical order

```
select * from assets.cleaner order by
cleaner_surname, cleaner_name;
```

Aggregates, Group By, Order By and Distinct

- List all the cleaners in reverse order

```
select * from assets.cleaner order by
(cleaner_surname, cleaner_name) desc;
```

- What happens when you remove the brackets?

```
select * from assets.cleaner order by
cleaner_surname, cleaner_name desc;
```

Aggregates, Group By, Order By and Distinct

- DISTINCT
  - You can use this in a query to avoid duplicate results
  - It will return the first instance of each row it finds and then ignore any subsequent rows that are the same

Aggregates, Group By, Order By and Distinct

- Find the times when at least one temperature sensor is recording data

```
SELECT date_and_time
from assets.temperature_values;
```

vs.

```
SELECT DISTINCT date_and_time
from assets.temperature_values;
```

DISTINCT ON

- This keeps the data only for the first time a value occurs, and is usually used with ORDER BY
  - Find the first time a temperature value was measured

```
SELECT DISTINCT ON(value_degrees_c)
value_degrees_c, date_and_time
    from assets.temperature_values
    order by value_degrees_c, date_and_time;
```


Spatial Data Management

- Overview
  - **SQL - Query Language**
    - Simple Queries
    - Aggregates, Group By, Order By and Distinct
    - **Querying Multiple Tables**
  - Moving from Logical to Physical

Querying Multiple Tables

- Queries on multiple tables allow the user to ask more complex questions of the database
- We will focus on three types
  - Sub queries/nested queries
  - Set queries
  - Join queries

Querying Multiple Tables

- Reminder – for your assignment you are required to have a minimum of TWO functional requirements that query multiple tables!
  - To make your life easier, the other 6 functional requirements can be queries on one table only, but do make them realistic
    - select the number of buildings in the university is probably too simplistic (select count(*) from) …
    - calculate the area of external walls that require cleaning would be a better option

Sub Queries (Nested Queries)

- A sub query is a query within a query
- The output of any query can be used like a temporary/virtual table as input to another query
- So instead of:

  SELECT * FROM << TABLE NAME >>
- You can have (note the brackets)

  SELECT * FROM (<< SELECT * FROM << TABLE NAME >>)

Sub Queries (Nested Queries)

```
SELECT <FIELDNAME1, FIELDNAME2>
FROM <TABLENAME1>
WHERE <FIELDNAME1> IN
(SELECT <FIELDNAME2> FROM <TABLENAME2>)
```

Note: the two tables – TABLENAME1 and TABLENAME2 can be different (most of the time) but very occasionally the tables can also be the same table

Sub Queries (Nested Queries)

- Find the windows in the Pearson Building
  - First write a sub query to find the ID of the Pearson Building

```
select building_id from assets.buildings
where building_name = 'Pearson';
```

Sub Queries (Nested Queries)

- Find the windows in the Pearson Building
  - Then nest this inside a query to find the windows (use brackets)

```
select * from assets.windows where building_id =
(select building_id from assets.buildings where
building_name = 'Pearson');
```

Sub Queries (Nested Queries)

- Find the newest window
  - First find the maximum (latest) installation date

```
select max(window_installation_date) from
assets.windows;
```

Querying Multiple Tables

- Additional Examples
  - Which building is the newest window installed in?

```
select * from assets.buildings where building_id
= (select building_id from assets.windows where
window_installation_date = (select
max(window_installation_date) from
assets.windows));
```

Querying Multiple Tables

- Additional Examples
  - When did the maximum temperature reading occur?

```
select date_and_time from
assets.temperature_values
where value_degrees_c = (select
max(value_degrees_c) from
assets.temperature_values);
```

Querying Multiple Tables

- Additional Examples
  - What room was the minimum temperature recorded in? (nested queries)

```
select room_number from assets.rooms
where room_id = (
select room_id from assets.temperature_sensor
where sensor_id = (
select temperature_sensor_id from assets.temperature_values
where value_degrees_c = (select max(value_degrees_c) from
assets.temperature_values)));
```


