Semantic Constraints

The semantic constraints on the database to fulfil the needs of the database are basically the primary keys of the database. By definition, primary keys ensure originality. I will look at why the primary keys were chosen as primary keys and the impact it has on the database and its constraints:

- In the hotel table, the hotel_id is the only primary key as it is all we need to define a hotel, I could have used name, but it is very plausible that we could have similar hotel name, for instance Jurys Cork and Jurys Limerick could conflict quite easily so a NUMBER data type was a better choice here.
- In the group table, the same theory applies and its why we needed a group id, There is a possibility (if quite small) that groups could have the same name.
- Occupancy and Rate have the same primary keys, hotel_id and date. The importance of this is that hotel's are limited to only submit one entry per day per week. These were the obvious choice for primary key as they and they alone define a certain days occupancy and rate for a hotel.
- The table comp_set's primary key is comp_id as there must be a unique comp_id for every comp set created and hotel_id isn't a primary key here as hotel's can have a number of comp sets.
- The table comp_set_list has two primary key's, comp_id and hotel_id, this works because as a constraint I need that no comp_set can include 2 of the same hotels. By defining hotel_id as a primary key I ensure this originality.
- The table users has two primary keys, hotel_id and username, this is important as for any particular hotel, we can have a number of logins, at first this seems non logical, but for instance certain users can have certain access to testing features etc, which would be unavailable to the same hotel on a different login.

Database Security

The company is serious on security, and we have a rigid security policy. Due to laws with confidentiality it is vital that no hotel's are given access to individual hotel data of their opponents. This side of affairs is handled exclusively with the php frontend to the website and this is as a result that creating a new database user for access would be implausible as the website grew in size. Generally, we don't allow any user to write data to the database except for when they are submitting, and they then have read access to all the rest except obviously the PHP front-end gives them averages for a group of hotels rather than any individual hotels.

Onto the security implemented at the database level, as we have a number of employee's who need access to the database at different levels, we created different users who have different purposes and access levels:

the users are as follow:

hotel_access_read

- Access to only basic hotel data

hotel_access_update

- Access to update basic hotel information

submit_access

- Access to read and write submit data

all_read

- can read all tables

super_all

- Super User, has all privileges.

The commands used for this was as follows:

GRANT SELECT **ON** hotel **TO** hotel_access_read
GRANT SELECT,UPDATE **ON** hotel **TO** hotel_access_update
GRANT SELECT,INSERT, UPDATE **ON** rate, occupancy **TO** submit_access
GRANT SELECT **ON** * **TO** all_read

Trigger

The trigger I used was an auto increment trigger which creates a sequence which increments by 1 and has no limit, the code for this trigger is below:

```
create sequence hotel_seq
start with 1
increment by 1
nomaxvalue;

create trigger hotel_increment_trigger
before insert on hotel
for each row
begin
select hotel_seq.nextval into :new.hotel_id from dual;
end;
/
```

It is unneeded to reiterate here, but I created auto incrementing id's for group id and comp_id using the same method, its worth noting that the same can be got without a trigger by using an INSERT query such as the example one below:

INSERT INTO hotel (hotel_id, name, city, group_id, rooms, contact_name,

contact_email, contact_phone) VALUES (hotel_seq.nextval, 'Frankies Guesthouse Hotel Dublin', 'Dublin', '0', '323', 'frankie', 'blah@exampl.com ', ' ');

Views

```
CREATE VIEW week_over_view
as
SELECT
hotel.hotel_id, SUM(occupancy.last, SUM(rate.last), SUM((occupancy.last/hotel.rooms)
*rate.last)
FROM
hotel, rate, occupancy
WHERE
hotel.hotel_id = occupancy.hotel_id AND hotel.hotel_id = rate.hotel_id AND rate.hotel_id
= occupancy.hotel_id AND occupancy.submit_date = rate.submit_date;
```

This view shows the occupancy, rate and revpar for the week for each hotel.

```
CREATE VIEW max_occupancy
as
SELECT
hotel.name,occupancy.hotel_id, occupancy.last
FROM
occupancy, hotel
WHERE
occupancy.hotel_id = hotel.hotel_id AND
occupancy.last =
(SELECT
MAX(occupancy.last)
FROM
occupancy);
```

This view shows the best occupancy figures for the week, It would be easy to create a view that shows the best rate by simply replacing occupancy with rate.

