Published in final edited form as:
Comput Biol Chem. 2007 April ; 31(2): 143–147. doi:10.1016/j.compbiolchem.2007.01.004.

A Method of Microarray Data Storage Using Array Data Type

Lam C. Tsoi[a] and **W. Jim Zheng**[b,*]

[a]Bioinformatics Graduate Program, Hollings Cancer Center, Medical University of South Carolina, 135 Cannon Street, Suite 303, Charleston, SC 29425

[b]Department of Biostatistics, Bioinformatics and Epidemiology, and Bioinformatics Core Facility, Hollings Cancer Center, Medical University of South Carolina, 135 Cannon Street, Suite 303, Charleston, SC 29425

Abstract

A well-designed microarray database can provide valuable information on gene expression levels. However, designing an efficient microarray database with minimum space usage is not an easy task since designers need to integrate the microarray data with the information of genes, probe annotation, and the descriptions of each microarray experiment. Developing better methods to store microarray data can greatly improve the efficiency and usefulness of such data. A new schema is proposed to store microarray data by using array data type in an object-relational database management system – PostgreSQL. The implemented database can store all the microarray data from the same chip in an array data structure. The variable length array data type in PostgreSQL can store microarray data from same chip. The implementation of our schema can help to increase the data retrieval and space efficiency.

Keywords

Microarray; database schema; array data type; PostgreSQL

1. Introduction

The number of available public microarray databases is dramatically increasing as the cost of computer hardware decreases (Galperin 2006). However, the development of a good microarray database can only be achieved through a well-designed relational schema, which helps the system to manage the data effectively, and increases the data retrieval speed. In current microarray database (Ball et al. 2005; Cheung et al. 2002; Killion et al. 2003; Sherlock et al. 2001), it is common for a relational schema of a microarray database to consist of tables to store the accession number and description of gene (*GENE* table), the description of the chip (*CHIP* table), the probe name and the expression value (*DATA* table), and the experiment information (*EXPERIMENT* table). The schemas for most of the databases would tend to store the probe's expression value of each experiment in a record (a row) in the table *DATA*. For instance, in the Stanford (Ball et al. 2005; Sherlock et al. 2001) and Longhorn microarray databases (Killion et al. 2003), each expression value identified by the experiment and the probe set (probe's name) is stored as a record (each Expression Value Per Record, EVPR). For

* Corresponding author. Tel: +1 843 876-1123, Fax: +1 843 876-1126, Email addresses: Lam C. Tsoi – E-mail: tsoi@musc.edu, W. Jim Zheng* – E-mail: zhengw@musc.edu.

example, if we use Y98 (Affymetrix), a chip with 9335 probe sets to study the gene expression of yeast (*Saccharomyces cerevisiae*), performing 8 experiments will yield 9335×8=74680 records in the table *DATA*. The exponential growth of microarray data will eventually makes the table extremely large and becomes unmanageable (Sarkans et al. 2005). As a result, alternative method has been proposed in ArrayExpress to store expression values in NetCDF format and keep the whole microarray data as BLOBs in the database (Sarkans et al. 2005). In the schema like SMD, for a researcher to query all the expression values of a particular probe set such as AFFX-MurIL2 from Y98, the system would have to run through each experiment done on the Y98 to find the correct probe set. On the other hand, the query has to go through the BLOBs in NetCDF format and find the right record when query ArrayExpress database, or the NetCDF has to be pre-computed to store expression values in the data warehouse. Since the experiments performed using the same chip will have the same number of records, here we propose a new schema by using the array data type to store the expression values of microarray experiments, which will increase the efficiency of space usage and data retrieval.

2. Material and Methods

Our database is implemented in PostgreSQL, an object-relational open-source database management system (DBMS) that can be freely downloaded at http://www.postgresql.org/. In order to compare the performances of EVPR and our proposed array-data type schemas, we created the tables and attributes that would be necessary for each type of schema (Results), and uploaded all the information of chip, probset, and gene, and the experimental results to both schemas. The two databases were implemented using PostgreSQL in a PC with Pentium(R) 4 CPU (2.4GHz) and 1.00GB of ram. Microarray data from eight experiments of chip Y98 were used, and another seven experiments with artificially generated experiment data (by random number) were added. We also included two additional chips (called Y99 and C50), which also have 9335 probsets, with 15 (for Y99) and 20 (for C50) sets of artificially generated experiment data. The overview of the number of records needed for each table of the two schemas is shown in Table 1. We tested the performance of our schema by comparing the query times of retrieving different types of query to that of the schema storing each expression value per record (EVPR), and the core tables for this schema is shown in Fig. 1B. The SQL commands used to test the databases' performance are shown in Table 2.