Sub Queries (Nested Queries)

- Find the newest window
  - Then nest it to get the rest of the details about the window (you can sub query on the same table)

```
select * from assets.windows where
window_installation_date = (select
max(window_installation_date) from
assets.windows);
```

Sub Queries – the WITH statement

```
with maxwin as (select
max(window_installation_date) as maxw
from assets.windows)
select * from assets.windows where
window_installation_date = (select maxw
from maxwin);
```

Querying Multiple Tables

- Set Queries
  - Cartesian Product
  - Union
  - Intersection
  - Difference
- Join Queries
  - Inner
  - Left Outer
  - Full Outer

Querying Multiple Tables

- Set Queries
  - Equivalent of standard set operations
  - Result in the creation of lists of elements (the result set of the query)
  - Allow the user to identify commonalities or differences between attributes in the same table or in different tables
  - Union, intersect and difference operators can only return one column of data.

Querying Multiple Tables

- Set Queries - Cartesian Product (also known as a cross join)
  - A = {1,2,3}
  - B = {A,B}

```
SELECT *
FROM A,B;
```

RESULT = {(1,A),(1,B),(2,A),(2,B),(3,A),(3,B)}

Querying Multiple Tables

- Set Queries - Cartesian Product

```
select a.*,b.*
from assets.buildings a, assets.windows b;
```

- Note the use of the ALIASes a and b to substitute for the table names to define the columns to be selected
- * in this case means 'all columns'

Querying Multiple Tables

- Set Queries - Union
  - Union of two sets
    - The set of elements that belong to set A, set B or both sets
  - Union of two tables
    - The set of tuples that belong to table A, table B or both tables

Querying Multiple Tables

- Set Queries - Union
  - Develop an index of UCL showing all the location names (this could be used for a searchable map of the campus)

```
select university_name from assets.university
union all
select building_name from assets.buildings
union all
select room_number from assets.rooms;
```

Querying Multiple Tables

- Set Queries - Difference
  - Difference of two sets A and B
    - The set of elements that belongs to A but not to B
  - Difference of two tables A and B
    - The set of values that belongs to table A but not to table B

Querying Multiple Tables

- Difference – find out which rooms don't have temperature sensors

```
select room_id from assets.rooms
except
select room_id from assets.temperature_sensor;
```

Querying Multiple Tables

- Difference – find out which rooms don't have temperature sensors
  - Nest the query to get more details about the rooms

```
select * from assets.rooms
where room_id in
(select room_id from assets.rooms
except
select room_id from assets.temperature_sensor);
```

Querying Multiple Tables

- Set Queries - Intersection
  - Intersection of two sets A and B
    - The set of elements that belongs to both set A and B
  - Intersection of two tables A and B
    - The set of tuples that belongs to both table A and table B

Querying Multiple Tables

- Difference – find out which rooms DO have temperature sensors
  - Nest the query to get more details about the rooms

```
select * from assets.rooms
where room_id in
(select room_id from assets.rooms
intersect
select room_id from assets.temperature_sensor);
```

Queries

- Worksheet – SQL Queries

Querying Multiple Tables

- Join Queries
  - An alternative way to use data from multiple tables to answer a question
  - Sometimes more efficient than a nested or set query
    - If time allows, we will cover query efficiency as an 'advanced topic'

Querying Multiple Tables

- Join Queries
  - Most powerful query in SQL
  - Two types
    - Inner
    - Outer
  - Rely on PRIMARY KEY/FOREIGN KEY relations being set up
    - (Or can also do spatial joins –see later in the course)
  - Must specify the join criteria

Querying Multiple Tables

- Join Queries - Inner Joins
  - Returns only the elements from BOTH tables where the given condition is met
  - Generate a list of all the rooms and their corresponding buildings

```
select a.*, b.*
from assets.rooms a INNER JOIN assets.buildings b
on a.building_id = b.building_id;
```