Table Creation and Primary Keys

```
CREATE TABLE occupancy (
        hotel_id NUMBER(5),
        date DATE,
        last NUMBER(3),
        curr NUMBER(3),
        next NUMBER(3),
```

```
        CONSTRAINT occupancy_pk PRIMARY KEY(hotel_id, date)
);

CREATE TABLE rate (
        hotel_id NUMBER(5),
        date DATE,
        last NUMBER(3),
        curr NUMBER(3),
        next NUMBER(3),
        CONSTRAINT rate_pk PRIMARY KEY(hotel_id, date)
);

CREATE TABLE hotel (
        hotel_id NUMBER(5),
        name VARCHAR(40),
        city VARCHAR(30),
        group_id NUMBER(5),
        rooms NUMBER(4),
        contact_name VARCHAR(40),
        contact_email VARCHAR(40),
        contact_phone VARCHAR(40),
        CONSTRAINT hotel_pk PRIMARY KEY(hotel_id)
);

CREATE TABLE user (
        hotel_id NUMBER(5),
        username VARCHAR(10),
        password VARCHAR(10),
        CONSTRAINT user_pk PRIMARY KEY(hotel_id, username)
);

CREATE TABLE comp_set (
        comp_id NUMBER(5),
        hotel_id NUMBER(5),
        CONSTRAINT comp_set_pk PRIMARY KEY(comp_id)
);

CREATE TABLE comp_set_list (
        comp_id NUMBER(5),
        hotel_id NUMBER(5),
        CONSTRAINT comp_set_list_pk PRIMARY KEY(comp_id, hotel_id)
);

CREATE TABLE groups (
```

```
        group_id NUMBER(5),
        name VARCHAR(40),
        CONSTRAINT group_pk PRIMARY KEY(group_id)
);
```

Foreign Key Constraints

ALTER TABLE occupancy add CONSTRAINT occupancy_fk FOREIGN KEY(hotel_id) REFERENCES hotel(hotel_id);

ALTER TABLE rate add CONSTRAINT rate_fk FOREIGN KEY(hotel_id) REFERENCES hotel(hotel_id);

ALTER TABLE hotel add CONSTRAINT hotel_fk FOREIGN KEY(group_id) REFERENCES groups(group_id);

ALTER TABLE users add CONSTRAINT user_fk FOREIGN KEY(hotel_id) REFERENCES hotel(hotel_id);

ALTER TABLE comp_set add CONSTRAINT comp_set_fk FOREIGN KEY(hotel_id) REFERENCES hotel(hotel_id);

ALTER TABLE comp_set_list add CONSTRAINT comp_set_list_fk FOREIGN KEY(hotel_id) REFERENCES hotel(hotel_id);

ALTER TABLE comp_set_list add CONSTRAINT comp_set_list_fk_2 FOREIGN KEY(comp_id) REFERENCES comp_set(comp_id);

Insertion of Rows

To get a proper view how the company actually worked, it wasn't logical to insert a few rows into the table so I decided the best way to do this was to make .sql files to insert a good number of rows so that proper views could be done.

I did this using perl scripts for each of the tables, I will append these perl scripts at the end of this document, all it does is simply generate sql statements with random data.

First script generated the hotel data, the hotel names are genuine one's in Dublin area, The rest of the data is just random numbers for the likes of rooms.

Here is an example of some of the insert rows:

INSERT INTO hotel (hotel_id, name, city, group_id, rooms, contact_name, contact_email, contact_phone) VALUES ('92', 'Frankies Guesthouse Hotel Dublin', 'Dublin', '0', '323', 'frankie', 'blah@exampl.com ', ' ');
INSERT INTO hotel (hotel_id, name, city, group_id, rooms, contact_name, contact_email, contact_phone) VALUES ('93', 'Gate Lodge Guest house Hotel Dublin', 'Dublin', '0', '62', 'example', 'sample@example.com ', ' ');
INSERT INTO hotel (hotel_id, name, city, group_id, rooms, contact_name, contact_email, contact_phone) VALUES ('94', 'Georgian Hotel Dublin', 'Dublin', '0', '380', 'georgian', 'blah@example.com', '');
INSERT INTO hotel (hotel_id, name, city, group_id, rooms, contact_name, contact_email, contact_phone) VALUES ('95', 'Glen Guesthouse Hotel Dublin', 'Dublin', '0', '221', 'glen', 'sample@example.com', '');
INSERT INTO hotel (hotel_id, name, city, group_id, rooms, contact_name, contact_email, contact_phone) VALUES ('96', 'Glenogra Guesthouse Hotel Dublin', 'Dublin', '0', '317', 'glenogra', 'blah@example.com', '');

Occupancy & Rates

The script for occupancy and rates are similar, for occupancy I generate random occ figures for 7 days of the week, this is a slice of how the company runs, each day also gets a rate.