3. Results

3.1. Microarray data storing in array data type schema

PostgreSQL supports most SQL commands and different data types, including array data type. In PostgreSQL, array data type can have a variable length and allow index access, so it can be used efficiently to store data. Figure 1A shows the relational diagram of the schema, and it consists of four tables: *MICROARRAY*, *CHIP*, *GENE* and *EXPERIMENT*. In our schema, the table *MICROARRAY* stores all the expression values from a probe set as one record, so Y98 will only have 9335 records in the table. Therefore, the number of records corresponding to a chip is the same as the number of probes of that chip, regardless of the number of experiments performed using the chip. The data type of the attribute MICROARRAY_DATA is a one-dimensional array that stores the expression values from all experiments done on the chip. Therefore, if we have 8 experiments done on a chip, each record array for the attribute MICROARRAY_DATA will have 8 values, one from each experiment. We would expect different chips to be used different numbers of time, and the PostgreSQL can accommodate this by allowing attributes of a table to be defined as a variable-length array data type. As a result, the array for MICROARRAY_DATA for each record in the table will have different array lengths. In Figure 1C, the left table shows 3 example records from table *MICROARRAY*. The first two records are from two probes of Y98. If there were 8 experiments done on this chip, there are 8 array entries in the attribute MICROARRAY_DATA. Let Y99 be another chip we have in our database, and it has a probe named Y99_probe_1. Assuming only 3 experiments were done on this chip; we thus have 3 array entries in MICROARRAY_DATA for the Y99's record.

In Figure 1C, the right table shows a portion of the table *EXPERIMENT*. Each record in the table *EXPERIMENT* is defined by the primary key EXPERIMENT_ID, so the first 8 records from this table correspond to chip Y98. In this table, the attribute INDEX could be understood as the “index number” of a particular experiment data on MICROARRAY_DATA, and is used to retrieve the corresponding array entry. The 8 records in the table *EXPERIMENT* would have the INDEX filled with 0-7 respectively (since the array entry starts with 0). If a researcher wants to retrieve all the data from an experiment using Y98, then the system will use the INDEX and CHIP_ID (i.e. Y98) from the table *EXPERIMENT* to query the corresponding entry of MICROARRAY_DATA in the table *MICROARRAY*. For example, if the researcher wants to query all the expression values in the Y98's experiment Wt-4-2. The SQL command will be:

select (MICROARRAY. PROBSET_ID, MICROARRAY. MICROARRAY_DATA [EXPERIMENT. INDEX]) from MICROARRAY, EXPERIMENT where MICROARRAY. CHIP_ID = EXPERIMENT. CHIP_ID and EXPERIMENT. EXP_ID= ‘Wt-4-2’;

Note that we now have “one to many” relationships for tables pairs CHIP--MICROARRAY and CHIP--EXPERIMENT. This is because each record in the table *MICROARRAY* is derived from one record in CHIP, but is composed of many experiments. Also, each record in the table *EXPERIMENT* uses one chip, and it appears in many records in the table *MICROARRAY*.

3.2. Practical usages of array data type schema

Another useful query is to identify all the genes in the database that has an expression value greater than a threshold. The data can be queried out as follow:

Select MICROARRAY.CHIP_ID, PROBSET_ID, EXP_ID, GENE_ID, MICROARRAY_DATA[EXPERIMENT.INDEX] from MICROARRAY, EXPERIMENT where MICROARRAY.CHIP_ID=EXPERIMENT.CHIP_ID and MICROARRAY_DATA [EXPERIMENT.CHIP_ID] > x.

In this case all the genes with expression value greater than x will be selected. The uniqueness of each entry is identified by the combination of CHIP_ID, EXPERIMENT_ID and PROBSET_ID.

The removal of records in the database is also straightforward. Since the entries in the array are only meaningful with valid index and experiment information from the *EXPERIMENT* table. Deleting an entry in the *EXPERIMENT* table will wipe out the possibility of using the corresponding expression value stored in the array data type in the *MICROARRAY* table. An additional auxiliary table can be used to keep track of deleted index value for each chip type. When a new experiment result enters into the database, the entry can recycle the removed index in the *EXPERIMENT* table, and update the value in the corresponding field of array data type in the *MICROARRAY* table. Therefore, only minimum operation is needed and it won't affect any other entries in either the *MICROARRAY* or *EXPERIMENT* table.