Querying Multiple Tables

- Join Queries - Inner Joins
  - Returns only the elements from BOTH tables where the given condition is met
  - Generate a list of all the rooms and windows - only rooms with windows will be listed

```
select a.*, b.*
from assets.rooms a INNER JOIN assets.windows b
on a.room_id = b.room_id;
```


Querying Multiple Tables

- Join Queries - Left Join
  - Returns the elements from table A (the first one in the list) where the condition is met.
  - Nulls used where common elements do not exist

Querying Multiple Tables

- Join Queries - Left Join
  - Generate a list of rooms and windows
  - Because the rooms are the LEFT side of the join (i.e. listed first), all rooms will be listed - and if they have windows these will be listed too
  - i.e. find which rooms don't have windows ..

```
select a.*, b.*
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id;
```

Querying Multiple Tables

- Join Queries – Left Join
  - All windows and rooms
  - As the WINDOWS are on the left this time, if there are any windows without rooms they will be listed, but any rooms without windows will not

```
select a.*, b.*
from assets.windows a LEFT JOIN assets.rooms b
on a.room_id = b.room_id;
```

Querying Multiple Tables

- Join Queries - Left Join
  - Generate a list of sensors and rooms
  - If the rooms are on the LEFT side of the join (i.e. listed first), all rooms will be listed – and if they have sensors these will be listed too
  - i.e. find which rooms do and don't have sensors..

```
select a.*, b.*
from assets.rooms a LEFT JOIN
assets.temperature_sensor b
on a.room_id = b.room_id;
```

Querying Multiple Tables

- Join Queries - Left Join
  - Generate a list of sensors and rooms
  - If the sensors are on the LEFT side of the join (i.e. listed first), all sensors will be listed – and if they have sensors these will be listed too
  - i.e. find which sensors are inside rooms and which are outside

```
select a.*, b.*
from assets.temperature_sensor a LEFT JOIN
assets.rooms b  on a.room_id = b.room_id;
```

Querying Multiple Tables

- Join Queries - Full Outer Join
  - Returns the elements from both left and right tables
  - Nulls used where common elements don't exist
  - Find out which sensors are in or out of rooms and which rooms do or don't have sensors in one query

```
select a.*, b.*
from assets.temperature_sensor a FULL JOIN assets.rooms b
on a.room_id = b.room_id;
```

Querying Multiple Tables

- Multiple Joins
  - Quite common that you need to JOIN data from more than two tables
  - Build these multi-joins up in the same way as you would a nested query
  - i.e. get a JOIN between two tables working and then that be comes a virtual 'table' that you can use in a JOIN with a third table

Querying Multiple Tables

- Multiple Joins
  - Are rooms with windows on average hotter than rooms without windows?
  - To answer this you need information about
    - Windows
    - Rooms
    - Temperature Sensors
    - Temperature values

Querying Multiple Tables

- Multiple Joins
  - Start by linking rooms and windows – we need both the rooms with windows and the rooms without windows so use an OUTER join
  - We only need 2 fields – don't select ALL fields as this gets difficult to manage

```
select a.room_id, b.window_id
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id;
```

Querying Multiple Tables

- Multiple Joins
  - We then need to find out the temperature of these rooms – NB we are only interested in rooms where there is a temperature sensor, so an INNER JOIN is sufficient

Querying Multiple Tables

- Multiple Joins

```
select c.sensor_id, d.room_id, d.window_id
from assets.temperature_sensor c INNER JOIN
(select a.room_id, b.window_id
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id) d
on c.room_id = d.room_id;
```

Querying Multiple Tables

- Multiple Joins
  - Now we need to find the temperature readings for these sensors ..
  - INNER JOIN is sufficient as we're only interested in sensors that have readings

Querying Multiple Tables

```
select f.*, e.sensor_id, e.room_id, e.window_id
from assets.temperature_values f INNER JOIN
(select c.sensor_id, d.room_id, d.window_id from
assets.temperature_sensor c INNER JOIN (select
a.room_id, b.window_id
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id) d
on c.room_id = d.room_id) e
on f.temperature_sensor_id = e.sensor_id;
```