The script uses oracles to_date function to generate a date type from a formatted date string.

These are some samples from the insertions that were done with this script:

INSERT INTO occupancy (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/5', 'yyyy/mm/dd'), 58, 88, 74);
INSERT INTO occupancy (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/6', 'yyyy/mm/dd'), 59, 76, 22);
INSERT INTO occupancy (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/7', 'yyyy/mm/dd'), 107, 40, 72);
INSERT INTO occupancy (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/8', 'yyyy/mm/dd'), 70, 90, 22);
INSERT INTO occupancy (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/9', 'yyyy/mm/dd'), 41, 65, 57);
INSERT INTO occupancy (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/10', 'yyyy/mm/dd'), 107, 43, 76);
INSERT INTO occupancy (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/11', 'yyyy/mm/dd'), 71, 46, 63);
INSERT INTO rate (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/5', 'yyyy/mm/dd'), 192, 96, 68);
INSERT INTO rate (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/6', 'yyyy/mm/dd'), 133, 182, 141);

INSERT INTO rate (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/7', 'yyyy/mm/dd'), 139, 71, 79);
INSERT INTO rate (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/8', 'yyyy/mm/dd'), 189, 81, 188);
INSERT INTO rate (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/9', 'yyyy/mm/dd'), 180, 134, 129);
INSERT INTO rate (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/10', 'yyyy/mm/dd'), 93, 162, 119);
INSERT INTO rate (hotel_id, submit_date, last, curr, next) VALUES (1, to_date('2010/12/11', 'yyyy/mm/dd'), 76, 87, 108);

Compsets

Comp sets are done by first assigning a comp_id to each hotel_id, although this sounds like its not normalised, each hotel can have multiple comp_id's as the previous week's comp_id's are stored.

each comp_id then becomes one of the primary keys for comp_set_list which holds the hotel_ids of the hotel's in the compset.

To generate these compsets, I made another perl script to generate a comp_set for each of the hotels.

I first generate a comp_set entry for that hotel, (as this is only a slice of the company database, a week slice to be specific) and then generate the entrys in the comp_set_list for that comp_id with random id's for the compset hotels.

some sample querys:

INSERT INTO comp_set (comp_id, hotel_id) VALUES ('4', '4');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('4','18','2');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('4','19','22');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('4','20','72');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('4','21','33');
INSERT INTO comp_set (comp_id, hotel_id) VALUES ('5', '5');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('5','22','98');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('5','23','71');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('5','24','55');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('5','25','76');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('5','26','7');
INSERT INTO comp_set (comp_id, hotel_id) VALUES ('6', '6');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('6','27','76');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('6','28','26');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('6','29','86');

INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('6','30','70');
INSERT INTO comp_set_list (comp_id,comp_set_list_id,hotel_id) VALUES ('6','31','65');

In the real world situation, the amount of group's is quite small so I decided to enter these manually, as a matter of confidentiality, I've decided to just randomly group a few hotel's together to show how the relationship works

The insertions for group:

INSERT INTO groups (group_id, name) VALUES ('1', 'Group A');
INSERT INTO groups (group_id, name) VALUES ('2', 'Group B');
INSERT INTO groups (group_id, name) VALUES ('3', 'Group C');
INSERT INTO groups (group_id, name) VALUES ('4', 'Group D');
INSERT INTO groups (group_id, name) VALUES ('5', 'Group E');

As the hotels were all defaulted to group 0 (i.e no group) then I simply had to update certain hotels to be a part of each group.

As I've said above, I've chosen random hotels to group. The queries for the updates are below:

UPDATE hotel SET hotel.group_id = 1 WHERE hotel.hotel_id IN (12, 13, 14, 15);
UPDATE hotel SET hotel.group_id = 2 WHERE hotel.hotel_id IN (45, 67, 89, 23);
UPDATE hotel SET hotel.group_id = 3 WHERE hotel.hotel_id IN (35, 48, 27);
UPDATE hotel SET hotel.group_id = 4 WHERE hotel.hotel_id IN (25, 47, 22, 11);
UPDATE hotel SET hotel.group_id = 5 WHERE hotel.hotel_id IN (81, 79);
.