3.3. Efficiency in querying data

The result of the query time is shown in fig. 2, and it suggests that our schema is more efficient when probe annotations are included in the query results (Query 1, 3, 4b), which are common when doing microarray analysis. Query 3b retrieve the expression values that are greater than a threshold (without the probe annotation), and the array type schema (although it has to go through iterations in MICROARRAY_DATA) is almost as efficient as in EVPR (which is just searching record by record). The data we can retrieve from query 4 is well-suited for doing microarray analysis from different experimental data, since every row is showing all the experimental values of a given probe. Although the index of each experiment has to be known before doing this query, researchers would not need to structure the output again as in the query 4b. However, it is not easy for EVPR to produce data directly like this as in query 4.

4. DISCUSSION

The implementation of the microarray database using our proposed model requires the DBMS to support array data type. Out of the most commonly used DBMS of microarray databases, SQL server and MySQL do not support this data type while PostgreSQL and Oracle now support variable-length array. By using array data type to store all the experimental results of the probe, the storage size and the performance of the database can be greatly reduced by keeping annotation and expression value in the same table. This increased efficiency makes this approach highly suitable for a research community that wants to develop specific microarray database at small or large scale. Processing queries will also be more efficient since the number of records in the table *MICROARRAY* is minimized, and accessing the array by index is very fast (Fig. 2). Also, the variable-length array property supported by the DBMS is well-suited to our model since we can allow chips with different demands (number of experiments done on the chip) to be stored in the same database. These measures are important for microarray databases, since the use of such databases typically involves retrieving large amounts of data by joining different tables for further analysis. In addition, storing normalized and analyzed data in our relational model will allow flexible comparisons across different chips or platforms at individual gene levels.

As pointed out by Stoeckert et al. (Stoeckert et al. 2002), it is essential for researchers to have a detailed understanding of the database structures if they want to import the data from other databases to their local database. Our proposed relational schema is easy to understand, as the major tables only consist of *CHIP*, *GENE*, *EXPERIMENT*, and *MICROARRAY*. Each record in the table *MICROARRAY* can stand alone as a unique probe set (independent of the experiments done on it), and the attribute INDEX acts as a key feature in our model to link the table *EXPERIMENT* and *MICROARRAY*. For chips that have existing records, new data can be uploaded to the database by adding new entries to MICROARRAY_DATA.

In the EVPR model, the common primary keys for table that stores the expression data are the EXPERIMENT_ID and PROBESET_ID, which defines one expression value in each record. Since it is redundant to store the descriptions of probes in this table, table *PROBE* (see Fig. 1B) would be needed to store all the probes' descriptions. This model slows down the retrieval process if we query all the readings and descriptions of the probes in a chip (see Fig. 2 queries 1, 3, 4b), since it requires information from two tables (Data and Probe), and the computer system has to search and join the information for each record. On the contrary, we store the expression values and the descriptions of probes in the same table in our relational model. Therefore, the system does not have to go through the joining process, and this effectively increases the querying speed.

Acknowledgments

We would like to thank Chuming Chen for providing the technical support, Annie Simpson and Dr. Wenle Zhao for the valuable advice and suggestions. We also would like to thank Dr. Ashley Cowart for providing microarray data to populate our relational model in PostgreSQL. This work was partly supported by grants GC-3609-04-43766CM, and by a Hollings Cancer Center/Medical University of South Carolina Department of Defense grant “Translational Research on Cancer Control and Related Therapy” (Subcontract GC-3319-05-4498CM). L.C.T. is supported by NLM training grant 5-T15-LM007438-02.