Querying Multiple Tables

- Multiple Joins
  - Finally – what is the average reading for each room?
  - For this we need a GROUP BY query using both the room and the window IDs so that we can pick up any rooms without windows

Querying Multiple Tables

```
select g.room_id,g.window_id,avg(g.value_degrees_c) from
(select f.*, e.sensor_id, e.room_id, e.window_id
from assets.temperature_values f INNER JOIN
(select c.sensor_id, d.room_id, d.window_id from
assets.temperature_sensor c INNER JOIN (select a.room_id,
b.window_id
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id) d
on c.room_id = d.room_id) e
on f.temperature_sensor_id = e.sensor_id) g
group by g.room_id, g.window_id;
```

Querying Multiple Tables – the WITH statement

- The SQL in the above query is quite complicated
  - And will take time to write out
- You can use the WITH statement to write out a bit of the query and then reuse that SQL in the next bit
  - It is still one query, but easier to write out …

Querying Multiple Tables – the WITH statement

```
WITH rooms_windows as (select a.room_id,
b.window_id
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id)

select * from rooms_windows;
```

Note: only one ;

Querying Multiple Tables – the WITH statement

```
WITH rooms_windows as (select a.room_id, b.window_id
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id)

select c.sensor_id, d.room_id, d.window_id from
assets.temperature_sensor c INNER JOIN rooms_windows
d
on c.room_id = d.room_id;
```

Querying Multiple Tables – the WITH statement

- Use a second WITH statement around this new query

Querying Multiple Tables – the WITH statement

```
WITH temperature_rooms_windows as (WITH rooms_windows as
(select a.room_id, b.window_id
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id)
select c.sensor_id, d.room_id, d.window_id from
assets.temperature_sensor c INNER JOIN rooms_windows d on
c.room_id = d.room_id )

select f.*, e.sensor_id, e.room_id, e.window_id
from assets.temperature_values f INNER JOIN
temperature_rooms_windows e
on f.temperature_sensor_id = e.sensor_id;
```

Querying Multiple Tables – the WITH statement

- You can stack the WITH statements
  - Separate each one using a ,
  - Second query can reference the first one and so forth
  - Each part of the query – once you've worked it out – can become part of the WITH statement
  - Easier to read!

Querying Multiple Tables – the WITH statement

```
WITH
rooms_windows as (select a.room_id, b.window_id
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id),
temperature_rooms_windows as (select c.sensor_id, d.room_id, d.window_id
from assets.temperature_sensor c INNER JOIN rooms_windows d on c.room_id =
d.room_id)

select f.*, e.sensor_id, e.room_id, e.window_id
from assets.temperature_values f INNER JOIN
temperature_rooms_windows e
on f.temperature_sensor_id = e.sensor_id;
```

```
WITH
rooms_windows as (select a.room_id, b.window_id
from assets.rooms a LEFT JOIN assets.windows b
on a.room_id = b.room_id),
temperature_rooms_windows as (select c.sensor_id, d.room_id, d.window_id
from assets.temperature_sensor c INNER JOIN rooms_windows d on c.room_id =
d.room_id),
temperature_sensor_rooms as (select f.*, e.sensor_id, e.room_id, e.window_id
from assets.temperature_values f INNER JOIN
temperature_rooms_windows e
on f.temperature_sensor_id = e.sensor_id)

select g.room_id,g.window_id,avg(g.value_degrees_c) from
temperature_sensor_rooms g
group by g.room_id, g.window_id;
```

Spatial Data Management

- Overview
  - SQL - Query Language
    - Simple Queries
    - Aggregates, Group By, Order By and Distinct
    - Querying Multiple Tables
  - **Moving from Logical to Physical**

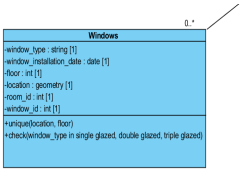
Database Design

- Database Design Tasks
  - Conceptual Design – a diagrammatic and text description of user requirements, documented as an Entity-Relationship diagram
  - Logical Design – takes the information gathered at Conceptual Design stage, and transforms it to take into account system performance and expected operational conditions.
  - Normalisation part of Logical Design, used to remove redundancies (duplicate data) from the model
  - Physical Design – takes the normalised logical design and converts it into actual build scripts for the database.