References

Ball CA, Awad IA, Demeter J, Gollub J, Hebert JM, Hernandez-Boussard T, Jin H, Matese JC, Nitzberg M, Wymore F, Zachariah ZK, Brown PO, Sherlock G. The Stanford Microarray Database accommodates additional microarray platforms and data formats. Nucleic Acids Res 2005;33:D580–582. [PubMed: 15608265]

Cheung KH, White K, Hager J, Gerstein M, Reinke V, Nelson K, Masiar P, Srivastava R, Li Y, Li J, Zhao H, Li J, Allison DB, Snyder M, Miller P, Williams K. YMD: a microarray database for large-scale gene expression analysis. Proc AMIA Symp 2002:140–144. [PubMed: 12463803]

Galperin MY. The Molecular Biology Database Collection: 2006 update. Nucleic Acids Res 2006;34:D3–5. [PubMed: 16381871]

Killion PJ, Sherlock G, Iyer VR. The Longhorn Array Database (LAD): an open-source, MIAME compliant implementation of the Stanford Microarray Database (SMD). BMC Bioinformatics 2003;4:32. [PubMed: 12930545]

Sarkans U, Parkinson H, Lara G, oezcimen A, Sharma A, Abeygunawardena N, Contrino S, Holloway E, Rocca-Serra P, Mukherjee G, Shojatalab M, Kapushesky M, Sansone S, Farne A, Rayner T, Brazma A. The ArrayExpress gene expression database: a software engineering and implementation perspective. Bioinformatics 2005;21:1495–1501. [PubMed: 15564302]

Sherlock G, Hernandez-Boussard T, Kasarskis A, Binkley G, Matese JC, Dwight SS, Kaloper M, Weng S, Jin H, Ball CA, Eisen MB, Spellman PT, Brown PO, Botstein D, Cherry JM. The Stanford Microarray Database. Nucleic Acids Res 2001;29:152–155. [PubMed: 11125075]

Stoeckert CJ Jr, Causton HC, Ball CA. Microarray databases: standards and ontologies. Nat Genet 2002;32(Suppl):469–473. [PubMed: 12454640]

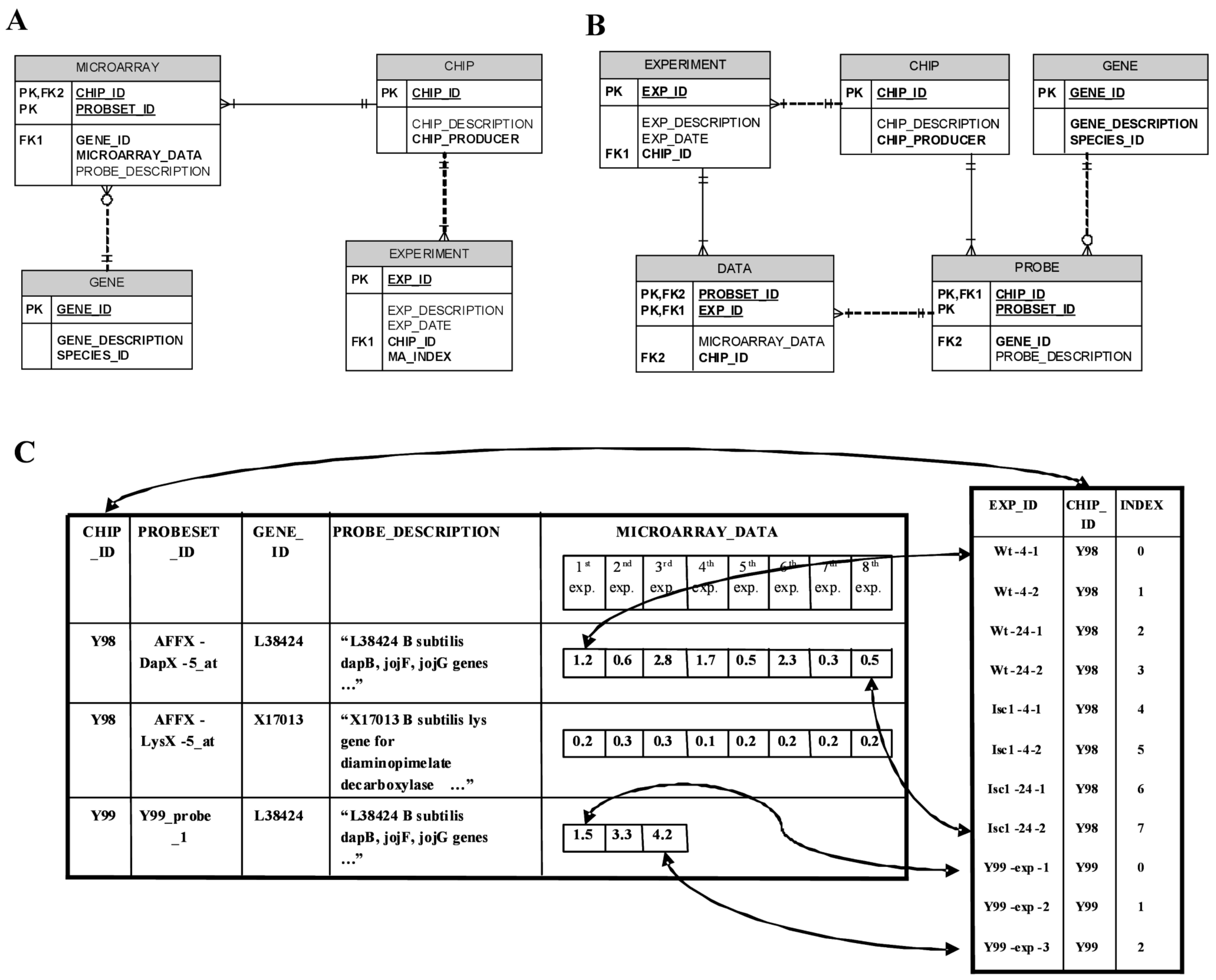

Figure 1. Using array data type to store microarray data

The top shows the schemas of the microarray databases: A) is the schema that uses array data type; B) is the schema that stores expression value per row (EVPR) C) A diagram to illustrate how microarray data are stored in the array data type schema. The left table is the table *MICROARRAY*, and the table *EXPERIMENT* is on the right. In the table *MICROARRAY*, two example probes from chip Y98 and one example probe from chip Y99 are shown. The attribute MICROARRAY_DATA is an array data type, with variable array length. The table *EXPERIMENT* shows the eight experiments done on Y98 and three experiments done on Y99. The attributes EXP_DESCRIPTION and EXP_DATE are not shown here. The arrows show the relationship between the two tables. For the probes from chip Y98, the first entries in MICROARRAY_DATA correspond to the experiment Wt-4-1; and for Y99 the third entries of MICROARRAY_DATA are the expression data from experiment Y99-exp-3.

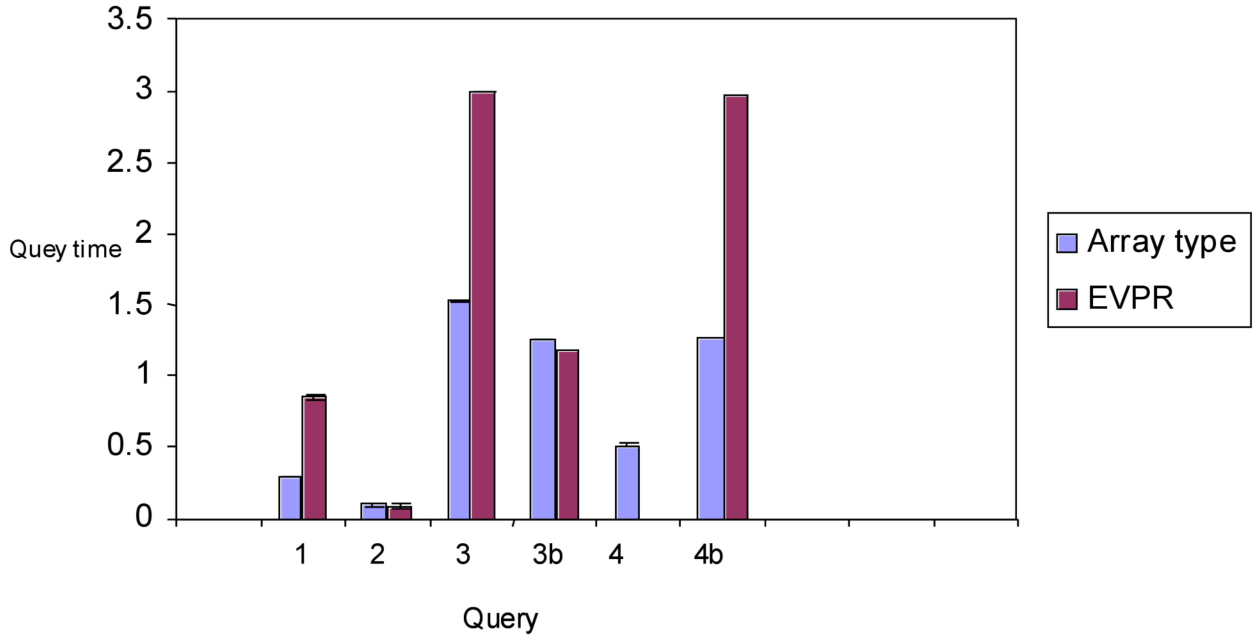

Figure 2. Performance comparisons between the two schemas

The query time needed for each type of query are plotted to direct compare the performance of two schemas. The array based schema is either comparable to or better than the EVPR based schema. The core tables used and the number of rows in each table are shown in Table 1, and the types of queries (1, 2, 3, 3b, 4 and 4b) are shown in Table 2. Each query was performed 5 times and the standard errors were calculated.