Logical to Physical

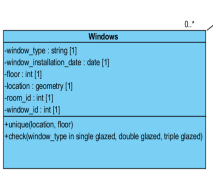
- Take the logical diagram and create the SQL for the tables, constraints and so forth
- Then insert the required data
- VALIDATE THE PROCESS AGAIN: Make sure that the resulting database can answer the questions that your system is supposed to answer

Logical to Physical – create the table



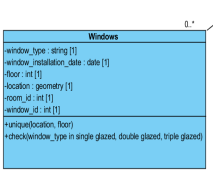

```
drop table if exists assets.windows;
create table assets.windows (
        window_id serial,
        building_id integer,
        window_type character varying(100),
        window_installation_date date,
        room_id integer,
        floor integer
);
```

Logical to Physical- create the primary key constraint




```
alter table assets.windows add constraint
window_pk primary key(window_id);
```

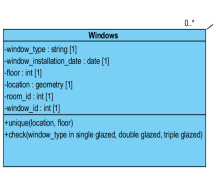
Logical to Physical- create the foreign key constraint



NB: A foreign key constraint references another table, so you need to make sure that table exists first!

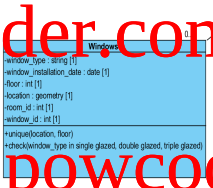
```
alter table assets.windows
add constraint windows_rooms_fk
foreign key(room_id) references
assets.rooms(room_id);
```

Logical to Physical- create any UNIQUE constraints



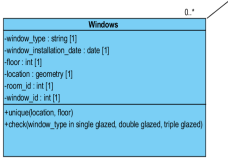

```
alter table assets.windows
add constraint windows_unique
unique(location, floor);
```

Logical to Physical - create any CHECK constraints




```
alter table assets.windows
add constraint window_type_check
check (window_type in ('single glazed','double
glazed','triple glazed'));
```


Logical to Physical- insert the data: some hints



We are using the SERIAL data type for ID values in our database

This automatically adds 1 to the last value EVERY TIME SOMEONE RUNS AN INSERT STATEMENT ON A TABLE

… EVEN IF THAT INSERT STATEMENT FAILS

So you can never guarantee that if you inserted 3 rows in a new table they will have IDs 1, 2 and 3

Logical to Physical- insert the data: some hints

- i.e. - you do not know in advance what ID values your data will have - these are completely arbitrary
- This is by design as usually multiple users will be inserting data at the same time so the SERIAL value just gives them the next value

> To avoid blocking concurrent transactions that obtain numbers from the same sequence, a nextval operation is never rolled back; that is, once a value has been fetched it is considered used, even if the transaction that did the nextval later aborts. This means that aborted transactions might leave unused "holes" in the sequence of assigned values.

https://stackoverflow.com/questions/35849944/postgresql-serial-incremented-on-failed-constraint-insert

- However - this causes problems as you don't know the ID value to use in the foreign key

UCL

Logical to Physical- insert the data: some hints

- Option 1 – use NULL and then use an UPDATE statement (green on next slide)
- Option 2 – use a nested query (red on next slide)

UCL

```
insert into assets.windows(building_id, window_type,
window_installation_date, room_id, floor, location)
values
((select building_id from assets.buildings where
building_name = 'Chadwick'), 'triple glazed','23-May-
2014',null,1,st_geomfromtext('POLYGON ((4 2 2,6 2 2, 6 2 4,
4 2 4, 4 2 2))',0));

update assets.windows a set room_id =
(select b.room_id from assets.rooms b
where st_3dintersects(a.location, b.location)
and a.floor = b.floor);
```

UCL

Spatial Data Management

- Overview
  - SQL - Query Language
    - Simple Queries
    - Aggregates, Group By, Order By and Distinct
    - Querying Multiple Tables
  - Moving from Logical to Physical