Table 1

Simulated data in the PostgreSQL for two different schemas

The core tables in each of the schema and the number of records needed to store the data are shown

	Table	Number of Rows
Expression Per Row	CHIP	3
	PROBE	28005
	EXPERIMENT	50
	GENE	6776
	DATA	466750
Array type	CHIP	3
	EXPERIMENT	50
	GENE	6776
	MICROARRAY	28005

Table 2
SQL queries used to compare the performance of two schemas

Query	SQL for Array Type	SQL for EVPR
1. Retrieve the (chip_id, probset_id, gene_id, expression values, probe_description, experiment_id) of given genes over ALL experiments:	select (microarray.chip_id, probset_id, gene_id, microarray_data[index], probe_description, exp_id) from microarray, experiment where (gene_id=’YAL005C’ or gene_id=’YAL001C’ or gene_id=’NC_001142’) and microarray.chip_id=experiment.chip_id;	select (smd_data.chip_id, smd_data.probset_id, gene_id, microarray_data, probe_description, exp_id) from smd_data, smd_probe where (smd_probe.gene_id=’YAL005C’ or smd_probe.gene_id=’YAL001C’ or smd_probe.gene_id=’NC_001142’) and smd_probe.probset_id=smd_data.probset_id and smd_probe.chip_id=smd_data.chip_id;
2. Retrieve the (chip_id, probset_id, gene_id, expression values, probe_description, experiment_id) over 7 experiments (of chip Y98) where the probe_description has the key word “protein membrane”	select (microarray.chip_id, probset_id, gene_id, microarray_data[index], probe_description, exp_id) from microarray, experiment where exp_id like ’Y98-Time-%’ and probe_description like ’%membrane protein%’ and microarray.chip_id=experiment.chip_id;	select (smd_data.chip_id, smd_data.probset_id, gene_id, microarray_data, probe_description, exp_id) from smd_data, smd_probe where exp_id like ’Y98-Time-%’ and probe_description like ’%membrane protein%’ and smd_probe.probset_id=smd_data.probset_id and smd_probe.chip_id=smd_data.chip_id;
3. Retrieve the probe that has expression value greater than a threshold	select (microarray.chip_id, probset_id, microarray_data [index], probe_description, exp_id) from microarray, experiment where microarray.chip_id=experiment.chip_id and microarray_data[index] > 1500;	select (smd_data.chip_id, smd_data.probset_id, microarray_data, probe_description, exp_id) from smd_data, smd_probe where microarray_data > 1500 and smd_probe.probset_id=smd_data.probset_id and smd_probe.chip_id=smd_data.chip_id;
3b. Retrieve the probe that has expression value greater than a threshold (the display excluds probe_description)	select (microarray.chip_id, probset_id, microarray_data [index], exp_id) from microarray, experiment where microarray.chip_id=experiment.chip_id and microarray_data[index] > 1500;	select (chip_id, probset_id, microarray_data, exp_id) from smd_data where microarray_data > 1500;
4. To retrieve 7 experiments data from a chip for microarray analyses (each row shows the expression values of a probe from all 7 experiments). It requires the users to get the index of the experiments first.	select (chip_id, probset_id, microarray_data[8], microarray_data[9], microarray_data[10], microarray_data[11], microarray_data[12], microarray_data[13], microarray_data[14], probe_description) from microarray where chip_id=’Y98’;	Not available
4b. To retrieve 7 experiments data from a chip for microarray analyses (each row represents one expression record)	select (microarray.chip_id, probset_id, microarray_data [index], probe_description, exp_id) from microarray, experiment where microarray.chip_id=’Y98’; and exp_id like ’Y98-Time-%’;	select (smd_data.chip_id, smd_data.probset_id, microarray_data, probe_description, exp_id) from smd_data, smd_probe where smd_data.chip_id=smd_probe.chip_id and smd_data.probset_id=smd_probe.probset_id and exp_id like ’Y98-Time-%’;